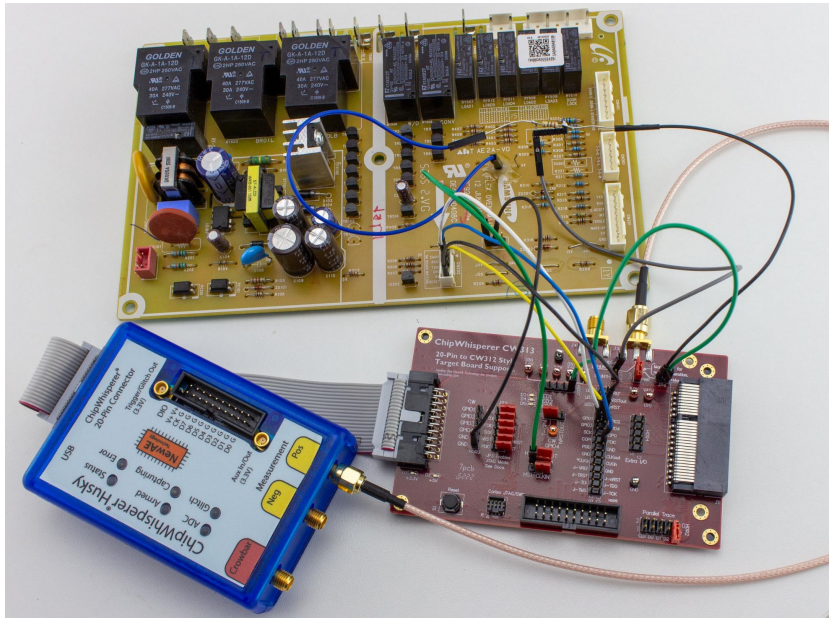
Adventures of my Oven (Pinocchio) & ChipWhisperer

Colin O'Flynn



Nova Scotia

This Halifax-area man's oven caught fire while making turkey dinner

Technician determined the stove's relay switch malfunctioned on 5-year-old range

Pars

77 comments

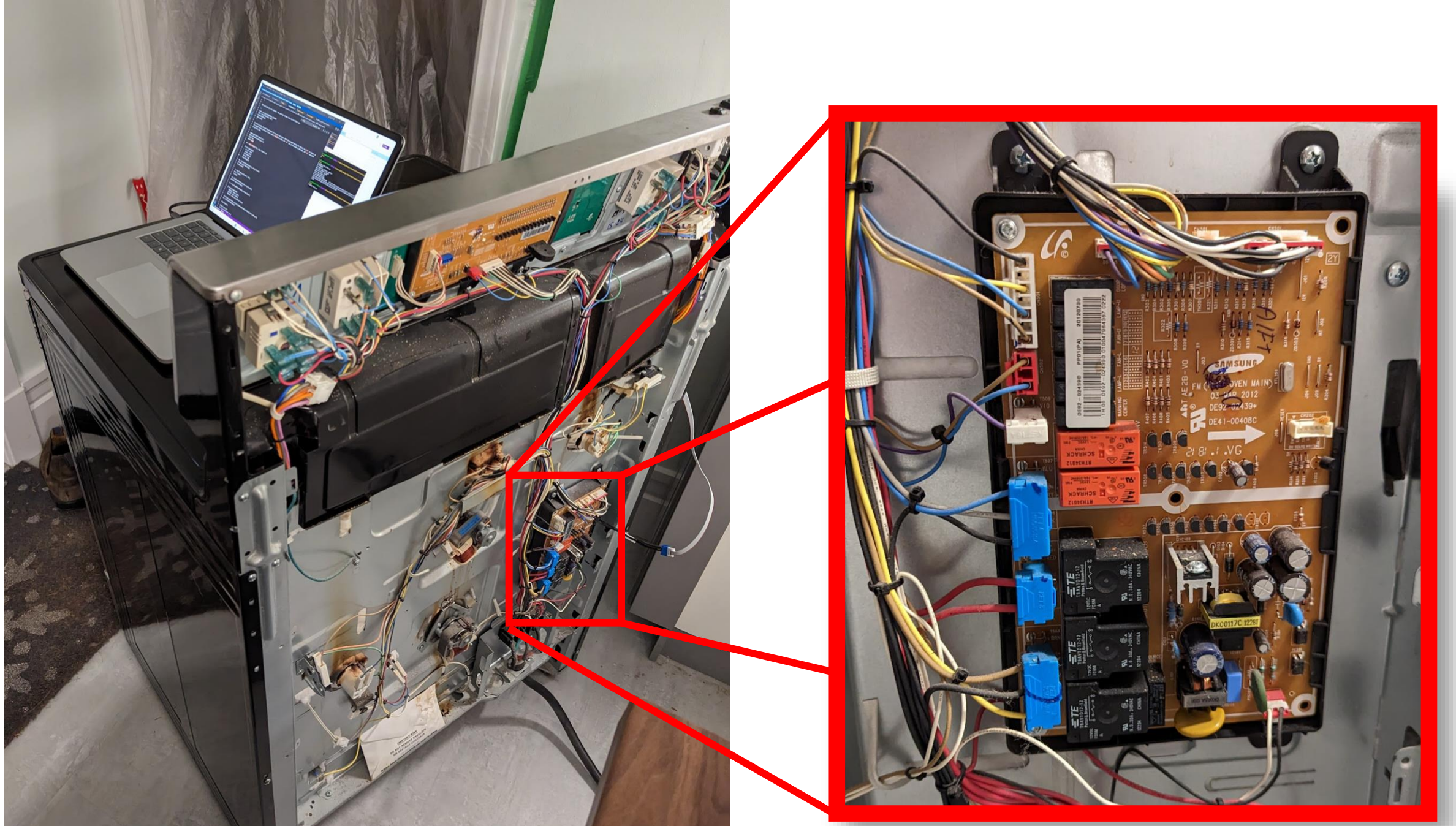
Rodney Parsons's Thanksgiving dinner turned into disaster this fall after his daughter discovered their range stove was on fire.

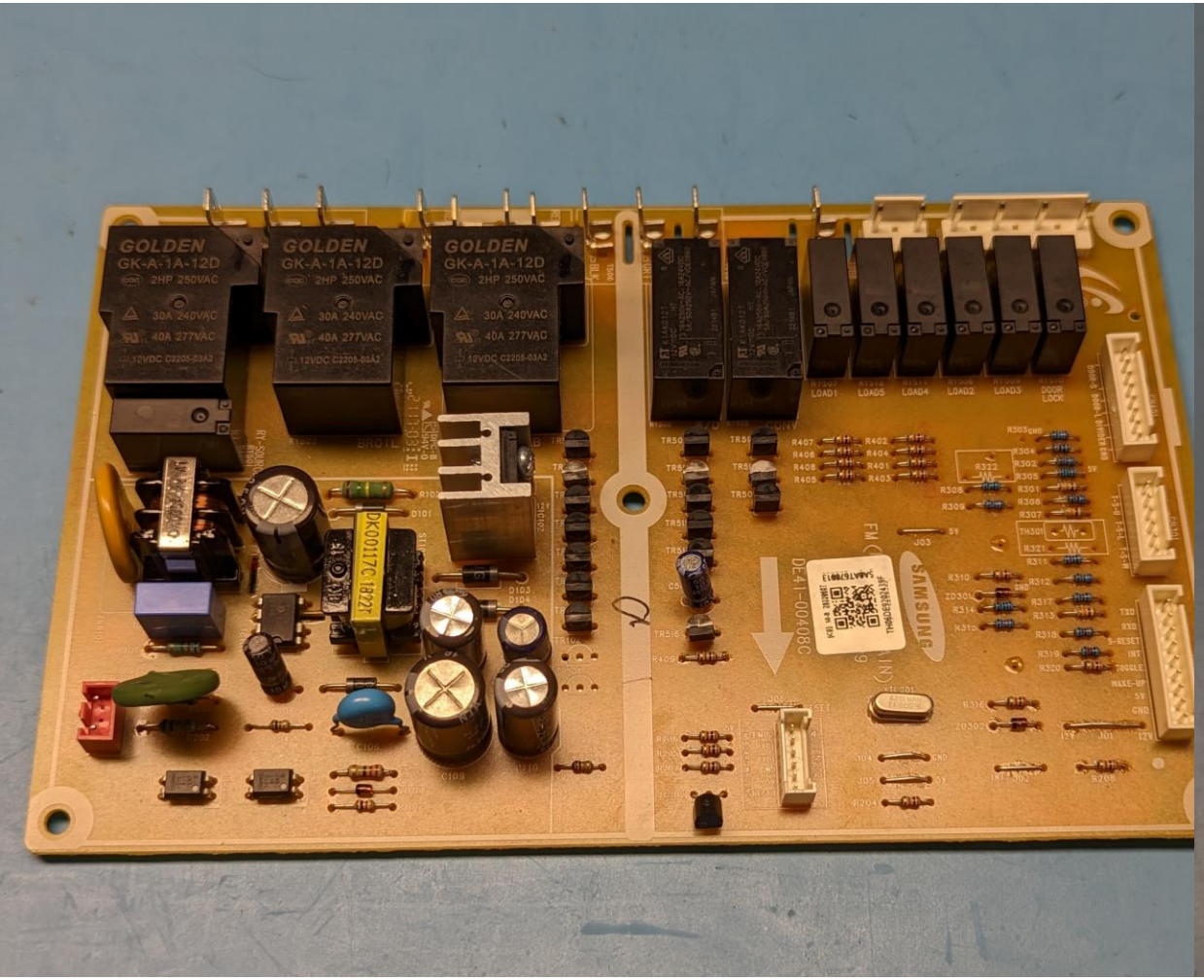
Company embroiled in lawsuit

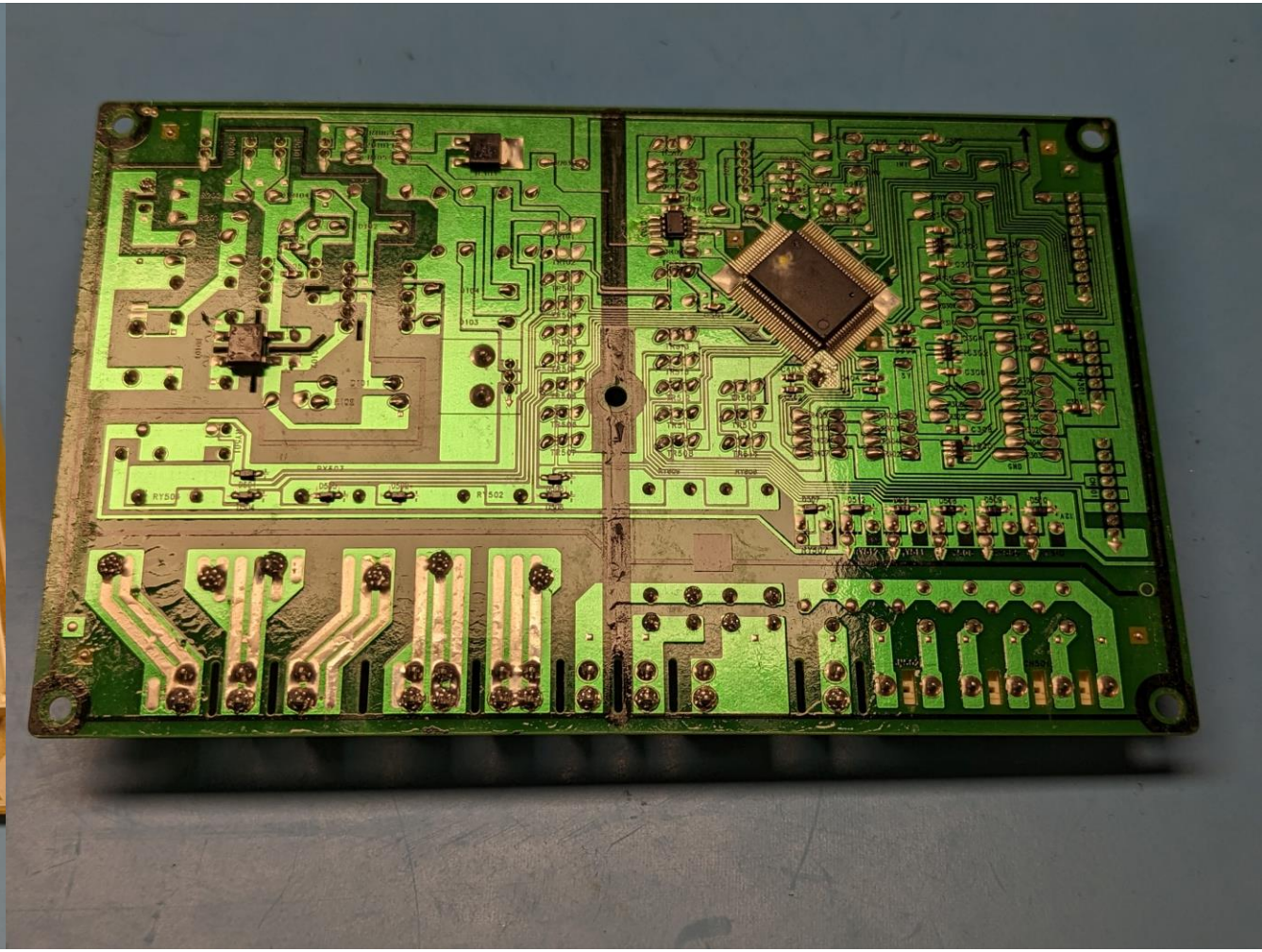
Samsung is the subject of a class action lawsuit filed in December 2020 in New Jersey pertaining to 87 Samsung stoves, including Parsons's model.

The lawsuit alleges that a defect in the oven temperature sensor causes failures in the range's control boards.

"When the control boards fail, the [range's] oven and burner temperatures deviate from the user-selected temperature settings," the document said.


SAMSUNG
DE92-02439*
DE41-00408C
DK00117C 12261


GOLDEN
GK-A-1A-12D
2HP 250VAC
30A 240VAC
40A 277VAC
12VDC C2205-03A2
GOLDEN
GK-A-1A-12D
2HP 250VAC
30A 240VAC
40A 277VAC
12VDC C2205-03A2
GOLDEN
GK-A-1A-12D
2HP 250VAC
30A 240VAC
40A 277VAC
12VDC C2205-03A2
LOAD1
LOAD5
LOAD4
LOAD2
LOAD3
DOOR LOCK
SAMSUNG
DE41-00408C



TMP91FW60

- TLCS 900/L1 CPU
- 8K RAM / 128 K flash
- Bootloader in ROM
- External xtal (no PLL)
- Obsolete…

(1) High-speed 16-bit CPU (900/L1 CPU)

- Instruction mnemonics are upward-compatible with TLCS-90/900
- General-purpose registers and register banks
- 16 Mbytes of linear address space
- 16-bit multiplication and division instructions; bit transfer and arithmetic instructions

1.3 Block Diagram

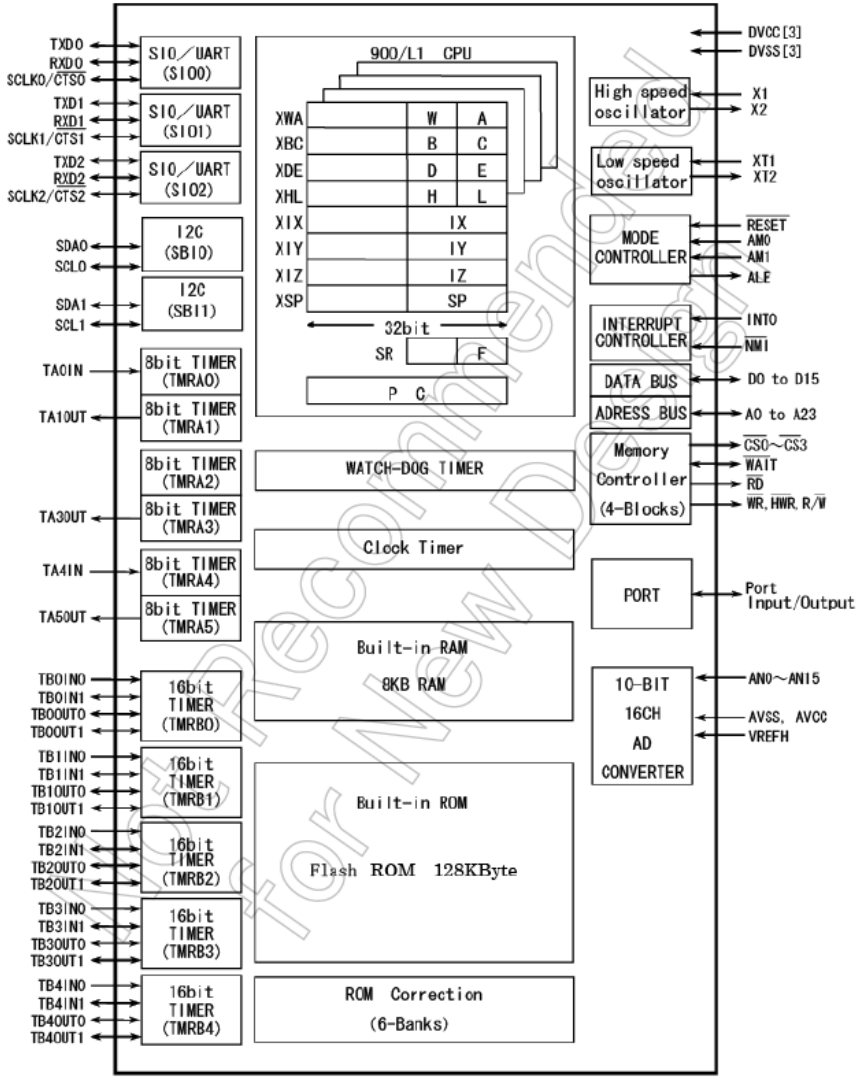


Figure 1-3 Block Diagram

Bootloader

14.4.6 Data Transfer Formats

Table 14-7 to Table 14-12 show the operation command data and the data transfer format for each operation mode.

Table 14-7 Operation Command Data

Operation Command Data	Operation Mode
10H	RAM Transfer
20H	Flash Memory SUM
30H	Product Information Read
40H	Flash Memory Chip Erase
60H	Flash Memory Protect Set

Table 14-8 Transfer Format of Single Boot Program [RAM Transfer]

	Transfer Byte Number	Transfer Data from Controller to Device	Baud Rate	Transfer Data from Device to Controller
BOOT ROM	1st byte	Baud rate setting UART 86H	Desired baud rate#1	-
	2nd byte	-		ACK response to baud rate setting Normal (baud rate OK) >UART 86H (If the desired baud rate cannot be set, operation is terminated.)
	3rd byte	Operation command data (10H)		-
	4th byte	-		ACK response to operation command#2 Normal 10H Error x1H Protection applied#3 x6H Communications error x8H
	5th byte to 16th byte	PASSWORD data (12 bytes) (02FEF4H to 02FEFFH)		-
	17th byte	CHECKSUM value for 5th to 16th bytes		-
	18th byte	-		ACK response to CHECKSUM value#2 Normal 10H Error 11H Communications error 18H
	19th byte	RAM storage start address 31 to 24 #4		-
	20th byte	RAM storage start address 23 to 16#4		-

Table 14-12 Transfer Format of Single Boot Program [Flash Memory Protect Set]

	Transfer Byte Number	Transfer Data from Controller to Device	Baud Rate	Transfer Data from Device to Controller
BOOT ROM	1st byte	Baud rate setting UART 86H	Desired baud rate[#1]	-
	2nd byte	-		ACK response to baud rate setting Normal (baud rate OK) >UART 86H (If the desired baud rate cannot be set, operation is terminated.)
	3rd byte	Operation command data (60H)		-
	4th byte	-		ACK response to operation command[#2] Normal 60H Error x1H Communications x8H
	5th byte to 16th byte	Password data (12 bytes) (02FEF4H to 02FEFFH)		-
	17th byte	CHECKSUM value for 5th to 16th bytes		-
	18th byte	-		ACK response to checksum value[#2] Normal 60H Error 61H Communications 68H
				ACK response to Protect Set command

Important Take-Aways (for next part)

1. Bootloader has no read-back command, only RAM program. Need to build/find 2nd stage bootloader.
2. Bootloader has TWO security protections that can be enabled:
    1. “Protection Flag” → Disables second-stage capability (leaves “erase” enabled). Disables RAM functionality, so no chance to read-back flash.
    2. 12-byte Password that can be set in Flash. Password locks RAM functionality but does not disable it.
3. Bootloader has a function that only needs password (even if protection is set).

Programmer / Disassembler / Simulator?

Toshiba BMSKTOPAS91FY42(A) kit for flash microcontroller TOPAS 900/L1

Last item available

Condition: **New – Open box**

"New item in Good Condition"

Quantity: 1 Last One / 1 sold

Price: **US $280.00**

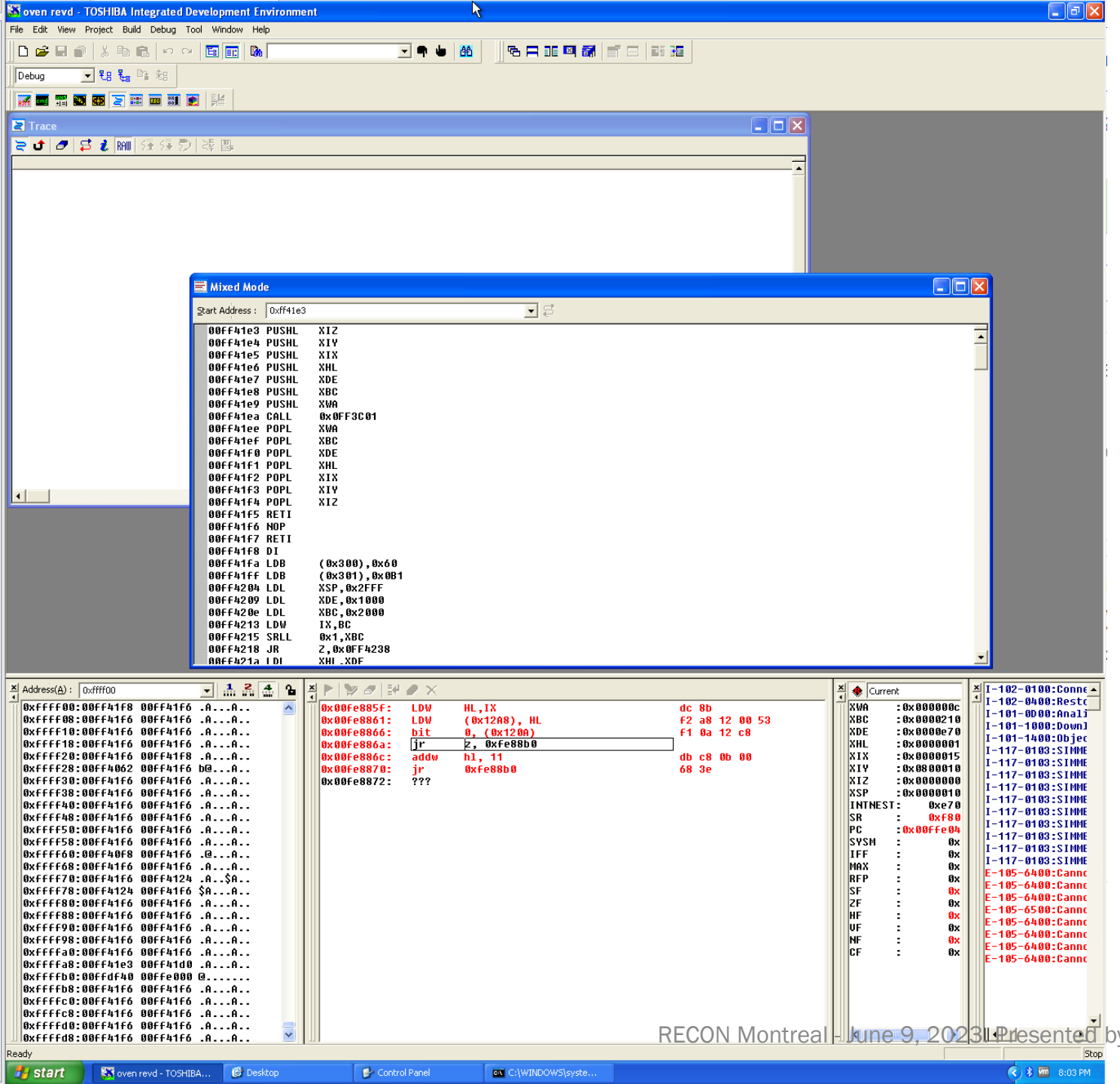
Buy another

Add to cart

$ Have one to sell? Sell now

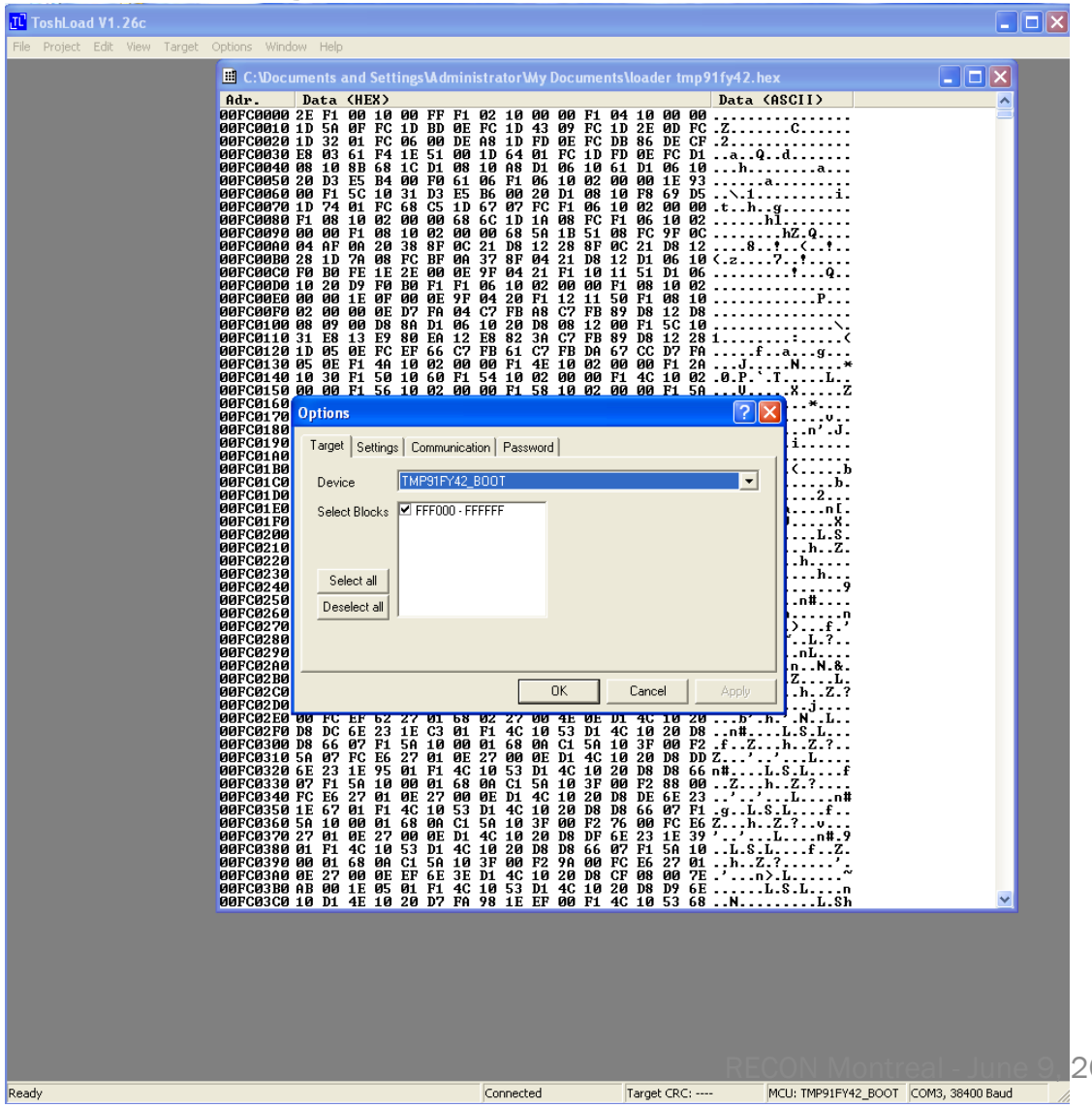
Best Offer:

Windows XP?



https://github.com/colinoflynn/Toshiba-TLCS-900-L-Resources

Can you Read Back Bootloader?



Segger “ToshLoad” can read-back bootloader (ROM) section!

Watch for how ROM remaps when in bootloader (single boot) mode.

(I made a Python version of this program so you *don’t* need Windows XP)

```
FUNCTION START: Receive & Verify Password
00fff2a2 CALR    0x0FFF5EF <-- RX
...
00fff2ce JR      NZ,0x0FFF2D5
00fff2d0 DJNZB   C,0x0FFF2C9
00fff2d3 JR      0x0FFF2D7
00fff2d5 LDB     L,0x1 <-- L is flag, if set to 1 comparison failed
00fff2d7 LDW     BC,0x0C <-- 12 bytes to compare
00fff2da LDL     XIX,(0x0FFF00C) <-- Points to 0004FEF4 (PW)
00fff2df LDB     RH1,0x0
00fff2e2 LDB     W,(XIX+) <--Load byte into W, inc XIX ptr (loop)
00fff2e5 CALR    0x0FFF635 <--- RX assumed
00fff2e8 CPB     W,A <--Compare W & A
00fff2ea JR      Z,0x0FFF2EE <-- Compare OK, skip fail set
00fff2ec LDB     L,0x1 <--Set 'fail' flag
00fff2ee DJNZW   BC,0x0FFF2E2 <--Jump to next byte (12 times)
00fff2f1 CALR    0x0FFF67B <-- checksum
00fff2f4 RET
```

```
FUNCTION START: RAM WRITE FUNCTION
00fff2f5 CALR    0x0FFF75F  <-- Load protection status
00fff2f8 CPB     A,0x0FF <-- Compare protection status
00fff2fb JR      NZ,0x0FFF290 <-- Send error if protection enabled
00fff2fd CALR    0x0FFF2A2 <-- PW Check
00fff300 CPB     RE1,0x0
00fff303 JR      NZ,0x0FFF28A
00fff305 CPB     RL1,0x0
00fff308 JR      NZ,0x0FFF29C <-- Error
00fff30a CPB     L,0x0
00fff30c JR      NZ,0x0FFF29C <-- Error
00fff30e CALR    0x0FFF5EF <- TX
00fff311 LDB     RH1,0x0
00fff314 CALR    0x0FFF635 <--
00fff317 LDB     QIXH,A
```

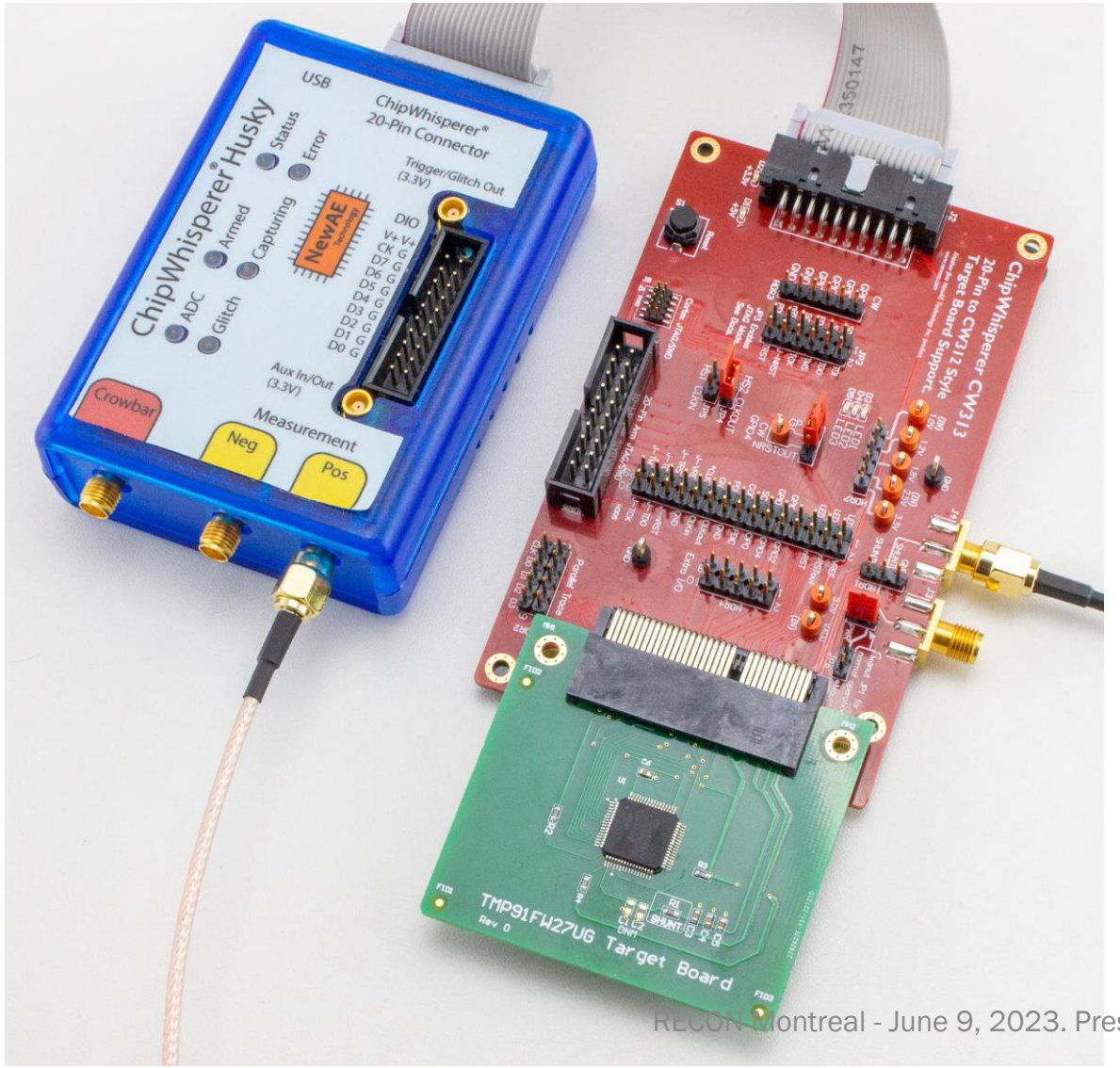
Important Take-Aways (for next stage)

1. Password check has slight code-flow dependency.

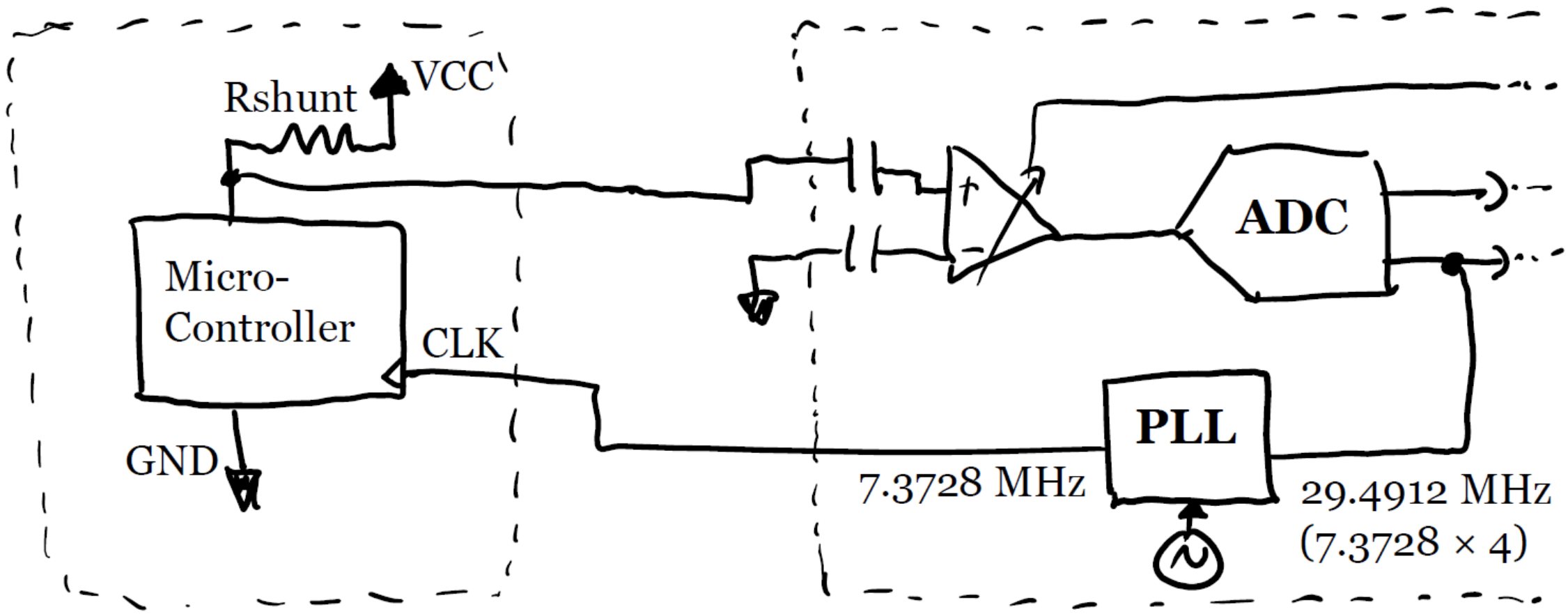
2. Fuse byte check has obvious fault injection location.


Select difficulty:
Easy
Medium
Hard

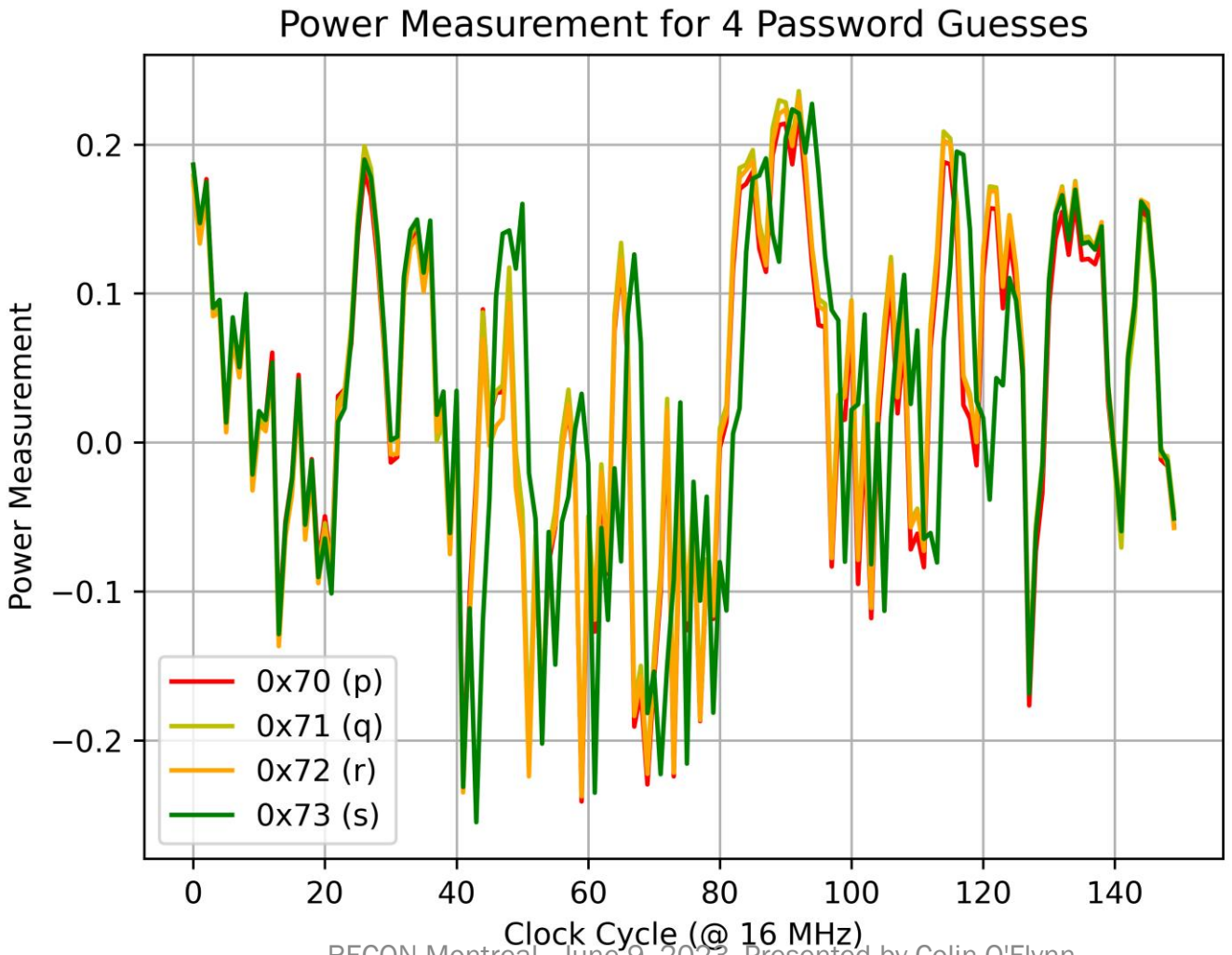
ChipWhisperer-Husky Intro

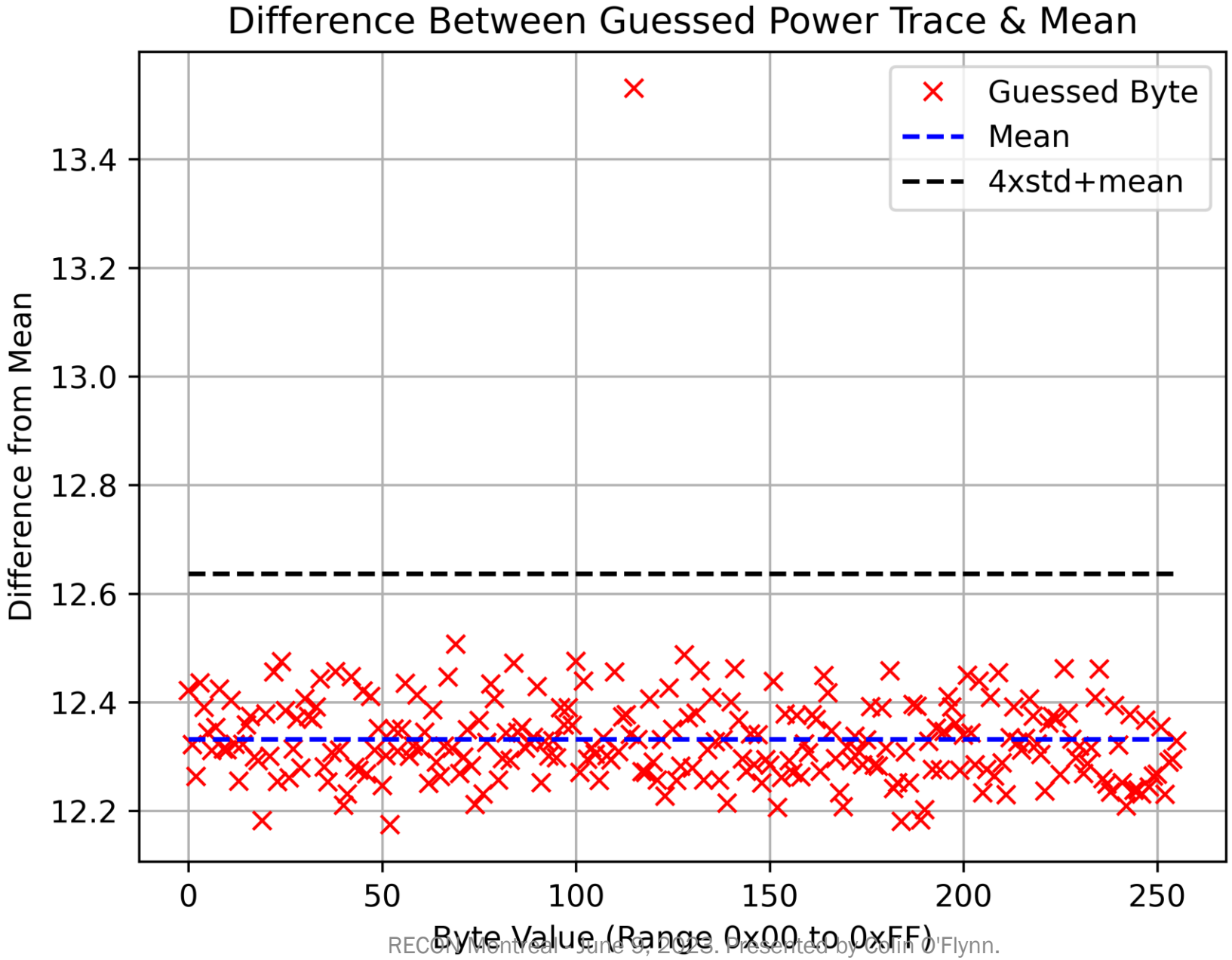


Power Analysis?



Easy-Mode Level 1: Password Power Analysis



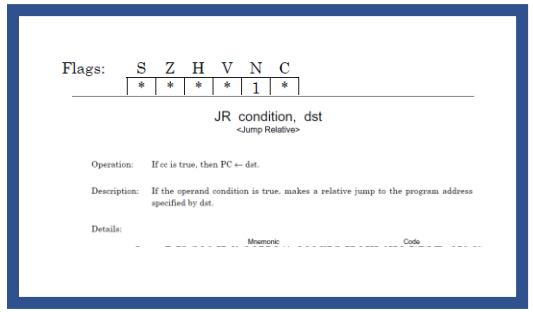

Difference Between Guessed Power Trace & Mean
Guessed Byte
Mean
4xstd+mean
Difference from Mean
13.4
13.2
13.0
12.8
12.6
12.4
12.2
0
50
100
150
200
250
Byte Value (Range 0x00 to 0xFF)

Fault Injection?

```
FUNCTION START: RAM WRITE FUNCTION
00fff2f5 CALR     0x0FFF75F  <-- Load protection status
00fff2f8 CPB      A,0x0FF <-- Compare protection status
00fff2fb JR       NZ,0x0FFF290 <-- Send error if protection enabled
00fff2fd CALR     0x0FFF2A2 <-- PW Check
```

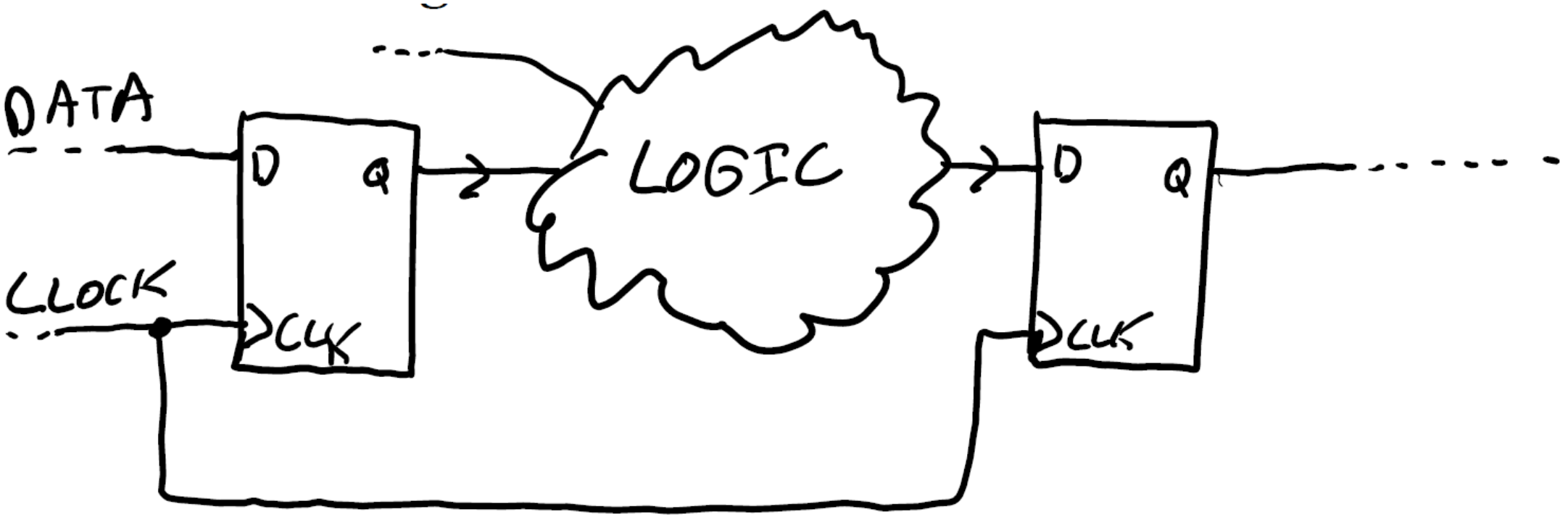
Fetch

Decode

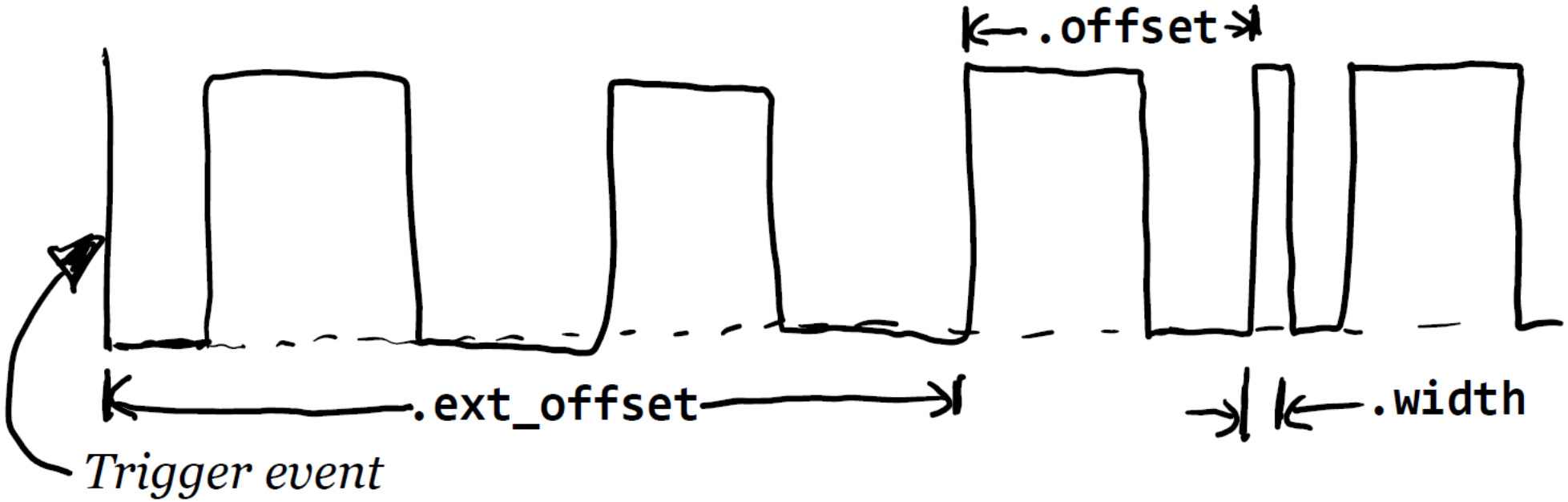


Execute

Fault Injection?



Clock Fault Injection



Easy-Mode Level 2: Fault Injection Tuning

Table 14-9 Transfer Format of Single Boot Program [Flash Memory SUM]

	Transfer Byte Number	Transfer Data from Controller to Device	Baud Rate	Transfer Data from Device to Controller
BOOT ROM	1st byte	Baud rate setting UART 86H	Desired baud rate#1	-
	2nd byte	-		ACK response to baud rate setting Normal (baud rate OK) >UART 86H (If the desired baud rate cannot be set, operation is terminated.)
	3rd byte	Operation command data (20H)		-
	4th byte	-		ACK response to CHECKSUM value#2 Normal 20H Error x1H Communications error x8H
	5th byte	-		SUM (upper)
	6th byte	-		SUM (lower)
	7th byte	-		CHECKSUM value for 5th and 6th bytes
	8th byte	(Wait for the next operation command data)		-

#1 For the desired baud rate setting, see Table 14-6.
#2 After sending an error response, the device waits for operation command data (3rd byte).

Flash memory SUM = MANY opportunities to glitch result (entire SUM operation)

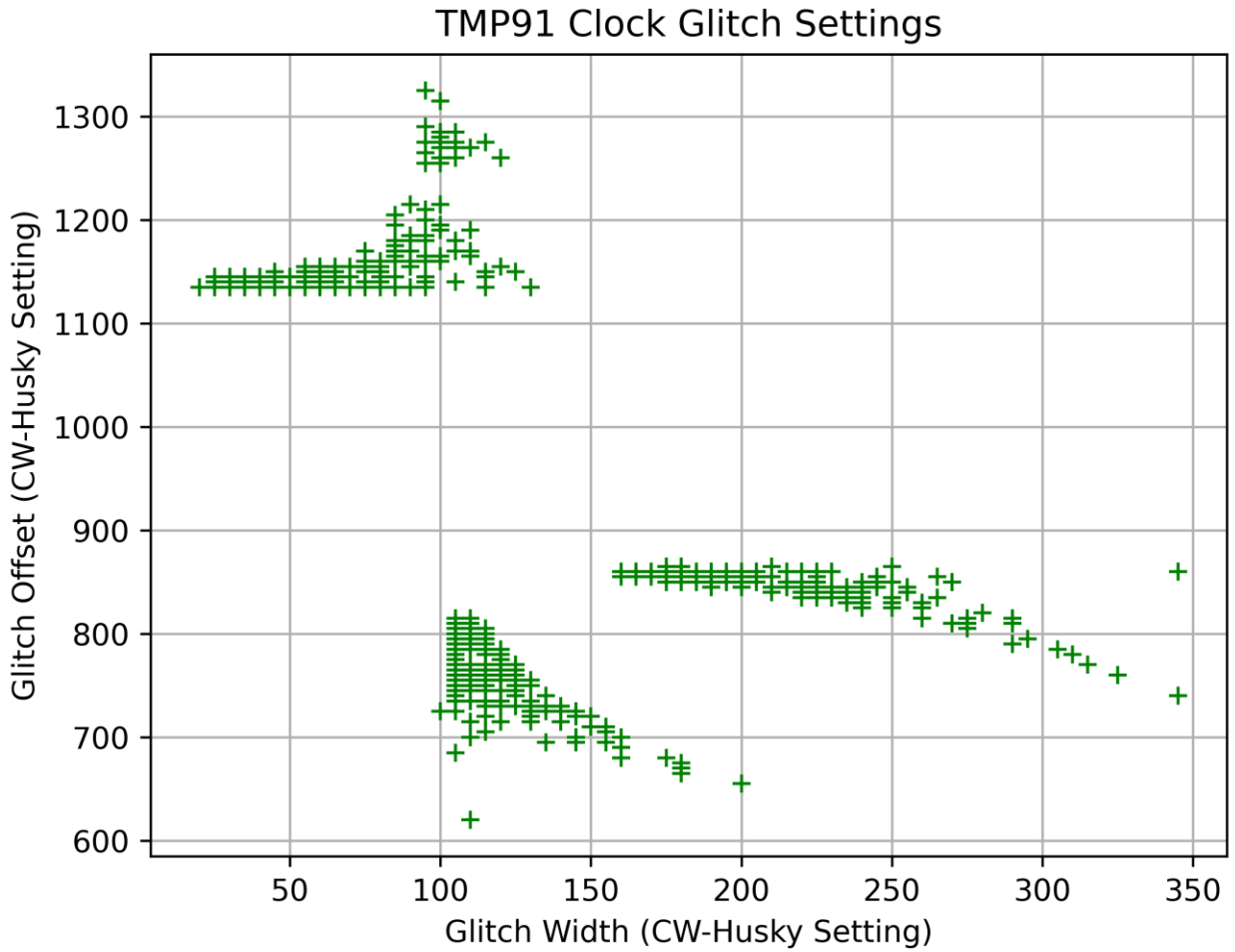
Fault Injection Setup / Demo

```
In [52]:
reset_target()
response, responsehex = tx_rx(b"\x86", 1, 1)
if responsehex[0] != 0x86:
    raise IOError("Sync Error")
response, responsehex = tx_rx(b"\x20", 4)
responsehex
```

```
Out[52]: [32, 250, 165, 97]
```

```
broken = False
for glitch_setting in gc.glitch_values():
    reset_target()
    scope.glitch.offset = glitch_setting[1]
    scope.glitch.width = glitch_setting[0]

    reset_target()
    target.ser.flush()
    response, responsehex = tx_rx(b"\x86", 1, 1)
    if responsehex[0] != 0x86:
        raise IOError("Sync Error")

    scope.arm()

    #Do glitch loop
    target.ser.write(b"\x20")

    ret = scope.capture()

    loff = scope.glitch.offset
    lwid = scope.glitch.width

    if ret:
        print('Timeout - no trigger')
        gc.add("reset")

        #Device is slow to boot?
        reset_target()

    else:

        response = target.ser.read(4)
        response = [ord(i) for i in response]

        if len(response) == 0:
            gc.add("reset")
        else:
            if  response != [32, 250, 165, 97]:
                broken = True
                gc.add("success")
                print(response)
                print(loff)
                print(lwid)
                print("💥", end="")
            else:
                gc.add("normal")

print("Done glitching")
```

Fault Injection Results (SUM Corruption)



Easy-Mode Level 3: Fault Injection Attack

```
scope.glitch.width = 2000 #100 #1000

for glitch_settings in gc.glitch_values():
    scope.glitch.ext_offset = glitch_settings[0]
    for i in range(sample_size):
        reset_target()

        target.ser.flush()
        response, responsehex = tx_rx(b"\x86", 1, 1)
        if responsehex[0] != 0x86:
            raise IOError("Sync Error")

        scope.arm()

        #Do glitch loop
        target.ser.write(b"\x10")

        ret = scope.capture()

        if ret:
            print('Timeout - no trigger')
            gc.add("reset")

            #Device is slow to boot?
            reset_target()

        else:
            response = target.ser.read(1)
            response = [ord(i) for i in response]

            if len(response) == 0:
                gc.add("reset")
            else:

                if response[0] != 0x16:
                    #broken = True
                    gc.add("success")
                    print(response)
                    print(hex(response[0]))
                    print(scope.glitch.ext_offset)
                    print("🦀", end="")

                    if response[0] == 0x10:
                        broken=True
                        break

                    #break
                else:
                    gc.add("normal")
    if broken:
        break
```

```
[16]
0x10
8015
🦀
```

```
In [59]: known_pw = [0xDE, 0xAD, 0xBE, 0xEF, 0xCA, 0xFE, 0xFA, 0xCE, 0x11, 0x22, 0x33, 0x44]

bl = tl.LowLevelBootloader(target.ser, reset_target, password=known_pw, reset_and_connect=False)
bl.cmd_ram_transfer(rc.B_F16_RAM1000_ROM10000_TLCS900L1["data"], rc.B_F16_RAM1000_ROM10000_TLCS900L1["start_address"], skipcm
rl = tl.RamCodeProtocol(target.ser)
```

```
In [60]: #Print the password (should match the known one)
time.sleep(0.1)
data = rl.cmd_read(0x02FEF4, 12)
':'.join(hex(ord(char)) for char in data)

Out[60]: '0xde:0xad:0xbe:0xef:0xca:0xfe:0xfa:0xce:0x11:0x22:0x33:0x44'
```

```
In [12]: #Read the full flash itself
#TMP91FW27UG in Single Boot Mode - flash is from 0x10000 to 0x30000 (starts @ 0x10000, length = 0x20000)
flash = rl.cmd_read(0x10000, 0x20000)
```

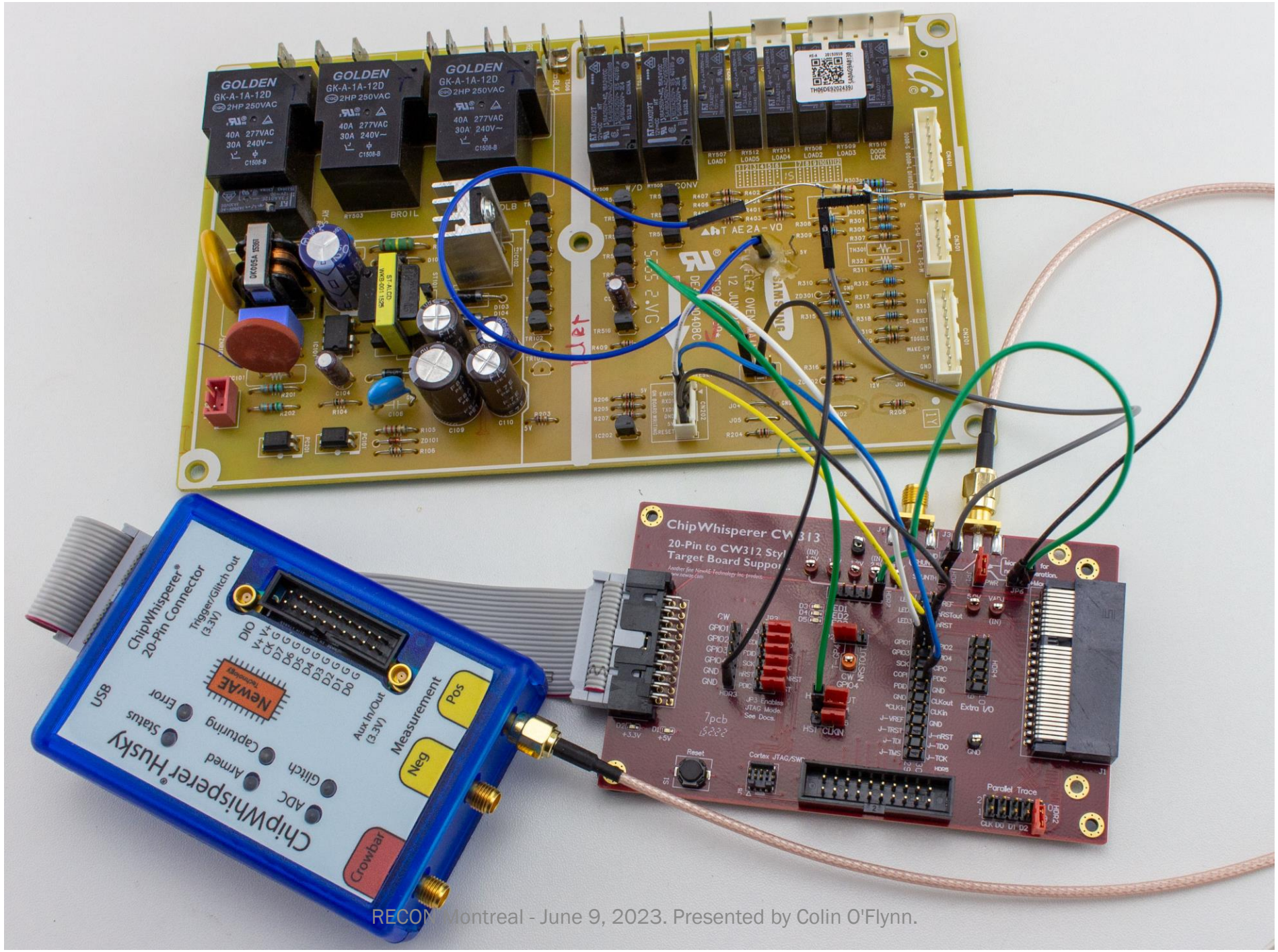
```
In [13]: len(flash)

Out[13]: 131072
```

```
In [ ]: known_pw = [0xDE, 0xAD, 0xBE, 0xEF, 0xCA, 0xFE, 0xFA, 0xCE, 0x11, 0x22, 0x33, 0x44]

bl = tl.LowLevelBootloader(target.ser, reset_target, password=known_pw, reset_and_connect=False)
bl.cmd_ram_transfer(rc.B_F16_RAM1000_ROM10000_TLCS900L1["data"], rc.B_F16_RAM1000_ROM10000_TLCS900L1["start_address"], skipcm
rl = tl.RamCodeProtocol(target.ser)
```

Skills & Resources

- Python class for communicating & programming TMP91 (including 2nd stage bootloader communications).
- Timing on power analysis.
- Rough timing / details on fault injection.


Select difficulty:
Easy
Medium
Hard


GOLDEN
GK-A-1A-12D
2HP 250VAC
40A 277VAC
30A 240V~
ChipWhisperer CW313
20-Pin to CW312 Style
Target Board Support
ChipWhisperer® Husky
ChipWhisperer® 20-Pin Connector
Trigger/Glitch Out (3.3V)
DIO
Aux In/Out (3.3V)
Measurement
Pos
Neg
Crowbar
USB
Status
Error
Armed
Capturing
Glitch
ADC
NewAE Technology
Reset
Cortex JTAG/SWD
Extra I/O
Parallel Trace

Medium-Mode Level 1: Power Analysis

In [18]:

```
%matplotlib notebook
import matplotlib.pylab as plt
import numpy as np
from tqdm.notebook import trange, tqdm

trace1 = None
go = True

i = 0x00

diffs = []


while go:
    reset_target()
    target.ser.flush()
    response, responsehex = tx_rx(b"\x86", 1, 1)
    if responsehex[0] != 0x86:
        raise IOError("Sync Error")

    response, responsehex = tx_rx(b"\x60", 1, 1)

    if responsehex[0] != 0x60:
        raise IOError("Unexpected ACK = 0x%x"%responsehex[0])

    write_pw("sam")

    scope.arm()
    target.ser.write(str(chr(i)))
    scope.capture()
    trace = scope.get_last_trace()

    if trace1 is None:
        trace1 = trace[:]
        start = np.where(trace1 < -0.3)[0][0] - 200
        end = start+400
        print("Using template at %d-%d"%(start,end))

    try:
        trace = resync_sad(trace, trace1, (start,end))[start-400:end-400]
    except ValueError:
        continue

    diff = np.sum(abs(trace - trace1[start:end]))
    diffs.append(diff)
    print("%x %f"%(i, diff))

    i += 1
    if i > 0x02:
        break

    plt.plot(trace)
```

Sending known part of password, then do the attack on next unknown byte

s..a..m..s..u..n..g..o..v..e..n..0

Medium-Mode Level 2: Fault Injection

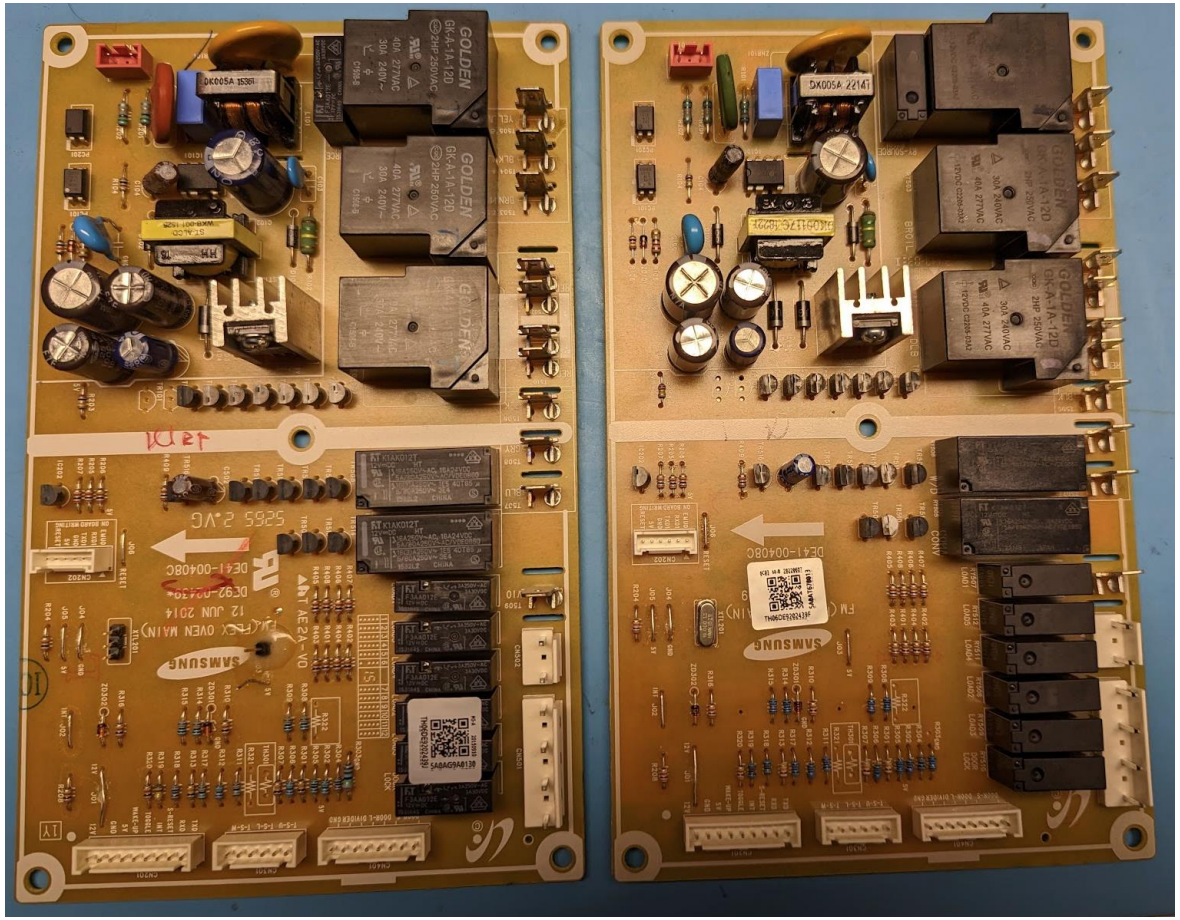
```
0x87
11710
[133]
0x85
11715
[17]
0x11
11750
[16]
0x10
11755
```

```
In [59]:
#known_pw = [0xDE, 0xAD, 0xBE, 0xEF, 0xCA, 0xFE, 0xFA, 0xCE, 0x11, 0x22, 0x33, 0x44]
known_pw = [ord(c) for c in "samsungoven0"]

bl = tl.LowLevelBootloader(target.ser, reset_target, password=known_pw, reset_and_connect=False)
bl.cmd_ram_transfer(rc.B_F16_RAM1000_ROM10000_TLCS900L1["data"], rc.B_F16_RAM1000_ROM10000_TLCS900L1["start_address"], skipcm
rl = tl.RamCodeProtocol(target.ser)
```

```
In [11]: resp = rcp.cmd_read(0x10000, 0x100)

In [12]: resp
Out[12]: 'ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
ÿÿÿÿÿÿÿÿÿÿ'
```

```
In [7]: bl = tl
        #bl.cmd
        #bl.cmd
        bl.cmd_

        Read: n
        Write:
```

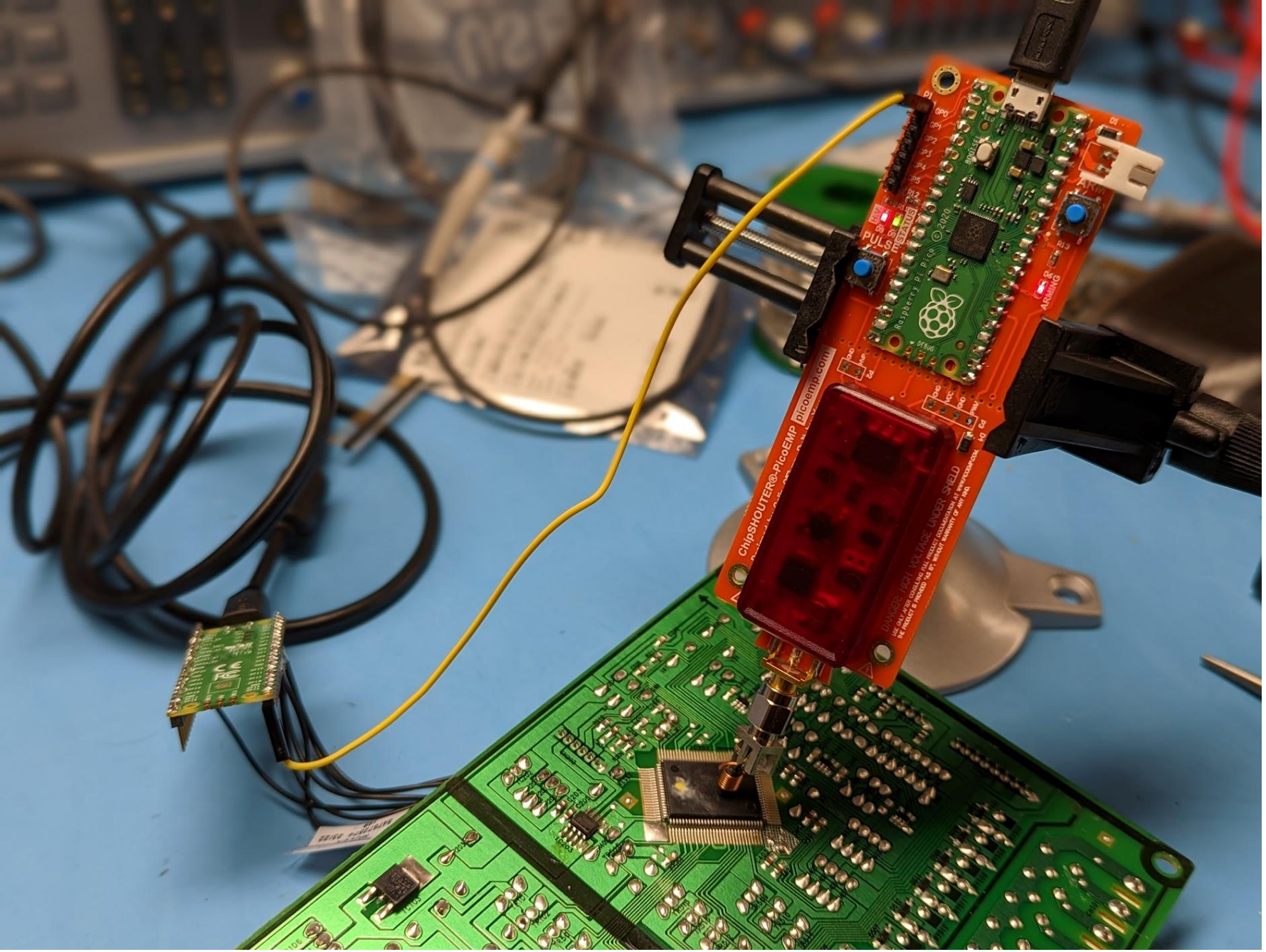
$$ → Samsung Parts Department



Did they have problems with returns?

Or eeded on the
ards!!

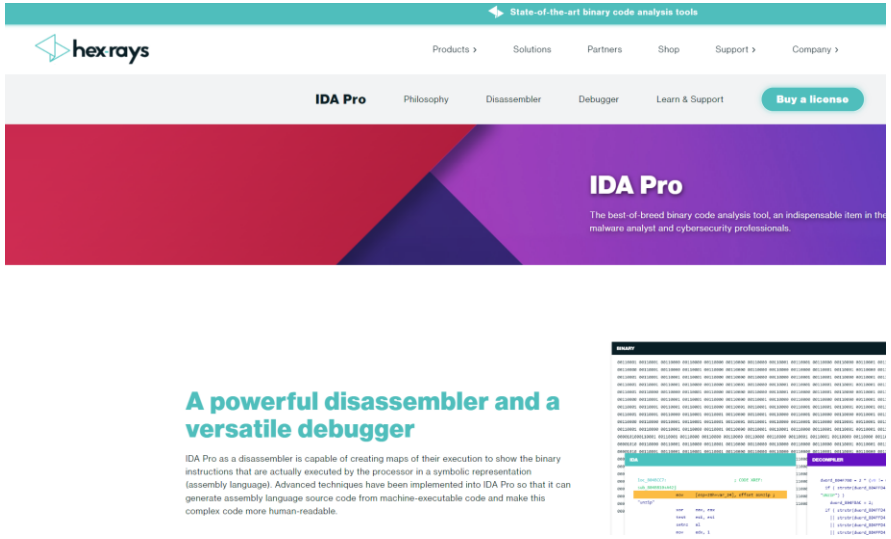

0000h: C2 DA 13 00 3F 01 B0 F6 C2 F8 13 00 3F 00 B0 FE ÂÚ..?.°öÂø..?.°þ
0010h: C2 C6 12 00 3F 00 6E 08 C2 64 11 00 3F 00 B0 FE ÂÆ..?.n.Âd..?.°þ
0020h: C2 4E 13 00 3F 00 B0 FE C2 1A 13 00 3F 00 B0 FE ÂN..?.°þÂ...?.°þ
0030h: C2 68 11 00 3F 00 B0 FE C2 71 11 00 3F 00 B0 FE Âh..?.°þÂq..?.°þ
0040h: C2 7A 11 00 3F 00 B0 FE D2 BC 12 00 3F 90 01 6B Âz..?.°þÒ¼..?..k
0050h: 09 D2 BE 12 00 3F 90 01 63 13 F2 A2 12 00 00 03 .Ò¾..?..c.ò¢....
0060h: F2 20 13 00 00 01 F2 F0 10 00 00 08 0E C2 5A 12 ò ....òð.....ÂZ.
0070h: 00 3F 01 6E 13 F2 A2 12 00 00 03 F2 20 13 00 00 .?.n.ò¢....ò ...
0080h: 02 F2 F0 10 00 00 08 0E C2 3A 14 00 3F 01 6E 0E .òð.....Â:..?.n.
0090h: F2 3A 14 00 00 00 F2 5C 12 00 00 03 68 26 F2 3A ò:....ò\....h&ò:
00A0h: 14 00 00 01 F2 5C 12 00 00 01 C2 58 12 00 3F 00 ....ò\....ÂX..?.
00B0h: 6E 06 F2 58 12 00 00 03 F2 6C 12 00 00 00 F2 6E n.òX....òl....òn
00C0h: 12 00 00 00 F2 F0 10 00 00 01 F2 58 14 00 00 00 ....òð....òX....
00D0h: 0E 8F 04 23 C2 6A 11 00 3F 0A 66 08 C2 6A 11 00 ...#Âj..?.f.Âj..
00E0h: 3F 0B 6E 16 C2 6B 11 00 3F 0F 6E 0E C2 5C 12 00 ?.n.Âk..?.n.Â\..
00F0h: 3F 05 66 06 F2 5C 12 00 00 03 CB 89 D8 12 D8 09 ?.f.ò\....Ë‰Ø.Ø.
0100h: 09 00 F2 6A 11 00 32 F3 07 E8 E0 00 00 CB 89 D8 ..òj..2ó.èà..Ë‰Ø
0110h: 12 D8 09 09 00 F2 6B 11 00 32 F3 07 E8 E0 00 00 .Ø...òk..2ó.èà..
0120h: CB 89 D8 12 D8 09 09 00 F2 68 11 00 32 F3 07 E8 Ë‰Ø.Ø...òh..2ó.è
0130h: E0 00 00 CB 89 D8 12 D8 09 09 00 F2 69 11 00 32 à..Ë‰Ø.Ø...òi..2
0140h: F3 07 E8 E0 00 00 CB 89 D8 12 D8 09 09 00 F2 6C ó.èà..Ë‰Ø.Ø...òl
0150h: 11 00 32 F3 07 E8 E0 00 00 CB 89 D8 12 D8 09 09 ..2ó.èà..Ë‰Ø.Ø..
0160h: 00 F2 6D 11 00 32 F3 07 E8 E0 00 00 CB 89 D8 12 .òm..2ó.èà..Ë‰Ø.
0170h: D8 09 09 00 F2 6E 11 00 32 F3 07 E8 E0 00 00 CB Ø...òn..2ó.èà..Ë
0180h: 89 D8 12 D8 09 09 00 F2 6F 11 00 32 F3 07 E8 E0 ‰Ø.Ø...òo..2ó.èà
0190h: 02 00 00 CB 89 D8 12 F2 98 12 00 32 F3 07 E8 E0 ...Ë‰Ø.ò˜..2ó.èà
01A0h: 00 00 CB 89 D8 12 F2 34 14 00 32 F3 07 E8 E0 00 ..Ë‰Ø.ò4..2ó.èà.
01B0h: 00 CB 89 D8 12 D8 80 F2 FE 13 00 32 F3 07 E8 E0 .Ë‰Ø.Ø€òþ..2ó.èà
01C0h: 02 00 00 C2 64 11 00 3F 00 6E 12 F2 A6 12 00 00 ...Âd..?.n.ò¦...
01D0h: 00 F2 A7 12 00 00 00 F2 A8 12 00 00 00 CB D8 66 .ò§....ò¨....ËØf
01E0h: 04 CB D9 6E 14 F2 D6 12 00 00 00 F2 DA 12 00 00 .ËÙn.òÖ....òÚ...
01F0h: 00 F2 68 12 00 00 00 68 0C F2 D8 12 00 00 00 F2 .òh....h.òØ....ò
0200h: DC 12 00 00 00 F2 E8 13 00 00 00 0E F2 66 11 00 Ü....òè.....òf..
0210h: 00 00 F2 64 11 00 00 00 F2 1E 12 00 00 00 F2 96 ..òd....ò.....ò–
0220h: 12 00 00 00 F2 9C 12 00 00 00 00 F2 5E 12 00 00 ....òœ.....ò^....
0230h: F2 66 12 00 02 00 00 F2 64 12 00 02 00 00 F2 8C òf.....òd.....òŒ
0240h: 12 00 00 00 F2 18 12 00 02 00 00 F2 C0 12 00 00 ....ò.......òÀ...
0250h: 00 F2 54 11 00 02 00 00 F2 38 11 00 02 00 00 F2 .òT.....ò8.....ò
0260h: 3C 14 00 00 00 0B 00 00 1D EF A3 FE EF 62 F2 9E <........ï£þïbòž
0270h: 12 00 00 00 F2 A0 12 00 00 00 F2 A2 12 00 00 00 ....ò ....ò¢....
0280h: F2 20 13 00 00 00 C2 F8 13 00 3F 00 66 06 F2 FA ò ....Âø..?.f.òú

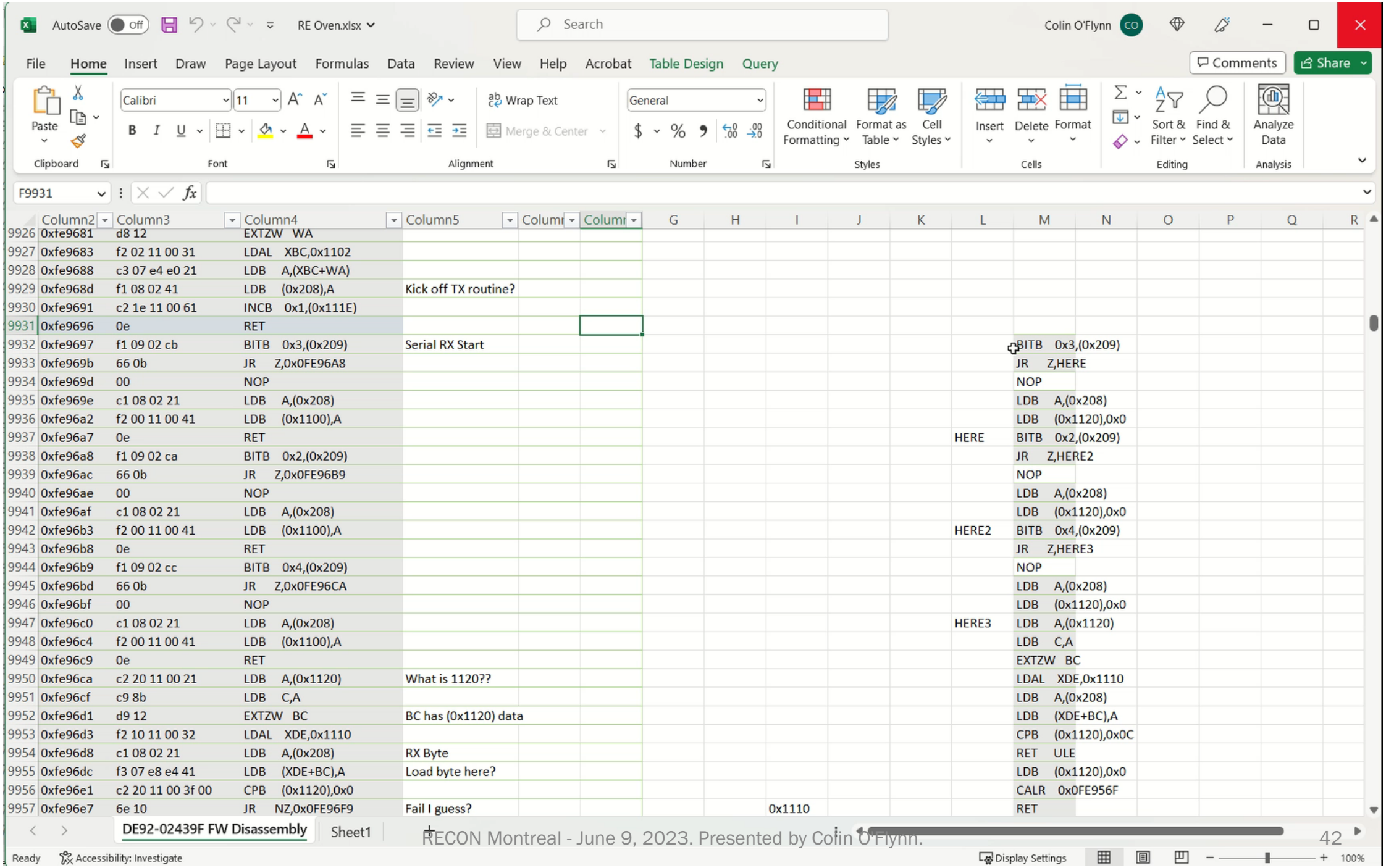


EMFI POC

- R-Pi Pico implements serial protocol.
- PicoEMP triggers an electromagnetic fault injection (EMFI).
- Tested on checksum request from bootloader → successfully corrupted checksums.
- Code available in repo (linked later).

Reverse Engineering Tools





	Column2	Column3	Column4	Column5	G	H	I	J	K	L	M	N
9926	0xfe9681	d8 12	EXTZW WA									
9927	0xfe9683	f2 02 11 00 31	LDAL XBC,0x1102									
9928	0xfe9688	c3 07 e4 e0 21	LDB A,(XBC+WA)									
9929	0xfe968d	f1 08 02 41	LDB (0x208),A	Kick off TX routine?								
9930	0xfe9691	c2 1e 11 00 61	INCB 0x1,(0x111E)									
9931	0xfe9696	0e	RET									
9932	0xfe9697	f1 09 02 cb	BITB 0x3,(0x209)	Serial RX Start							BITB	0x3,(0x209)
9933	0xfe969b	66 0b	JR Z,0x0FE96A8								JR	Z,HERE
9934	0xfe969d	00	NOP								NOP	
9935	0xfe969e	c1 08 02 21	LDB A,(0x208)								LDB	A,(0x208)
9936	0xfe96a2	f2 00 11 00 41	LDB (0x1100),A								LDB	(0x1120),0x0
9937	0xfe96a7	0e	RET							HERE	BITB	0x2,(0x209)
9938	0xfe96a8	f1 09 02 ca	BITB 0x2,(0x209)								JR	Z,HERE2
9939	0xfe96ac	66 0b	JR Z,0x0FE96B9								NOP	
9940	0xfe96ae	00	NOP								LDB	A,(0x208)
9941	0xfe96af	c1 08 02 21	LDB A,(0x208)								LDB	(0x1120),0x0
9942	0xfe96b3	f2 00 11 00 41	LDB (0x1100),A							HERE2	BITB	0x4,(0x209)
9943	0xfe96b8	0e	RET								JR	Z,HERE3
9944	0xfe96b9	f1 09 02 cc	BITB 0x4,(0x209)								NOP	
9945	0xfe96bd	66 0b	JR Z,0x0FE96CA								LDB	A,(0x208)
9946	0xfe96bf	00	NOP								LDB	(0x1120),0x0
9947	0xfe96c0	c1 08 02 21	LDB A,(0x208)							HERE3	LDB	A,(0x1120)
9948	0xfe96c4	f2 00 11 00 41	LDB (0x1100),A								LDB	C,A
9949	0xfe96c9	0e	RET								EXTZW	BC
9950	0xfe96ca	c2 20 11 00 21	LDB A,(0x1120)	What is 1120??							LDAL	XDE,0x1110
9951	0xfe96cf	c9 8b	LDB C,A								LDB	A,(0x208)
9952	0xfe96d1	d9 12	EXTZW BC	BC has (0x1120) data							LDB	(XDE+BC),A
9953	0xfe96d3	f2 10 11 00 32	LDAL XDE,0x1110								CPB	(0x1120),0x0C
9954	0xfe96d8	c1 08 02 21	LDB A,(0x208)	RX Byte							RET	ULE
9955	0xfe96dc	f3 07 e8 e4 41	LDB (XDE+BC),A	Load byte here?							LDB	(0x1120),0x0
9956	0xfe96e1	c2 20 11 00 3f 00	CPB (0x1120),0x0								CALR	0x0FE956F
9957	0xfe96e7	6e 10	JR NZ,0x0FE96F9	Fail I guess?			0x1110				RET	

Serial Monitor Built-In!?

- Not documented anywhere I could find (service docs).
- Could be useful for repair technicians!
  - Seems to only show status of various flags however, doesn’t seem to take any input.
- We could patch it to make a simple memory-dump monitor.


Bake
Roast
Bake
Broil
Pizza
Casserole
Chicken
Cake
E-84
Upper
Warm
On/Off
2
3
4
5
7
8
9
0

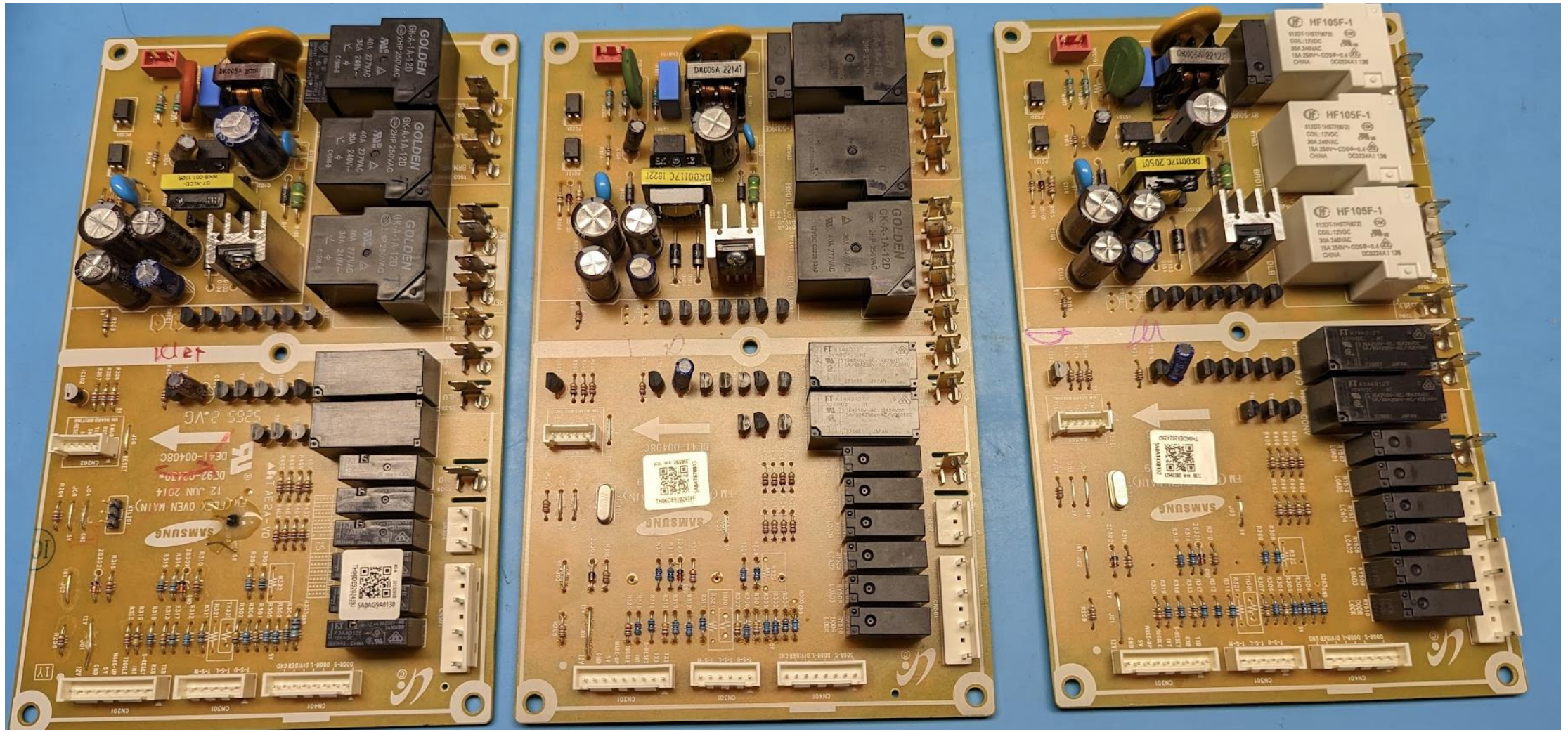
DE 02439F

OK, Just F

```
In [11]:  resp = rcp.cmd_read

In [12]:  resp

Out[12]: 'ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ ... ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
          ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ ... ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
          ÿÿÿÿÿÿÿÿÿ'
```

```
In [7]:  bl = tl
         #bl.cmd
         #bl.cmd
         bl.cmd_

         Read: r
         Write:
```

$$$ → Samsung Parts Department



Sidenote on Glitch Reliability

- Hitting too *early* seems more likely to trigger erase.
- my code tends to sweep early->late.
- Can increase reliability on specific targets (oven control board), I didn’t do that as thought it was just bad luck the 1st time…


Bake
Roast
Broil
Pizza
Casserole
Chicken Nuggets
Cake
Warm
Defrost
Steam Clean
Self Clean
On/Off
Level
Warming Drawer
Custom Cook
Cooking Time
Timer On/Off
Delay Start
Clock
E-84
12:00
Upper
Lower
1 2 3 4 5
6 7 8 9 0
OFF
LO
HI
MED

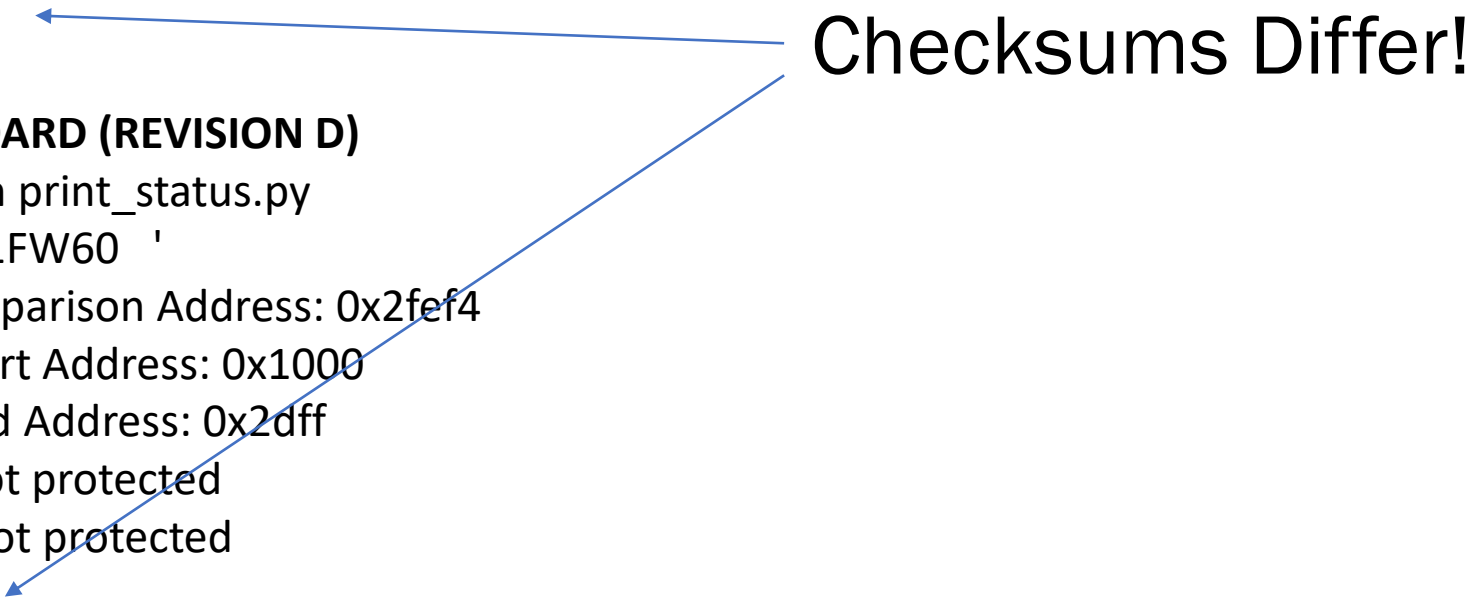
Have there been Firmware Fixes?

MY OVEN (REVISION D FIRMWARE)

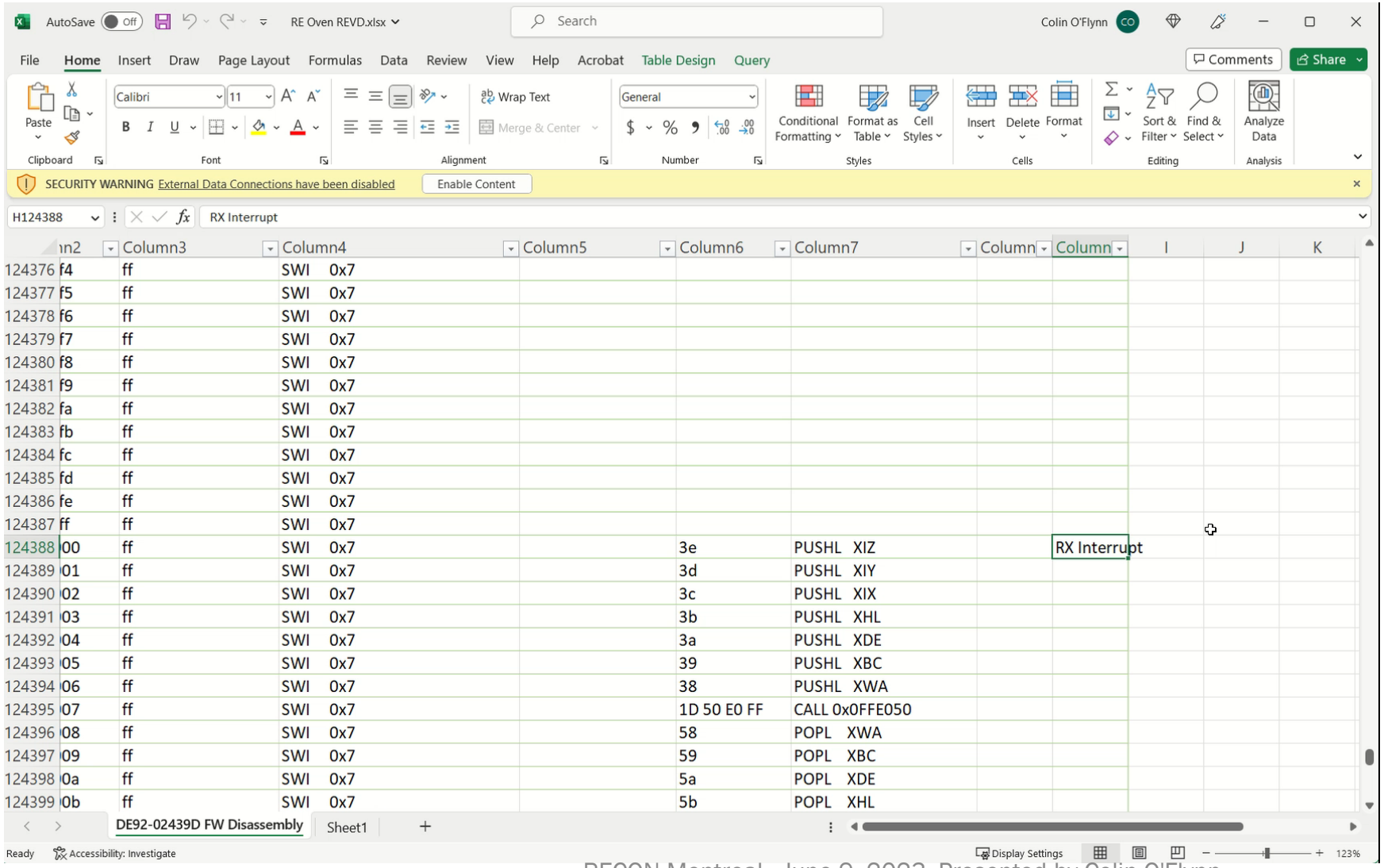
```
$ python print_status.py
b'TMP91FW60   '
PW Comparison Address: 0x2fef4
RAM Start Address: 0x1000
RAM End Address: 0x2dff
Read: protected
Write: protected
29171
```

NEW BOARD (REVISION D)

```
$ python print_status.py
b'TMP91FW60   '
PW Comparison Address: 0x2fef4
RAM Start Address: 0x1000
RAM End Address: 0x2dff
Read: not protected
Write: not protected
29238
```



..Add the Serial Monitor

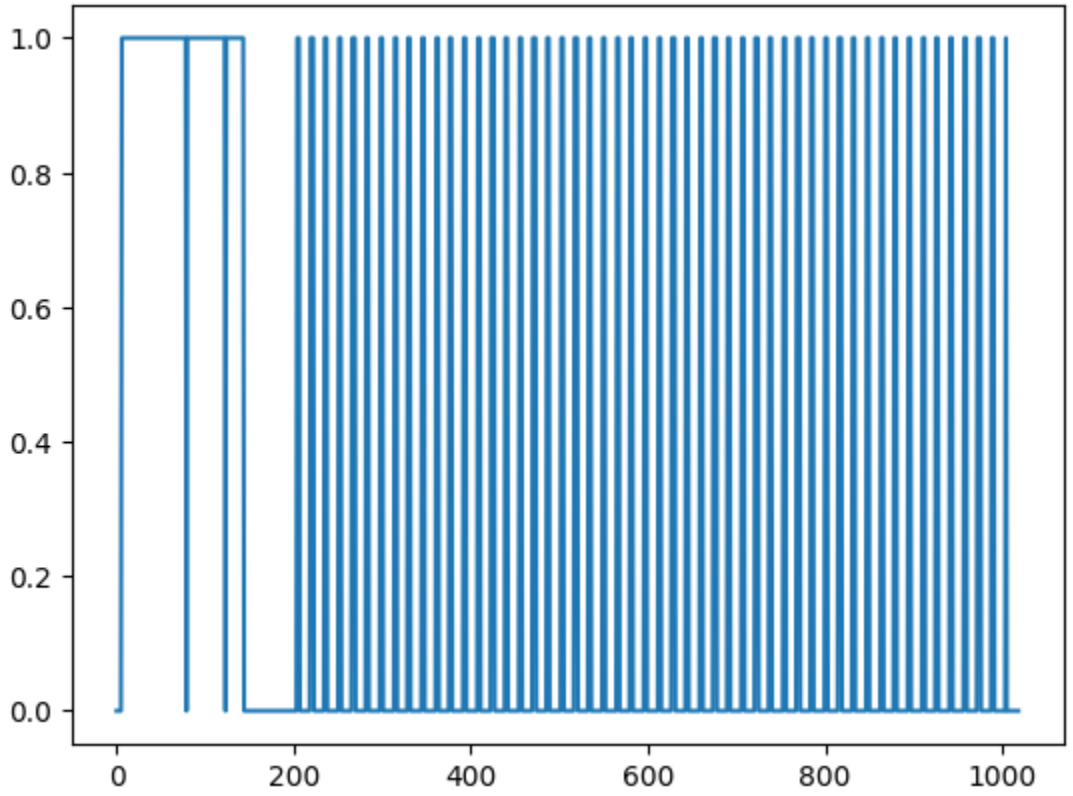


Slight risk of overwriting something else important....

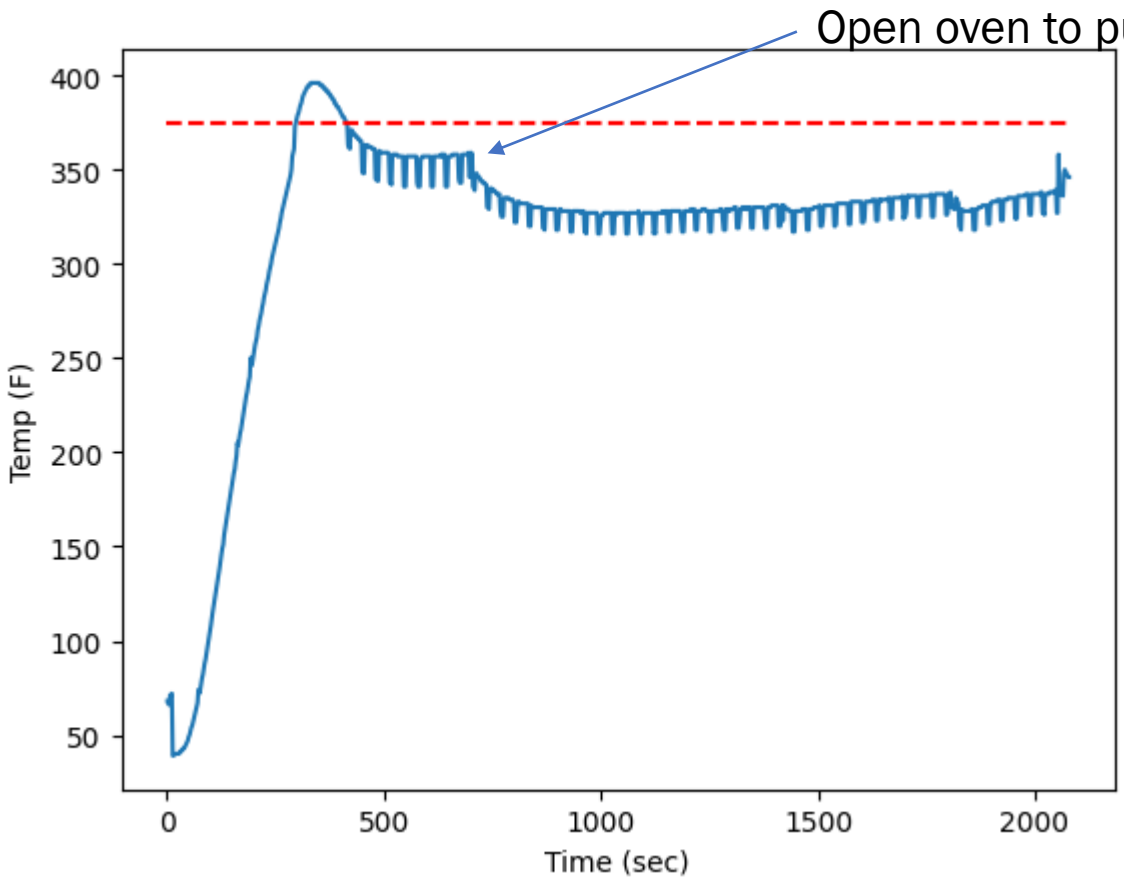
Examples of Global Variables

0x1248 = Top Temp in F

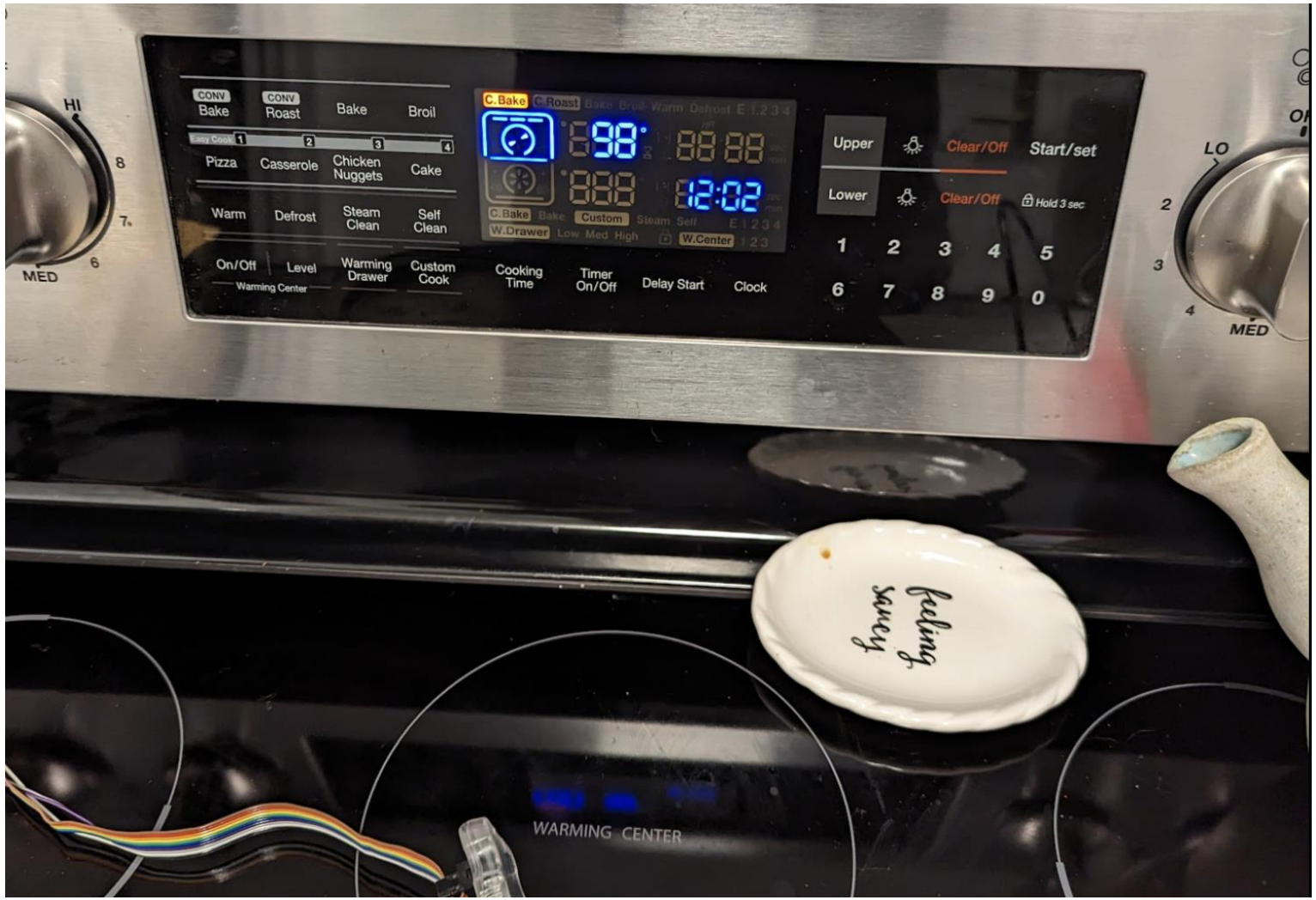
0x120a = Heater “ON” Flag



Set 375F, Cold Start, Load (Shepherds Pie)



Patched Display Logic



New Cooking/Display Logic (old-school thermostat)

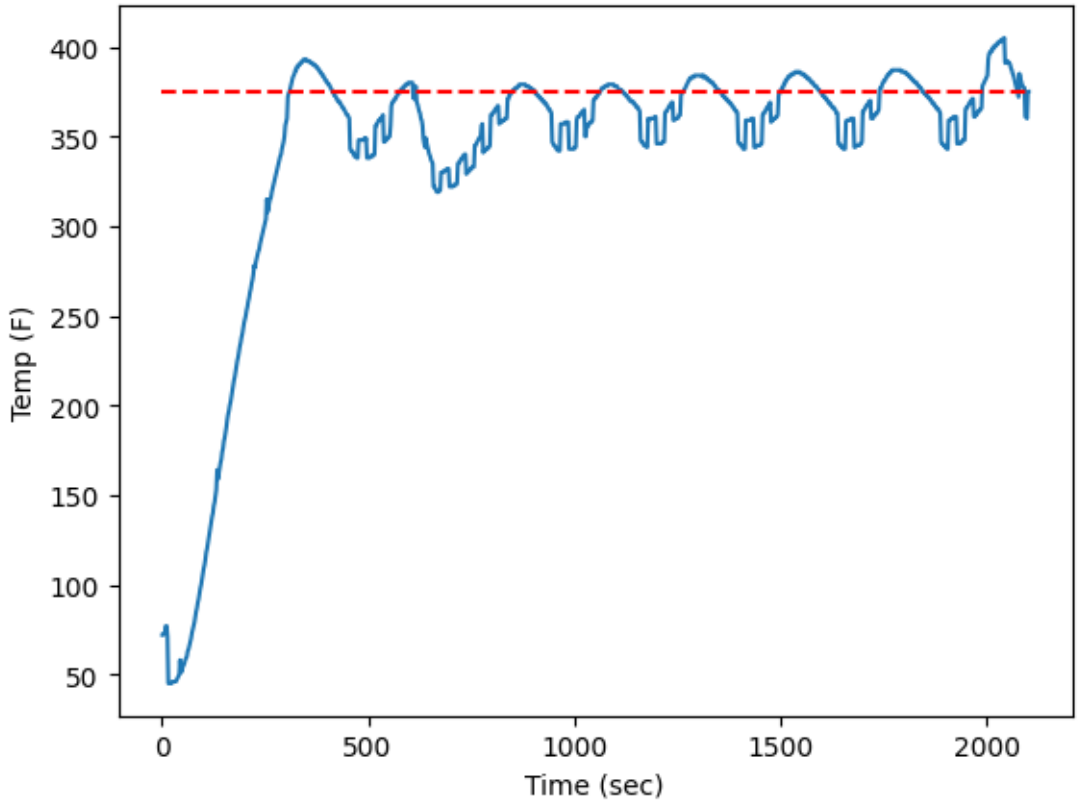
```
if temp < setpoint:
        heater(on)
        display(temp+11)
else:
        heater(off)
        display(temp)
```

JR NC,0x0FE889D	Tested		
LDW WA,IX			
SUBW WA,(0x12A8)			
CPW WA,0x2		From fe885f, insert:	
JR ULE,0x0FE8894	Tested	LDW HL, IX	
CPW IX,(0x12A8)		LDW (0x12A8),HL	
JR ULE,0x0FE888D	Tested	BIT 1, (0x120A)	
BITB 0x6,(0x11CC)		JR NZ, 0x0FE88B0	
JR Z,0x0FE888D	Tested	ADDW HL, 11	dc 8b
INCW 0x2,(0x12A8)		JR 0x0FE88B0	f2 a8 12 00 53
LDW HL,(0x12A8)			f1 0a 12 c8
JR 0x0FE88B0			66 44
LDW HL,IX	<--Patch to jump here from fe885f		db c8 0b 00
LDW (0x12A8),HL			68 3e
JR 0x0FE88B0			

```
0x00fe885f:  LDW   HL,IX                dc 8b
0x00fe8861:  LDW   (0x12A8), HL         f2 a8 12 00 53
0x00fe8866:  bit   0, (0x120A)          f1 0a 12 c8
0x00fe886a:  jr    z, 0xfe88b0          66 44
0x00fe886c:  addw  hl, 11               db c8 0b 00
0x00fe8870:  jr    0xfe88b0             68 3e
0x00fe8872:  ???
```

Code also stops it from going into the "maintain" temperature mode, leaves it in "preheat" mode.

Set 375F, Cold Start, Load (Shepherds Pie)

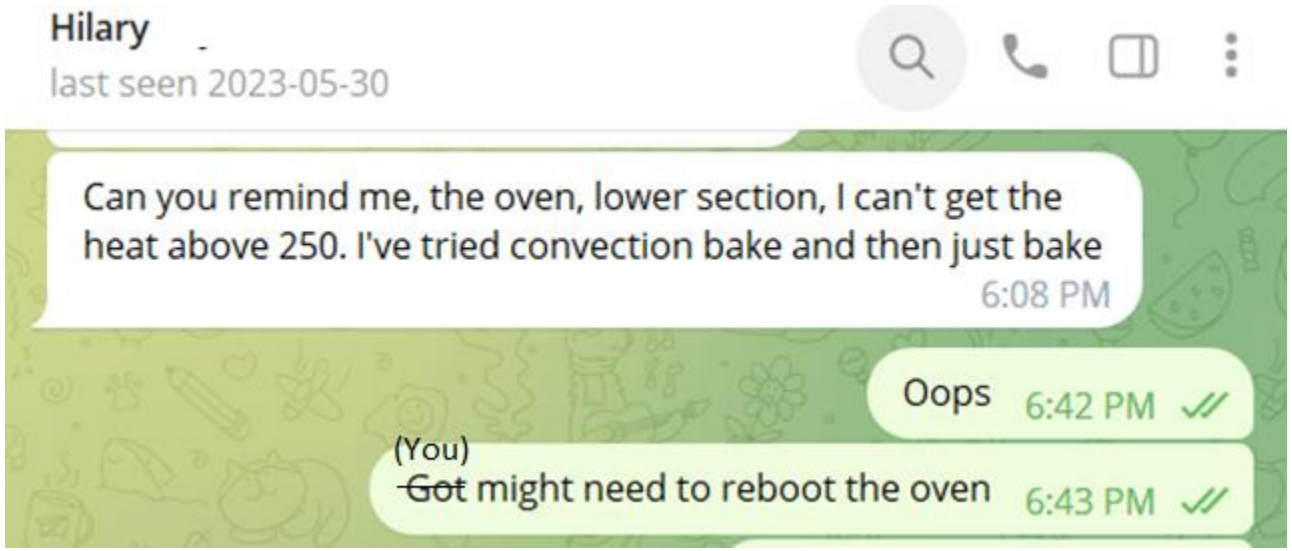


Soufflé Test


WARMING CENTER
Fast Boil

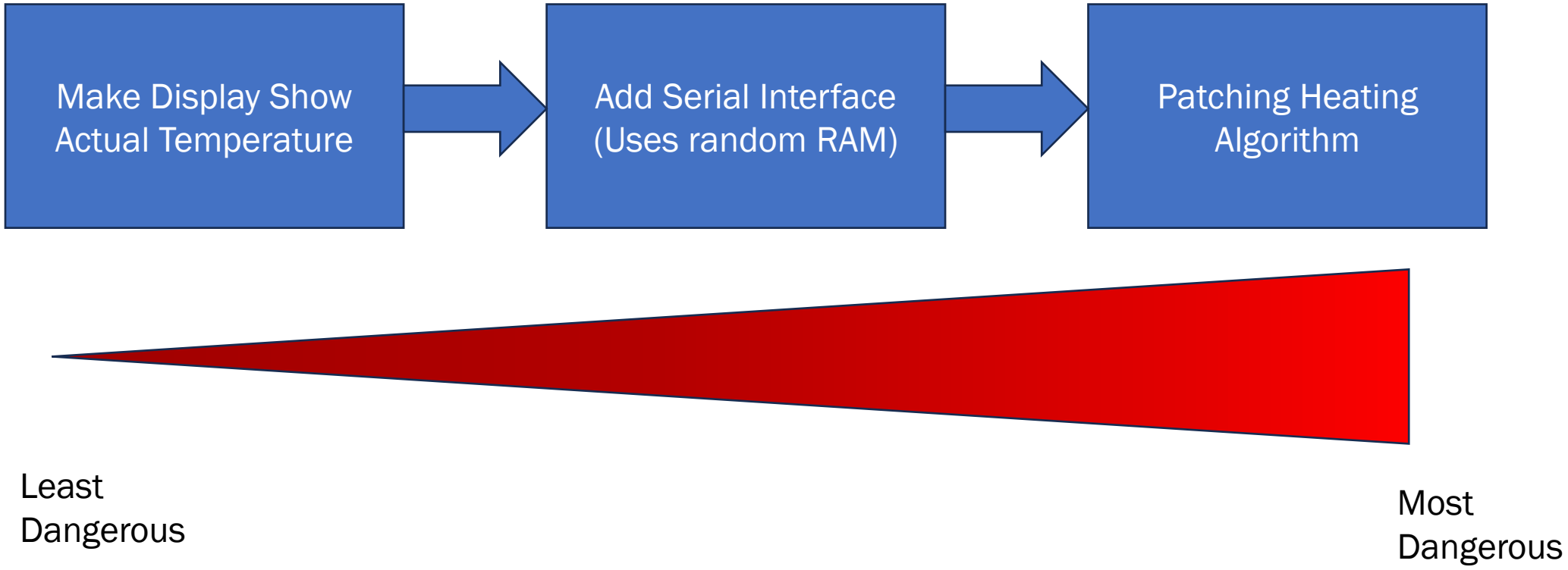
https://www.myrecipes.com/recipe/individual-chocolate-souffl-cakes

Known Bugs



With my patches: after the oven is plugged in for some length of time, seems it stops heating correctly. Need to power cycle at circuit breakers and will work again for a while.

Playing with Your Own Oven



Important Design Reminder

The range elements are knob controlled (mechanical action needed).

The heating elements IN the oven are **100% firmware controlled**.

What I learned?

- Might not be your fault having trouble with receipies & cooking time.
- Many ovens *actively lie to you* to hide their issues.
- Lots of wasted electronic waste generated from this problem (at minimum parts, at worst full ovens).
- Just reflashing boards should be a repair item (but isn’t).

Questions? Details?

https://github.com/colinoflynn/samsung-ovens-deconstructed

https://github.com/colinoflynn/Toshiba-TLCS-900-L-Resources

General overview at blog post on:

https://www.oflynn.com

colinoflynn@bluenoser.me

Hellsite: @colinoflynn