

Demon Debugger

VCF Midwest - 10-September-2022

Frank Palazzolo

Evan Allen

Introduction and Questions...

? ? ? ? ? ? ? ? ?

? ? ? ? ? ? ? ? ?

What is Demon Debugger?

Demon Debugger an flexible, open tool, used to help repair and reverse-engineer old computers*

*computers = computers, game consoles, arcade machines, SBCs, etc.



Demon Debugger - 2016

Repairing a Star Trek arcade machine

Built by Sega in 1982, purchased it in 1999

Where do I start?

Initial Idea - Can I use it to debug itself?

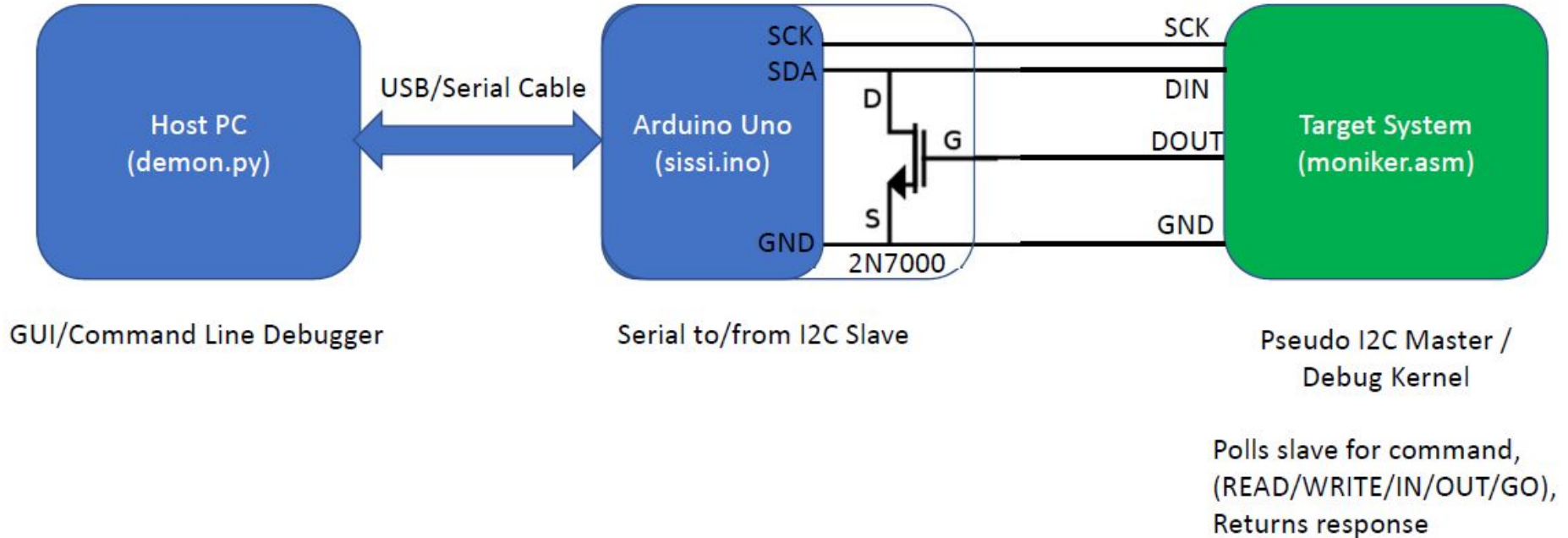
If it can run code, and has a little RAM:

- 1) Run my own Z80 code on this
- 2) Find a few I/O Pins that I can use to make it talk to me



Demon Debugger - Tethered mode

DEMON DEBUGGER System Architecture



Demon Debugger - 2016

Target code:

Polling loop

Bit-Bang I2C Master* for Z80

(Based on Wikipedia code)

Command Handler

Read from Address/Port

Write from Address/Port

Call and Address

Example of bit-banging the I²C protocol [\[edit\]](#) [\[edit source\]](#)

Below is an example of [bit-banging](#) the I²C protocol as an I²C controller. The example is written in [p](#)ack/nack).^[32]

```
1 // Hardware-specific support functions that MUST be customized:
2 #define I2CSPEED 100
3 void I2C_delay(void);
4 bool read_SCL(void); // Return current level of SCL line, 0 or 1
5 bool read_SDA(void); // Return current level of SDA line, 0 or 1
6 void set_SCL(void); // Do not drive SCL (set pin high-impedance)
7 void clear_SCL(void); // Actively drive SCL signal low
8 void set_SDA(void); // Do not drive SDA (set pin high-impedance)
9 void clear_SDA(void); // Actively drive SDA signal low
10 void arbitration_lost(void);
11
12 bool started = false; // global data
13
14 void i2c_start_cond(void) {
15     if (started) {
16         // if started, do a restart condition
17         // set SDA to 1
18         set_SDA();
19         I2C_delay();
20         set_SCL();
21         while (read_SCL() == 0) { // Clock stretching
22             // You should add timeout to this loop
23         }
24
25         // Repeated start setup time, minimum 4.7us
26         I2C_delay();
27     }
28 }
```

Demon Debugger - 2016

Who will I talk to? Arduino Uno

Already talks I2C!

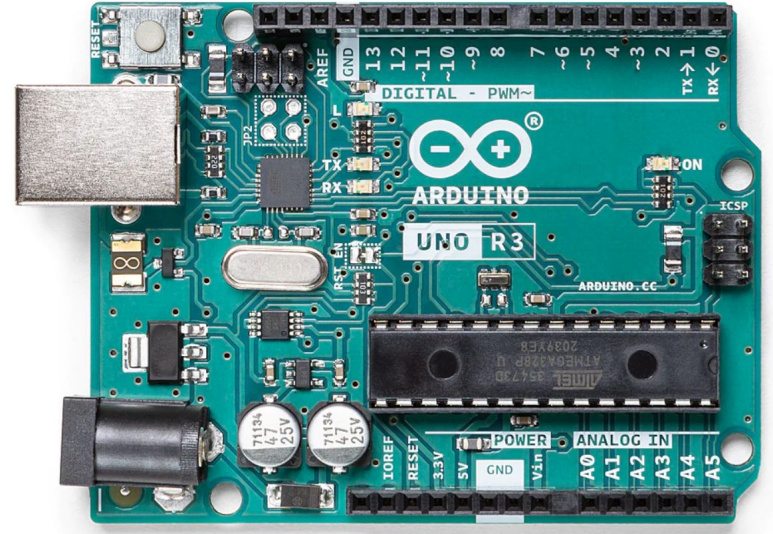
Arduino Uno code:

- Respond to Polls over I2C Slave interface

- Respond to USB Serial commands from PC

PC code:

- Command-line interface written in Python



Demon Debugger - 2016

Small Hardware Issue - I2C is a bidirectional bus.

I didn't want target to require bidirectional pin on the target

Use a single transistor to adapt:

Real I2C (2-wires + ground)

"Pseudo-I2C" (3-wires + ground)



Star Trek Repair successful!

...but it took a while

Demon Debugger was key in all steps, along with information from MAME

Repaired each board in the card cage

Repaired Vector Display

Z80 code is fairly easy to adapt to another platform, I/O and RAM



Atari Asteroids Repair

Ported the target code from Z80 to 6502

(subset of functions, no Port I/O)

No issues, worked like a champ!

Note: 6502 code is even easier to adapt to other hardware, uses page zero RAM

Now Demon Debugger is Multi-platform!



A 3rd Platform?



Tutorvision BIOS dump

How can I dump the Tutorvision BIOS?

CPU is a GI CP1610

LTO-Flash cart can run code, and it has a serial port.

Run the serial part of Demon Debugger on the cart

Did this, and the BIOS was dumped

3rd platform - sort of!



What if we just don't have the I/O?

Pristine, rare computer!

ROMs to dump on the inside

Cartridge port is compatible with the
ColecoVision (Z80)



What if we just don't have the I/O?

(ColecoVision Cart has 3 ROMs)

We can put Demon Debugger target code into ROM1.

Idea:

What if we turn read accesses to ROM2 and ROM3 into read and write signals to the Arduino?



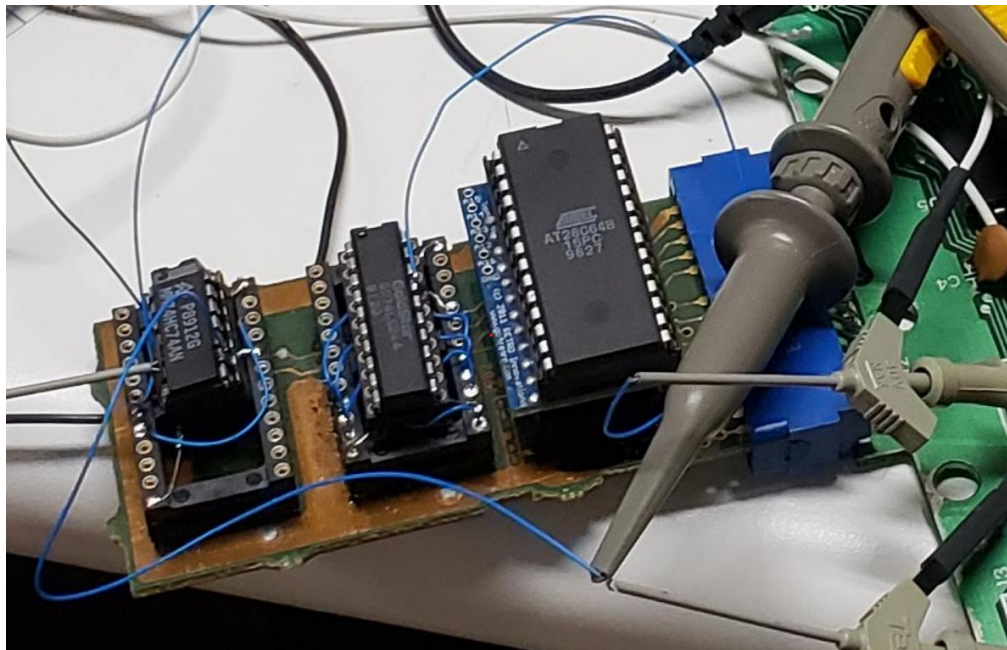
Birth of Demon Debugger - Memory Mapped I/O

Reads to ROM1 will read DD code

Reads to ROM2 will read D0

Reads to ROM3 will write A1/A0

Uses only a few chips, works great!



Birth of Demon Debugger - Memory Mapped I/O

New way of using Demon Debugger

Replace a 2K memory region

Compatible with a 2716 ROM

Code plus memory mapped I/O

Replace the ROM, I/O is included!

Note:

I/O in middle, can you guess why?

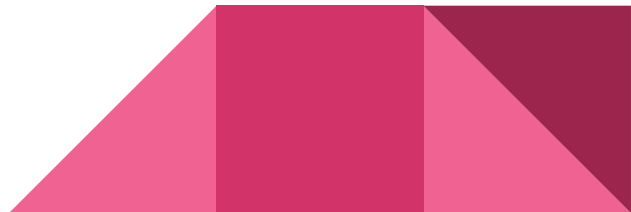
Memory Map:

0x000-0x1ff Available for Code

0x200-0x3ff Available for Code

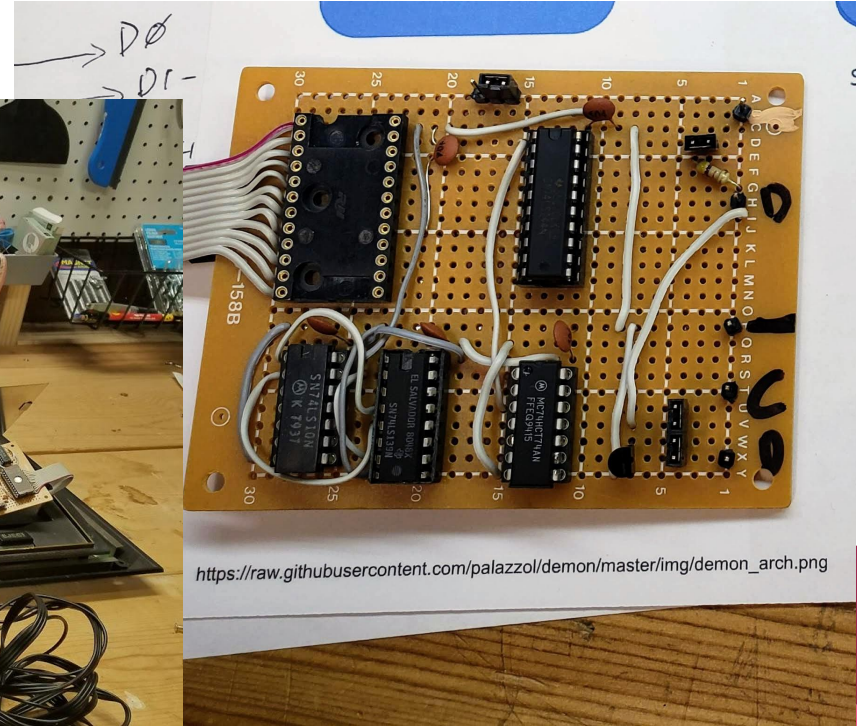
0x400-0x5ff I/O Region

0x400-0x5ff Available for Code



Birth of Demon Debugger - Memory Mapped I/O

Tested on a Bally Astrocade cartridge



Time for a PCB! Rev A

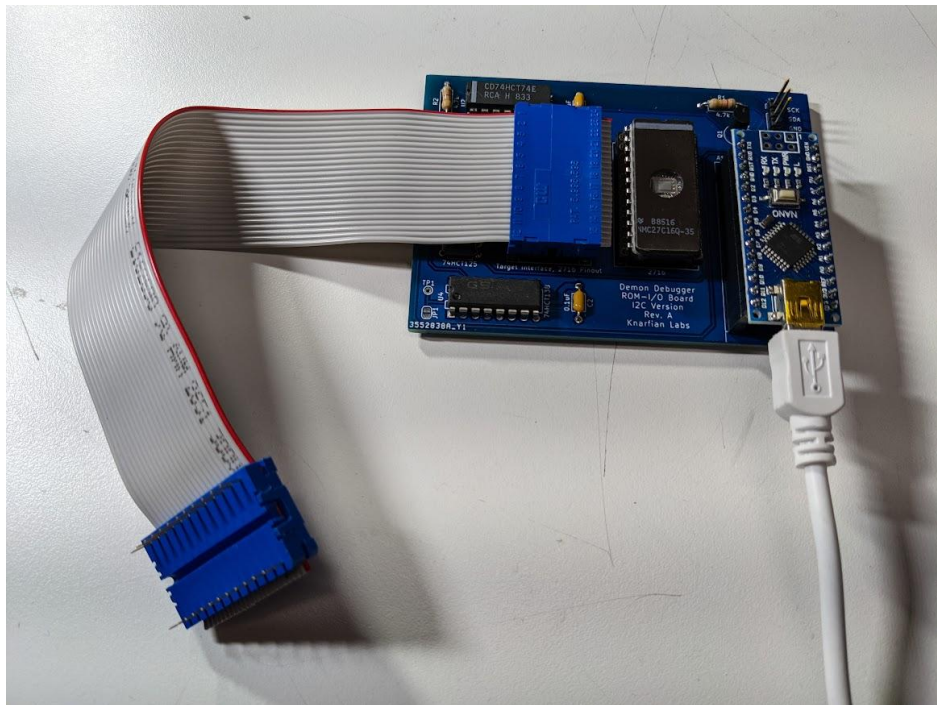
Memory Map I/O

Onboard EPROM chip

Cable to 2716 standard ROM header

Can still support tethered mode

What could go wrong?



Delayed edges, and other horrors

It doesn't work consistently! It needs one mod for a Z80, another for 6502!

The issue is using the hack of using Address Lines as Data to write to the Arduino. Address Lines are not easy to grab based on \sim OE and \sim CE states.

Solution:

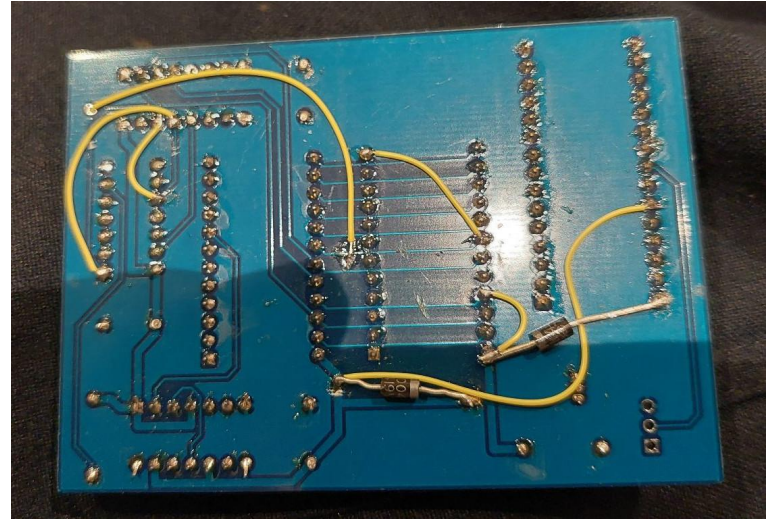
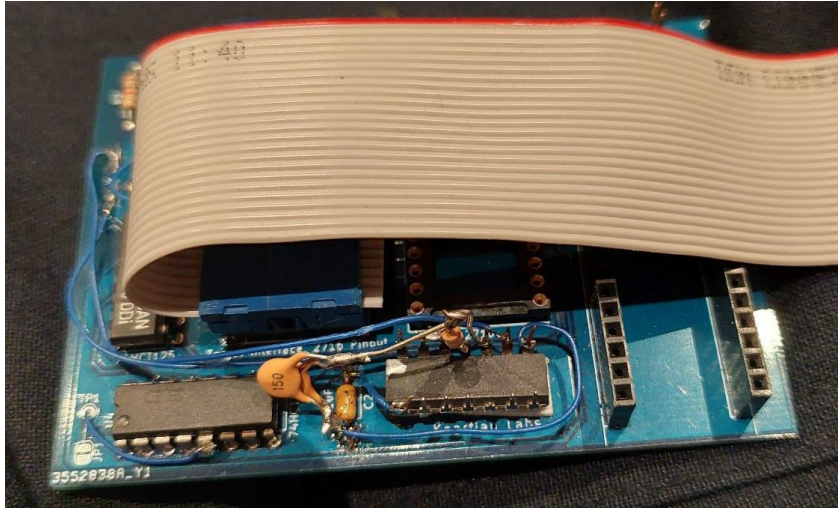
We already have a memory on board. When you want to write, just read from the I/O write location, but also let the memory provide the data to write.

Now all platforms can do IO exactly the same way!



Demon Debugger Hardware - Rev. B?

Hack up a Rev A board with the major fix, (and minor fixes) it works!!!



Anything else we want to add while we're at it...?

Demon Debugger - Reprogrammability

We now can use an onboard 2716 EPROM, or have a 28C16 (2K Flash chip)

But to use it, it still requires a chip programmer.

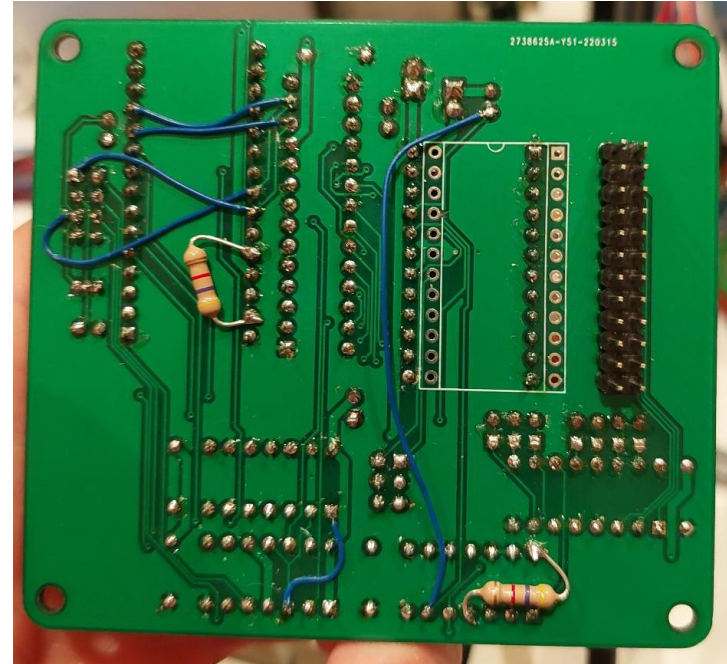
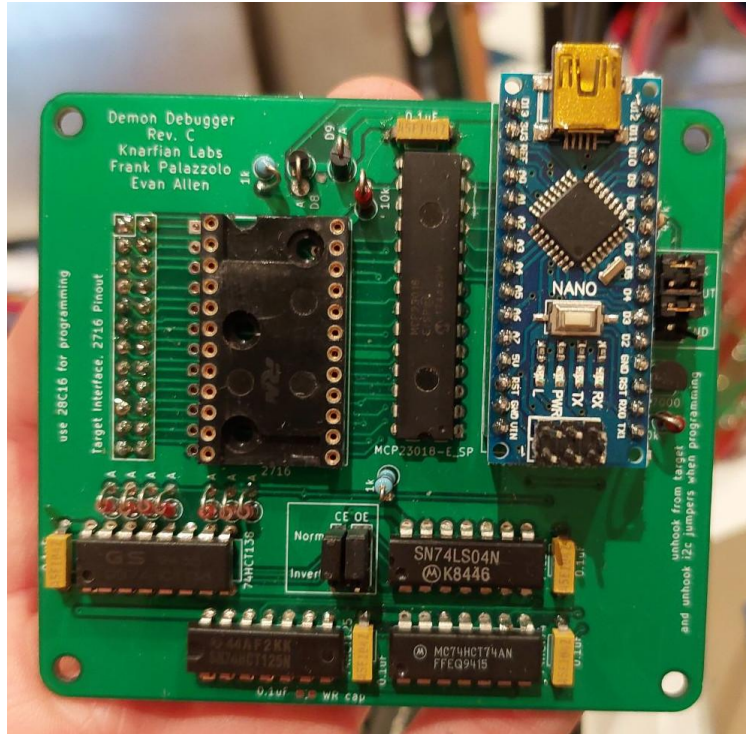
More convenient to expand the design to allow for reprogramming the Flash!

Rev C:

Add I/O expander to handle programming



Demon Debugger Hardware - Rev. C



Demon Debugger 2022 - Rev D

More fixes

- Need 2 separate I2C interfaces

- Other bug fixes

New Features

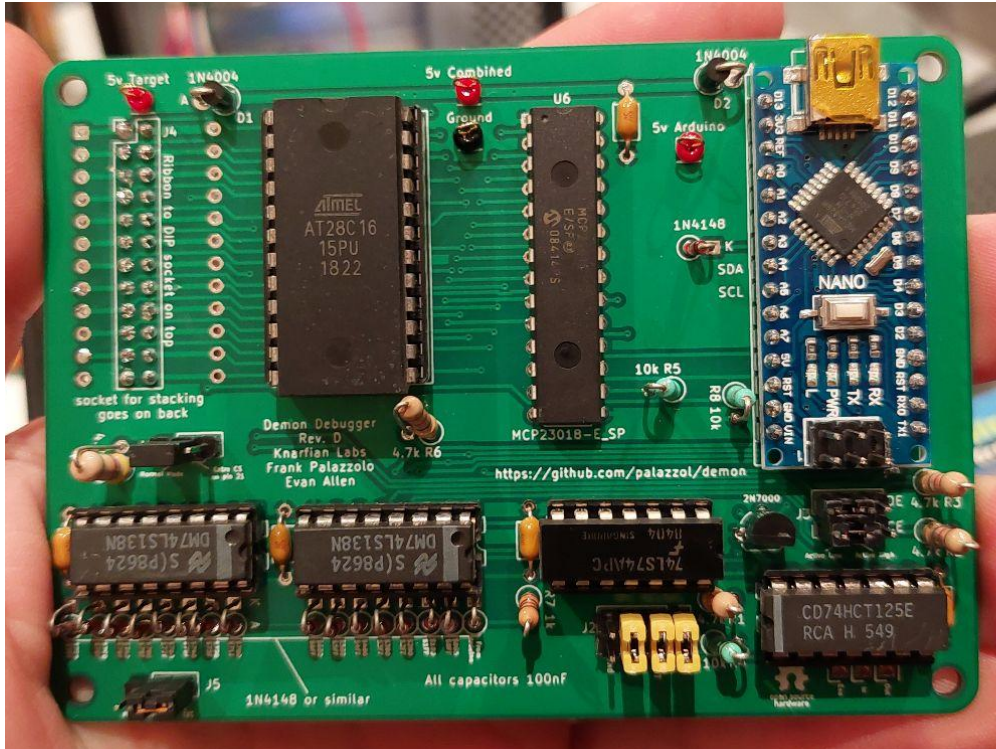
- We decide to shrink and move the I/O regions

- Maximize useable code region

- Support tiny roms



Demon Debugger 2022 - Rev D



Memory Map

0x000-0x7bf Available

0x7a0-0c7bf Read Arduino

0x7c0-0x7df Write Arduino

0x7e0-0x7ff Available

Demon Debugger GUI

New software necessary for reflashing,
might as well start on a GUI now

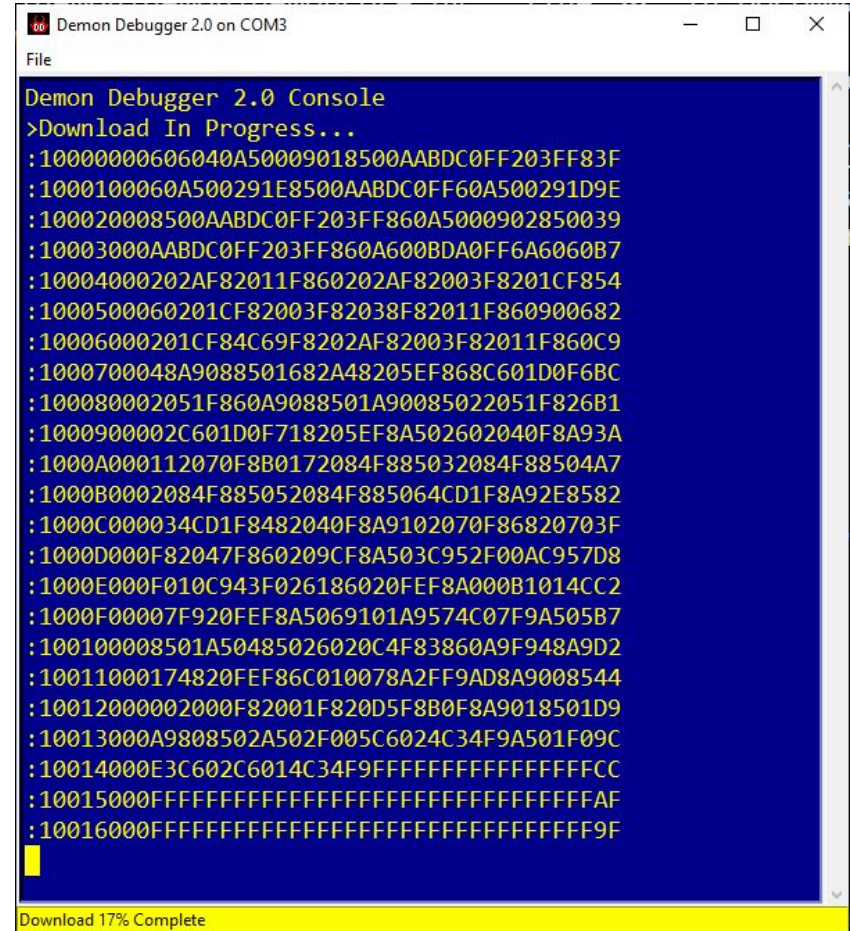
GUI uses Python + Tk (built in)

Current Status:

GUI upload/download code to flash

CLI program still used for the rest

2.0 Release - Make the GUI fully functional

The image shows a screenshot of the 'Demon Debugger 2.0 on COM3' application window. The window has a standard Windows title bar with minimize, maximize, and close buttons. Below the title bar is a menu bar with 'File'. The main area is a console window with a dark blue background and yellow text. The text in the console reads: 'Demon Debugger 2.0 Console', '>Download In Progress...', followed by a long list of hexadecimal data in columns. At the bottom of the console, there is a yellow progress bar and the text 'Download 17% Complete'.

```
Demon Debugger 2.0 Console
>Download In Progress...
:10000000606040A50009018500AABDC0FF203FF83F
:1000100060A500291E8500AABDC0FF60A500291D9E
:100020008500AABDC0FF203FF860A5000902850039
:10003000AABDC0FF203FF860A600BDA0FF6A6060B7
:10004000202AF82011F860202AF82003F8201CF854
:1000500060201CF82003F8203F82011F860900682
:10006000201CF84C69F8202AF82003F82011F860C9
:1000700048A9088501682A48205EF868C601D0F6BC
:100080002051F860A9088501A90085022051F826B1
:1000900002C601D0F718205EF8A502602040F8A93A
:1000A000112070F8B0172084F885032084F88504A7
:1000B0002084F885052084F885064CD1F8A92E8582
:1000C000034CD1F8482040F8A9102070F86820703F
:1000D000F82047F860209CF8A503C952F00AC957D8
:1000E000F010C943F026186020FEF8A000B1014CC2
:1000F00007F920FEF8A5069101A9574C07F9A505B7
:100100008501A50485026020C4F83860A9F948A9D2
:10011000174820FEF86C010078A2FF9AD8A9008544
:1001200002000F82001F820D5F8B0F8A9018501D9
:10013000A9808502A502F005C6024C34F9A501F09C
:10014000E3C602C6014C34F9FFFFFFFFFFFFFFFFFCC
:10015000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFAF
:10016000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF9F
```

Demon Debugger - Python API

Can be used for automated data acquisition, for example

Target reads and writes along with other acquisitions:

- Rigol Scope used for Data Acquisition

- Rigol Scope screen caps

- Logic Analyzer

- RGB color sensor



Demon Debugger - Software structure

Assembly code: target.asm

"Startup" Code - Can specialize!

Loop:

Get Commands over I2C* (romio or tether)

If command: Read/Write/Run,

Reply over I2C* (romio or tether)

"Every" Code - Can specialize!

Delay a bit

GOTO Loop

Arduino code: I2C Slave / Serial converter

If Serial command:

buffer it for I2C

If I2C command:

Read buffer, send serial if needed

Send buffer back over I2C

Python CLI or GUI:

UI - Command/Response

over Serial

Demon Debugger - Building a Target

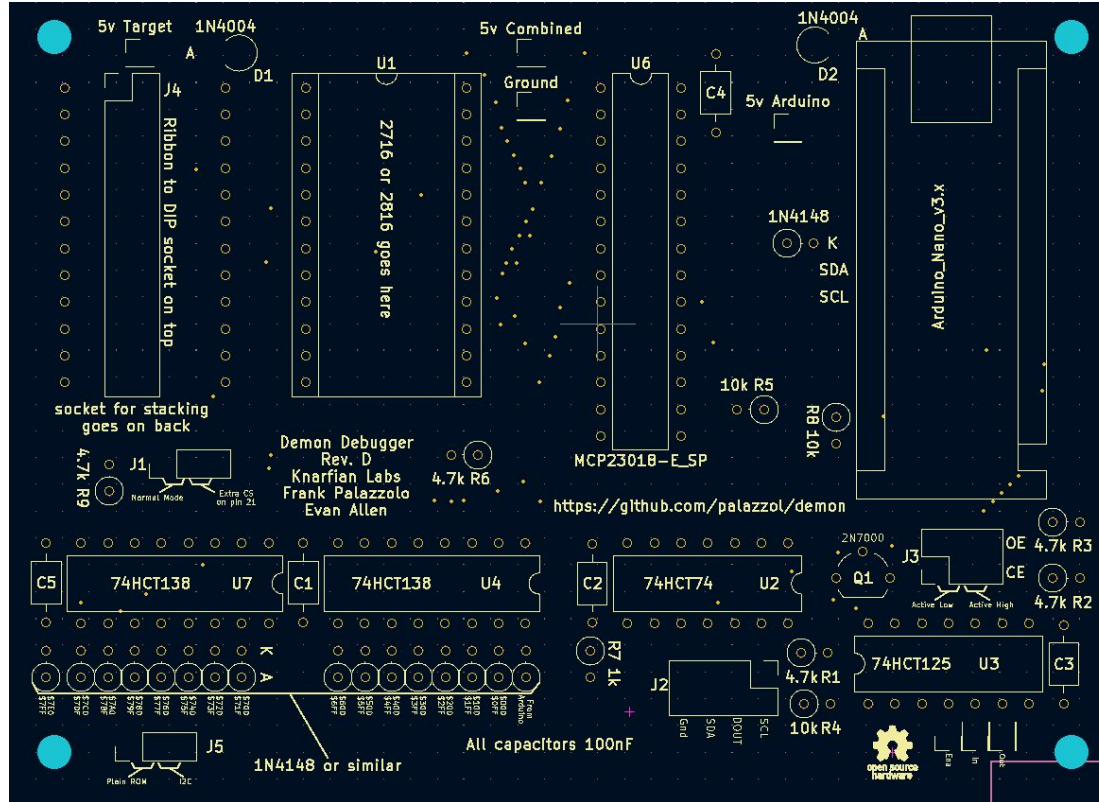
Python and TOML files

Look like old school .INI files, meant for human editing

ddmake.py tools generates new targets from templates and config files

[illegible]

Demon Debugger - Rev D. Hardware Features



Demon Debugger - Rev D. Hardware Features



Demon Debugger - The Future

More CPUs - 8085, 8080, RCA1802, TMS9900, CP1610

More Platforms - Test and deploy on Windows, Linux and Mac

Keep improving system for Target code

More GUI - Better GUI - Memory windows?

More Users!

Rev. E? - Raspberry Pi Pico?



Thank You!

Repo link - github.com/palazzol/demon

Evan Allen - abzman2k.wordpress.com

Frank Palazzolo - avoidspikes.blogspot.com