Building Tools for Hacking Deeply Embedded Systems

Travis Goodspeed

Recon, Montréal -- 11 July, 2010

Brief Introduction

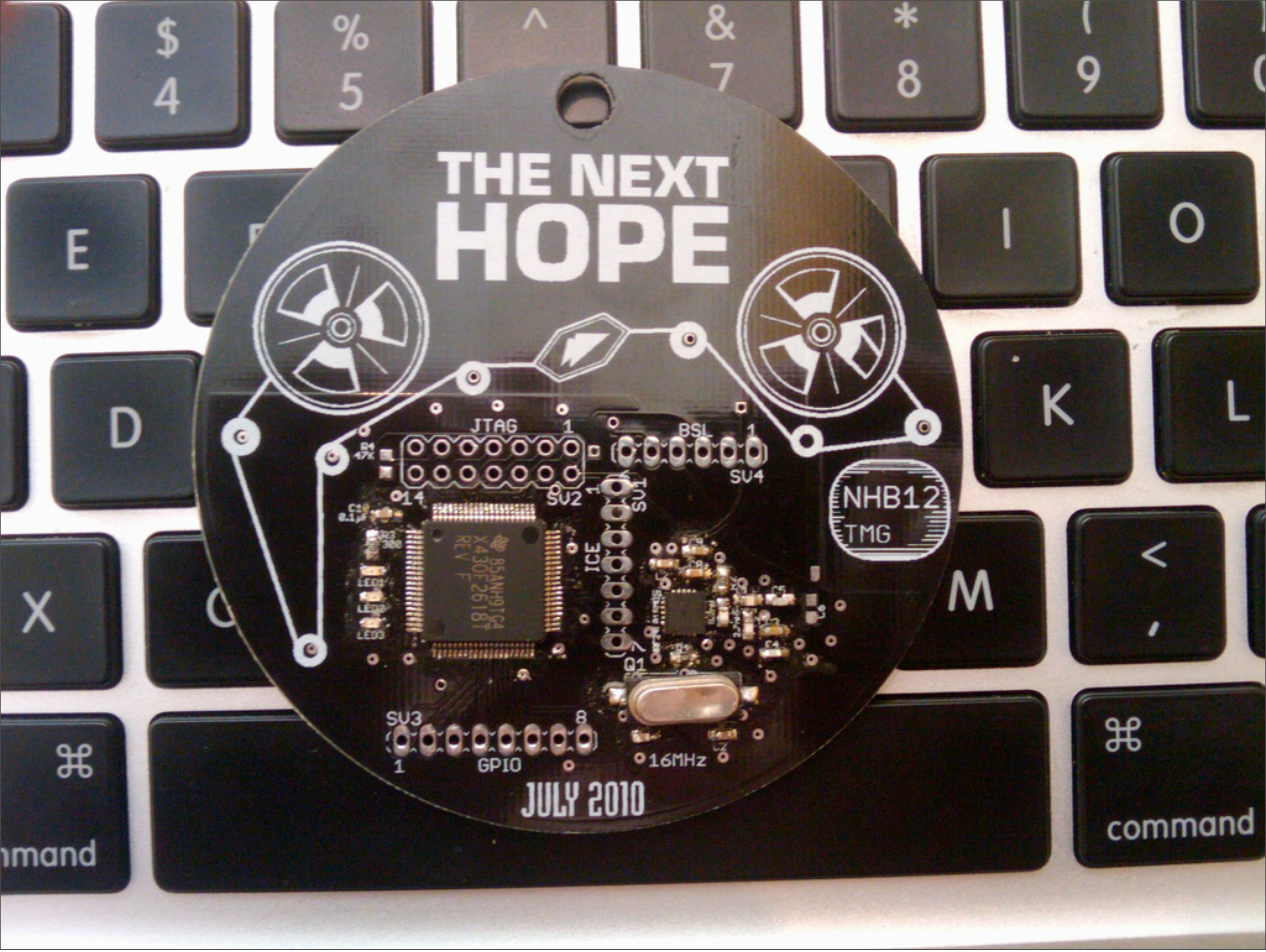
- 8, 16-bit Embedded Systems
  - No operating system, no symbol table, etc.
  - Very different access controls.
- Low-power Radios
  - 0 dBm, small payload, no link layer.

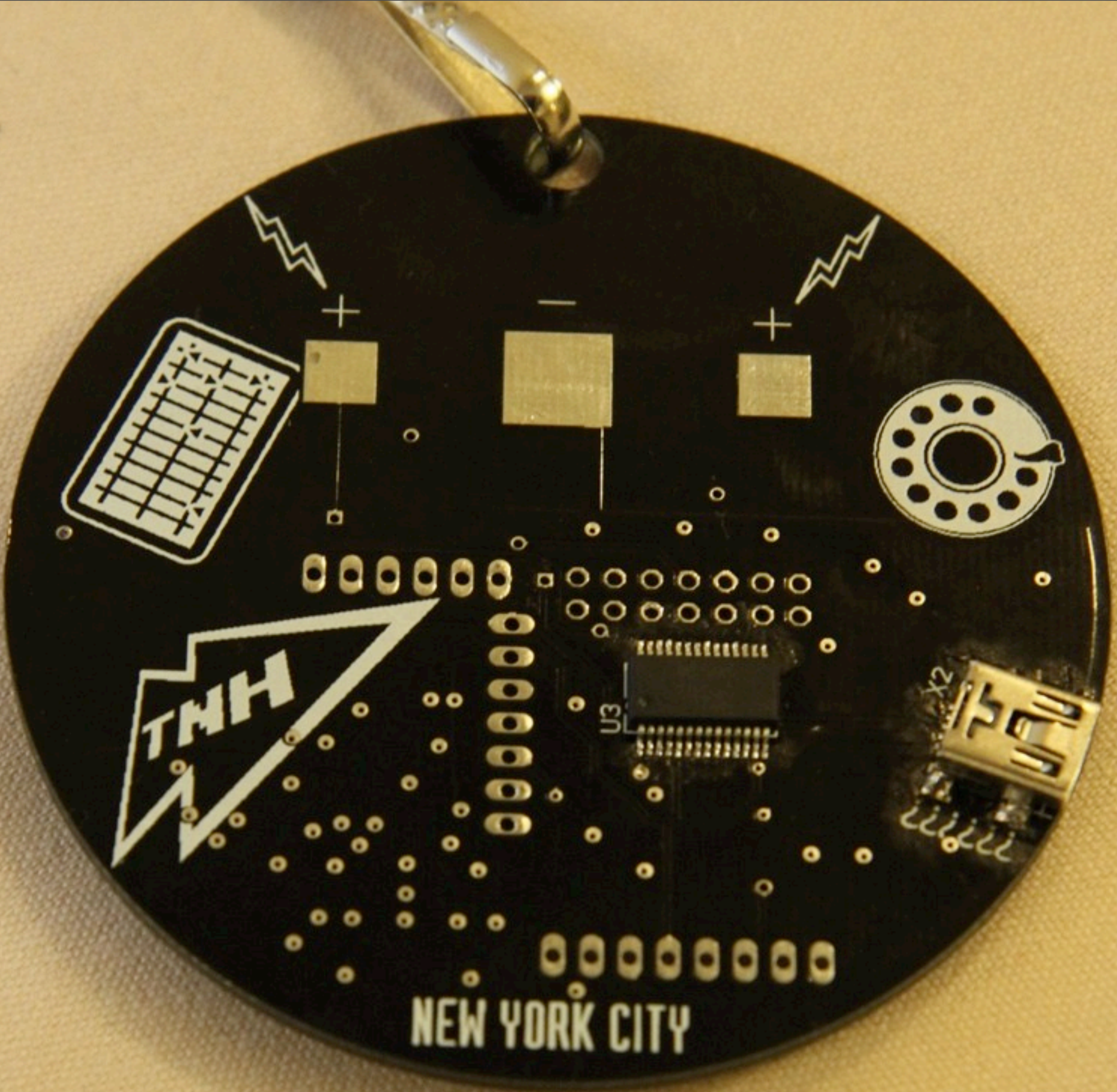
Target Hardware

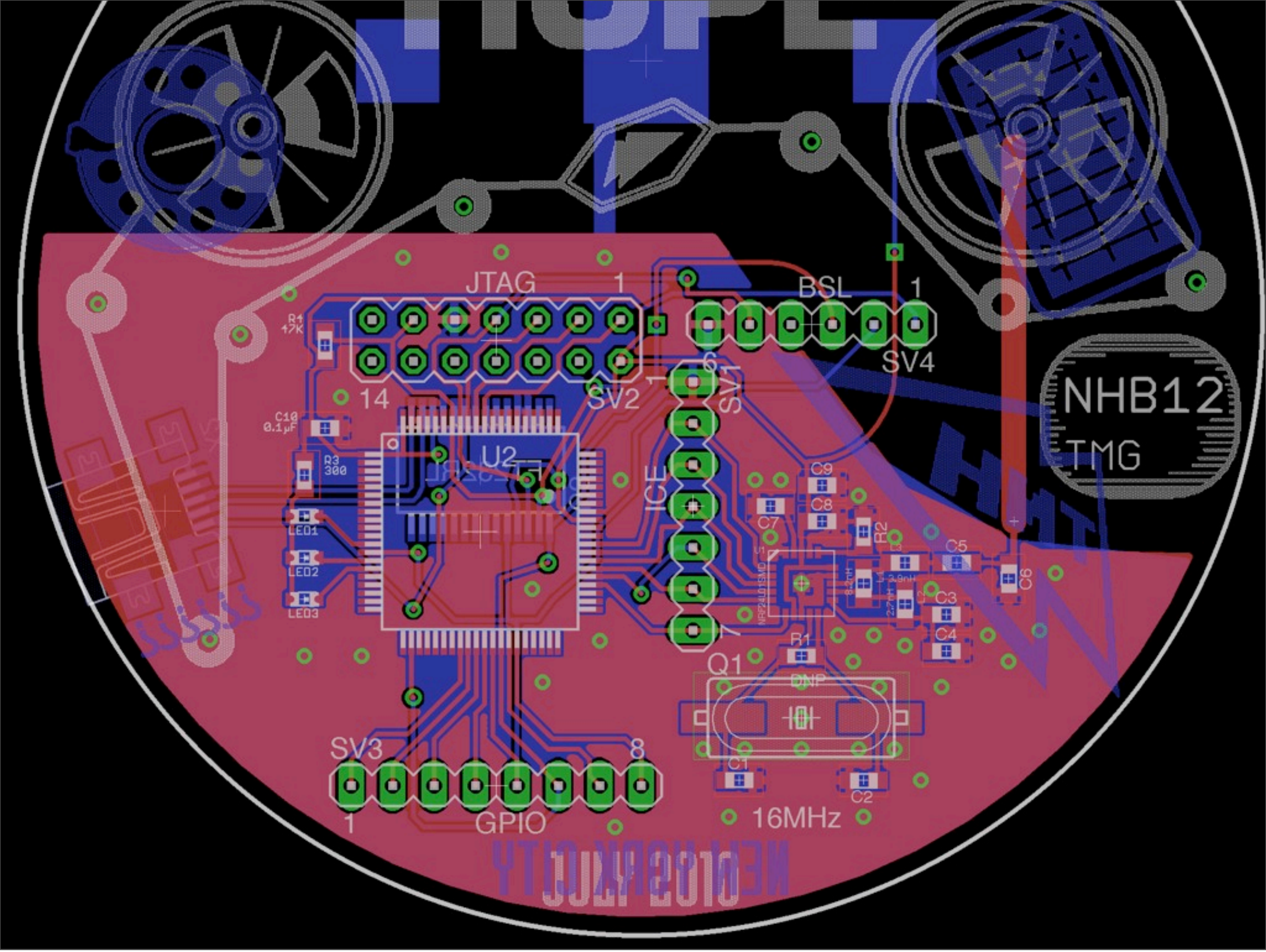
- ZigBee, ANT, 802.15.4, etc
- Wireless Sensor Networks
- Smart Meters
- Sports and Medical Equipment

Show of Hands

- Soldering?
- Intel 8051 or RISC assembly?
- Radio?


THE NEXT
HOPE
JTAG
1
14
SV2
BSL
1
SV4
SV1
ICE
7
NHB12
TMG
LED1
LED2
LED3
X430F2618T
REV F
Q1
16MHz
SV3
8
1
GPIO
JULY 2010


+
−
+
TNH
U3
X2
NEW YORK CITY

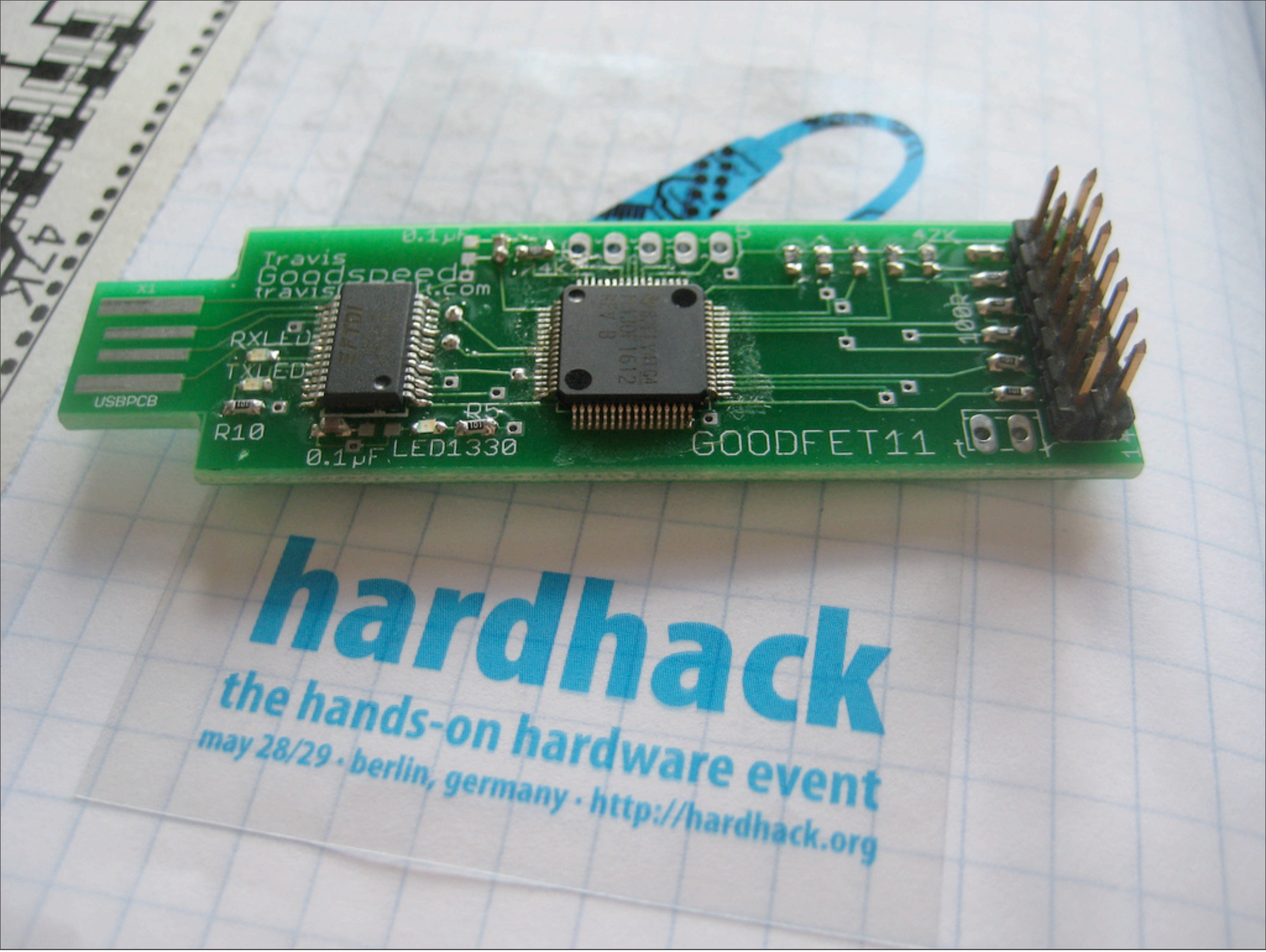

JTAG
1
14
SV2
BSL
1
SV4
6
SV1
1
ICE
7
NHB12
TMG
R1
47K
C10
0.1µF
R3
300
U2
LED1
LED2
LED3
C9
C8
C7
R2
C5
C6
C3
C4
R1
Q1
DNP
C1
C2
16MHz
SV3
8
1
GPIO
NRF24L01SMD
8.2nH
3.9nH
2.2nH

A Lecture in Parts

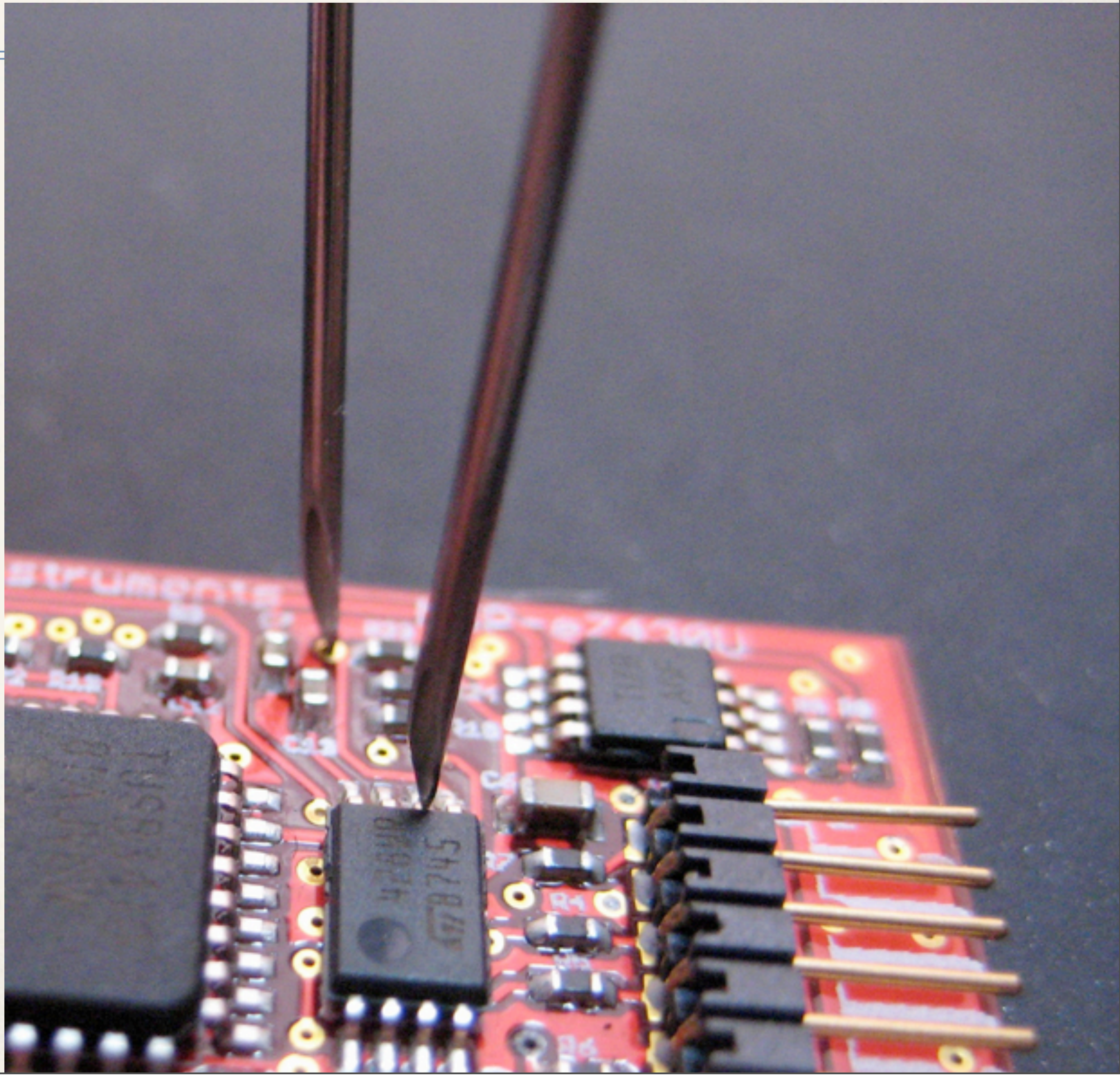
- Part 1: Sniffing a SPI Bus
- Part 2: Reversing a Clicker
- Part 3: Sniffing and Injecting a Clicker
- Some neat tricks.

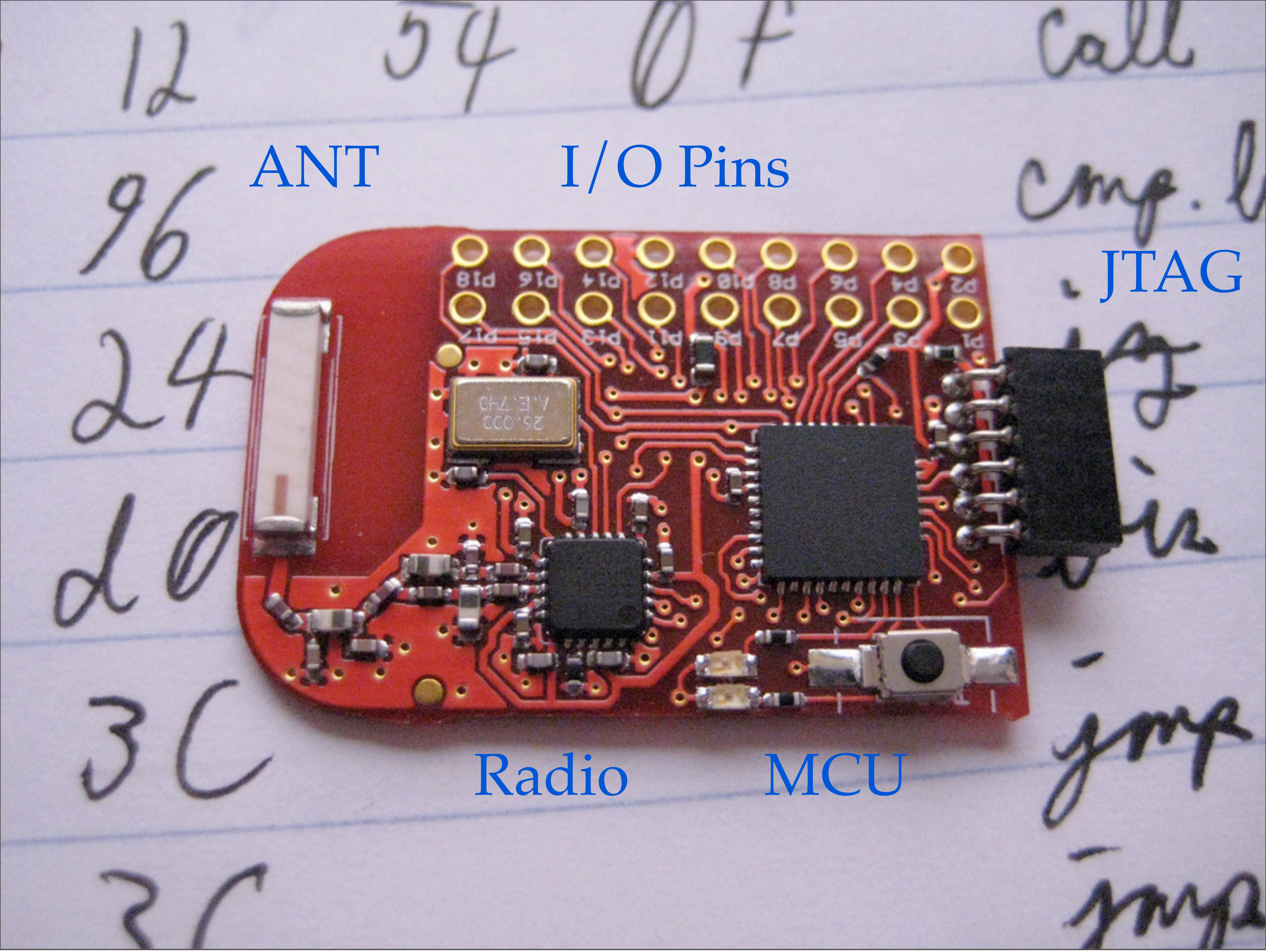
The GoodFET

- Similar to the Bus Pirate, vendor JTAG devices.
- Firmware in C, client in Python.
- Implements dozens of protocols
    - Debugging of 8051, MSP430, ARM.
    - Reading/Writing of SPI, I2C memory chips.
    - Radio access to Nordic RF, Chipcon radios.
- Cheap/Free Boards


Travis
Goodspeed
travisgoodspeed.com
RXLED
TXLED
USBPCB
R10
R5
0.1µF
LED1330
GOODFET11
100R
47K
hardhack
the hands-on hardware event
may 28/29 · berlin, germany · http://hardhack.org

Part 1: Tapping a SPI Bus





ANT

I/O Pins

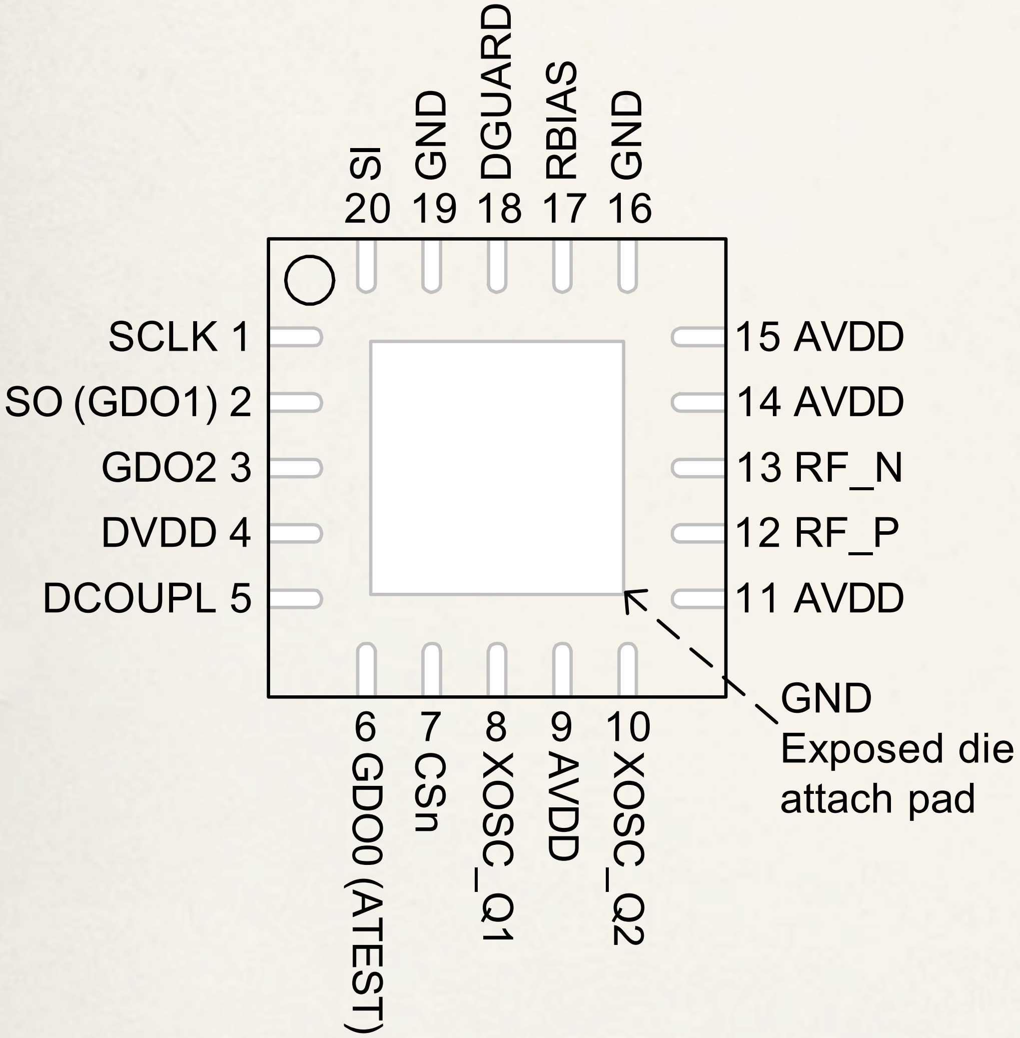
JTAG

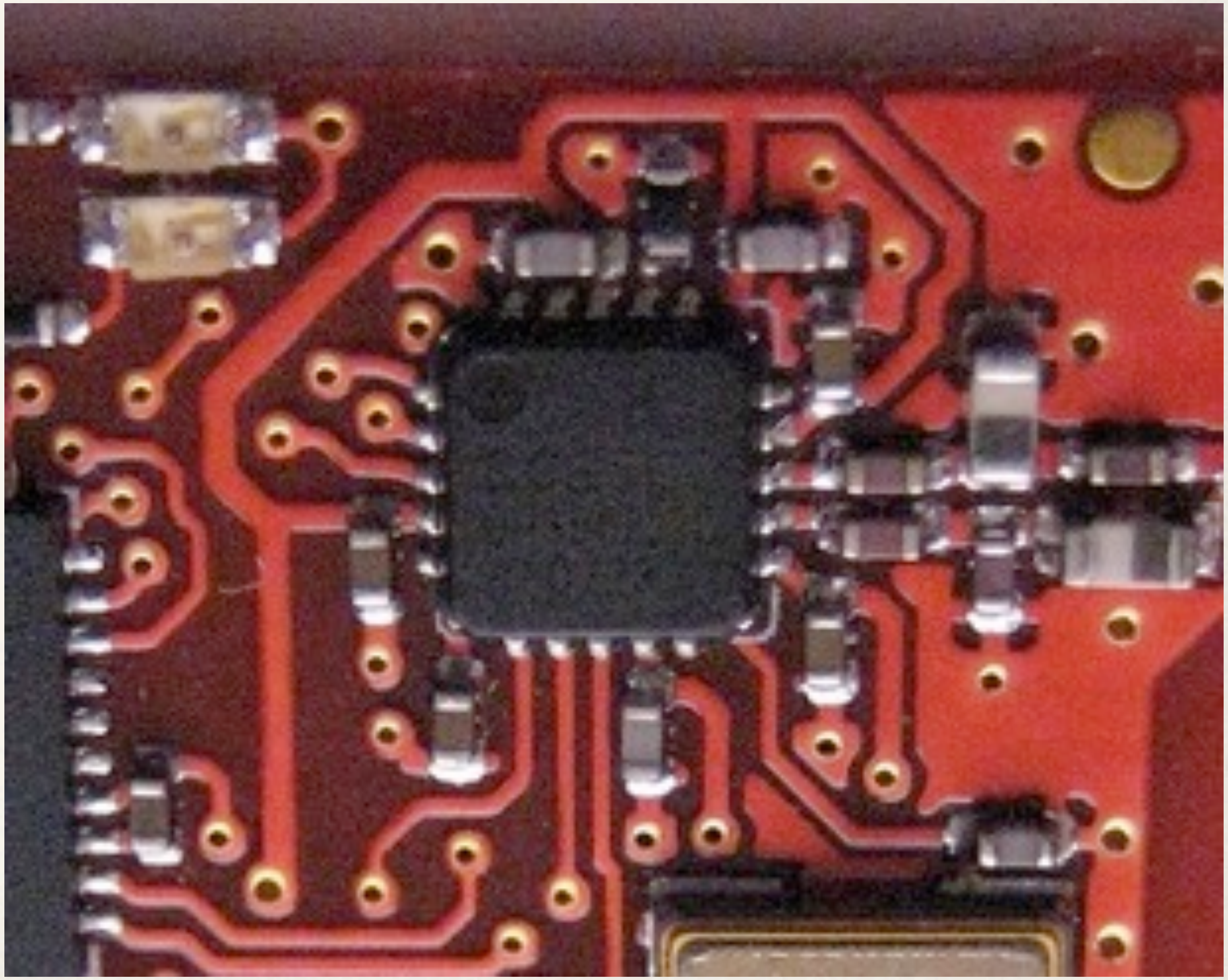
Radio

MCU

12 54 0F call
96 cmp.l
24
20
3C jmp
3C jmp
P18 P16 P14 P12 P10 P8 P6 P4 P2
P17 P15 P13 P11 P9 P7 P5 P3 P1
26.000

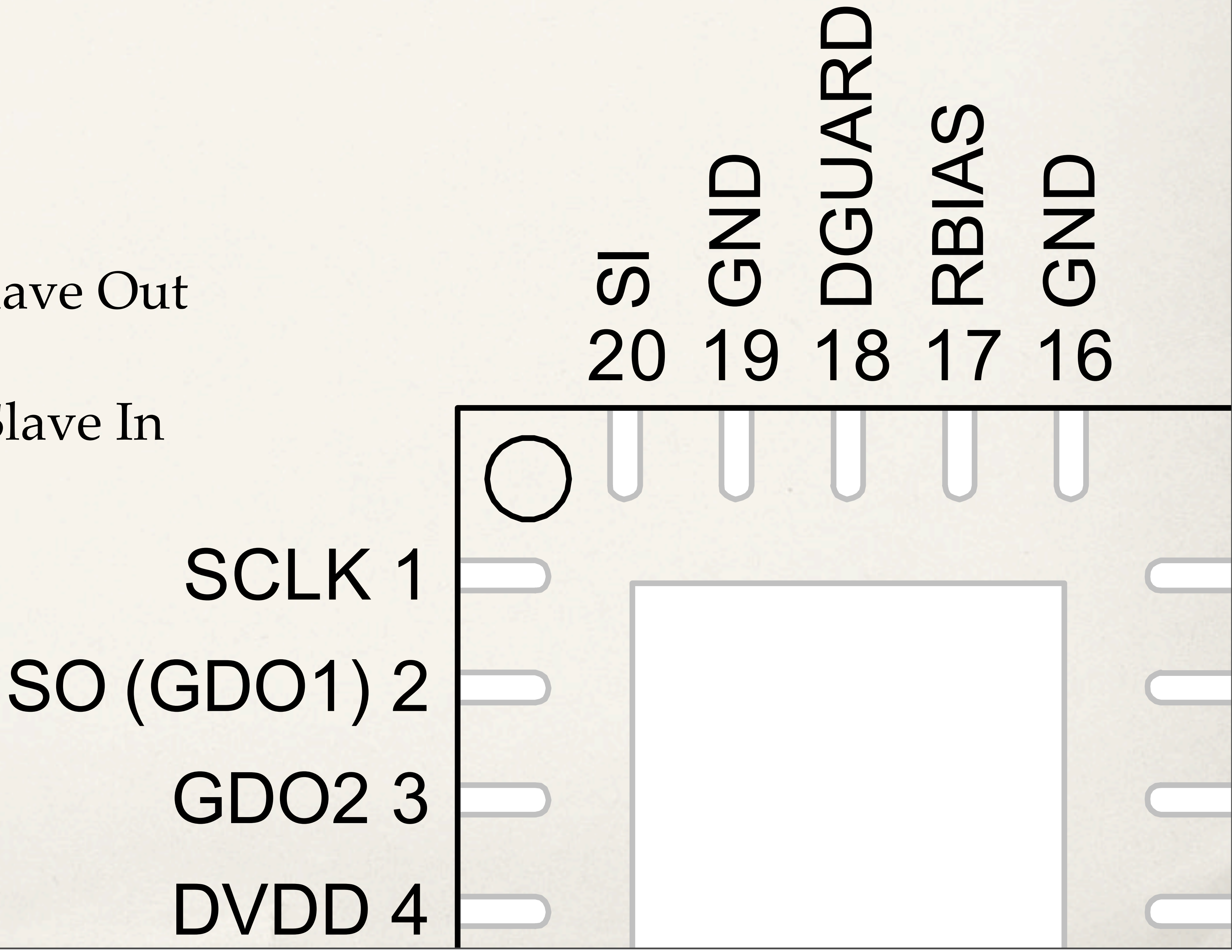
Pin Identification

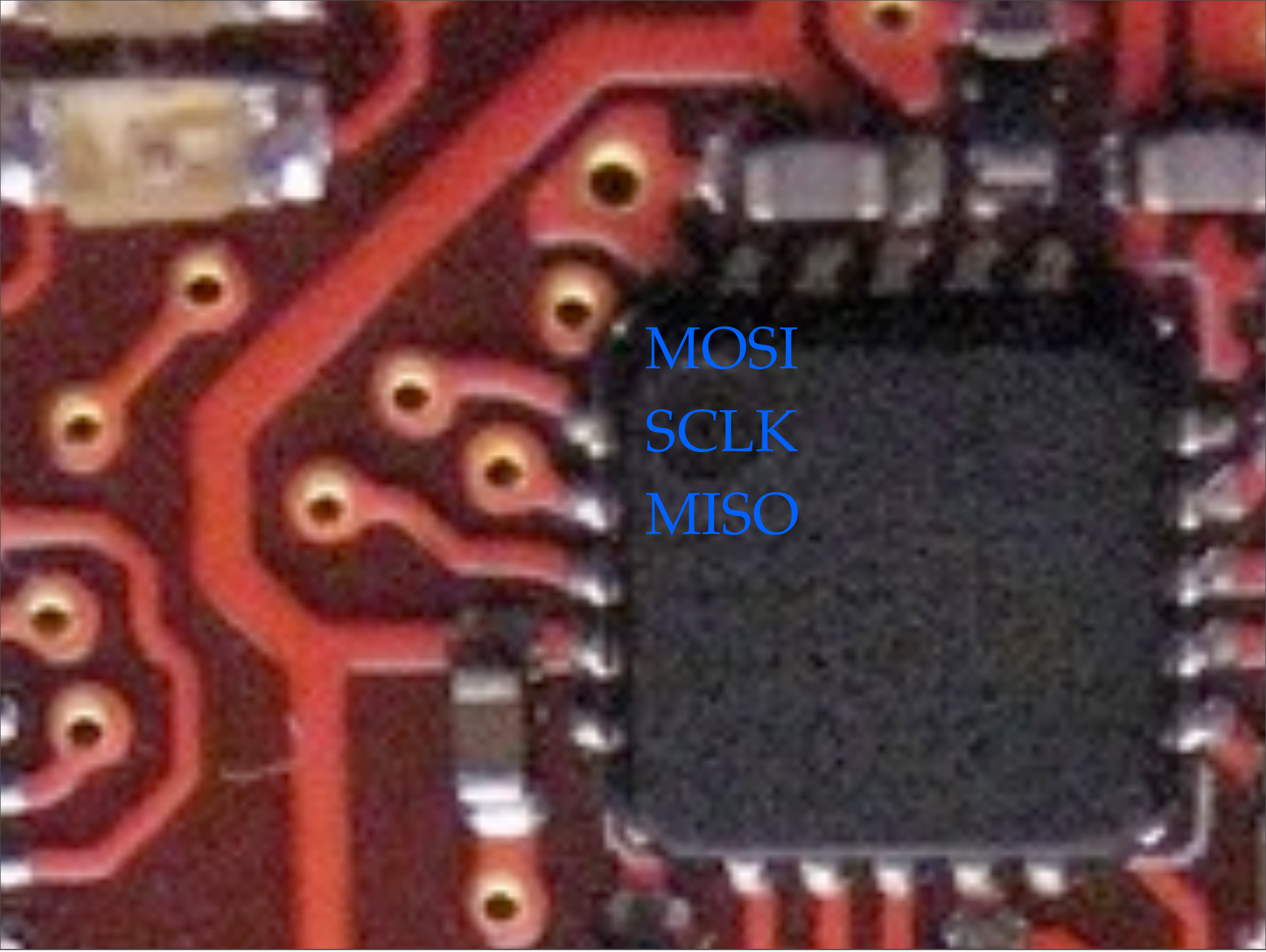


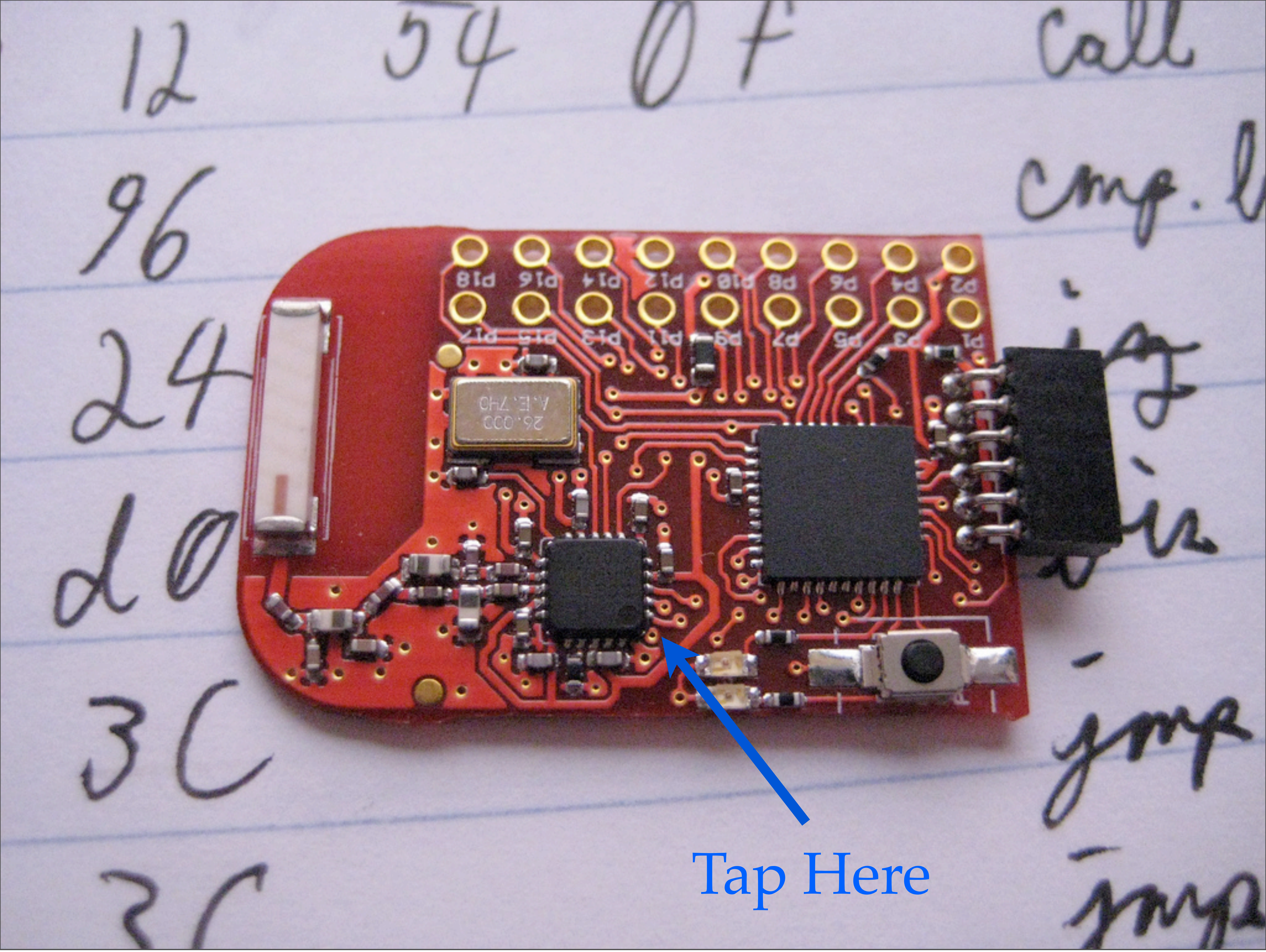


SPI Bus Pins

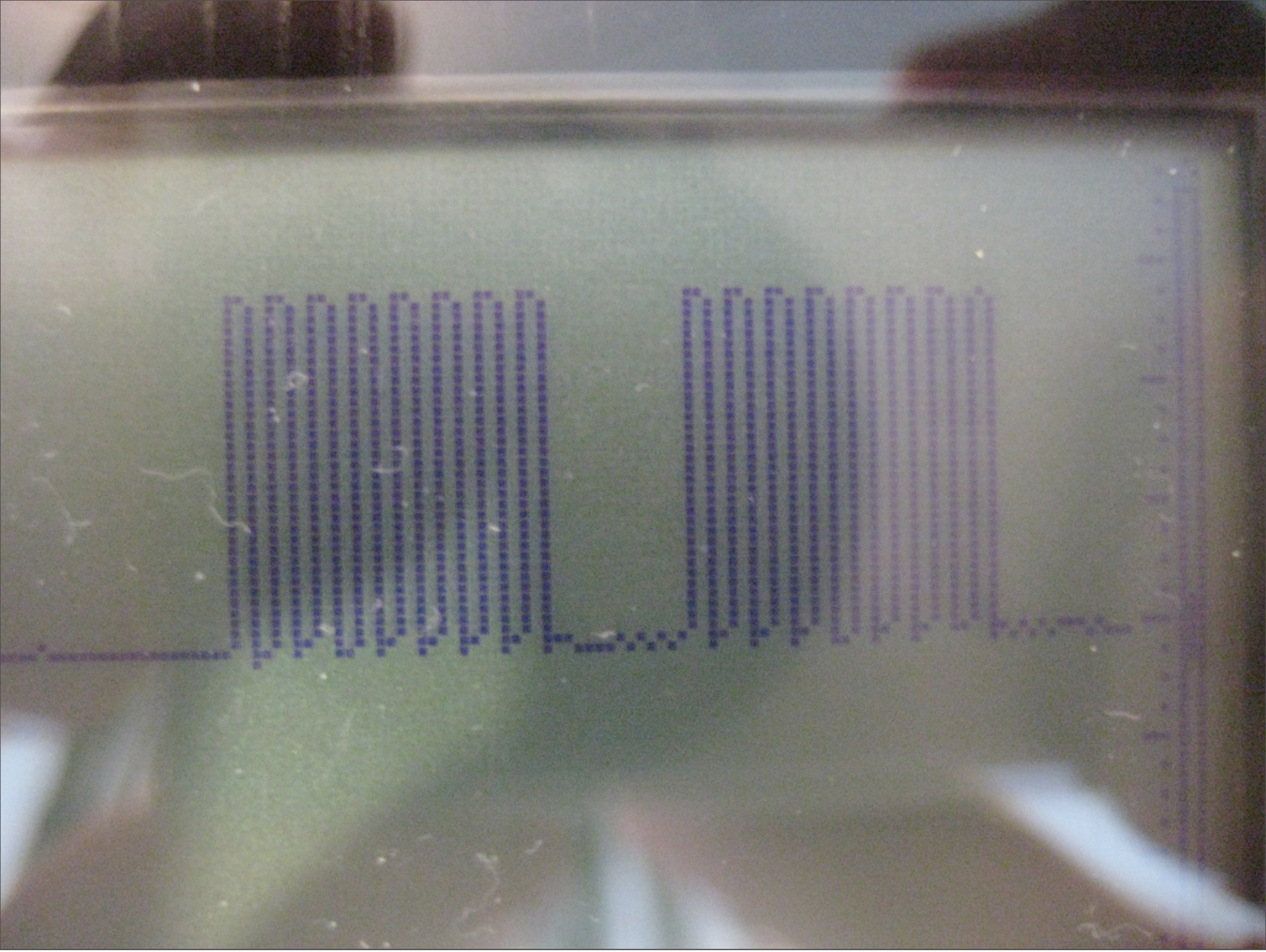
- SO -- Master In Slave Out
- SI -- Master Out Slave In
- SCLK -- Clock

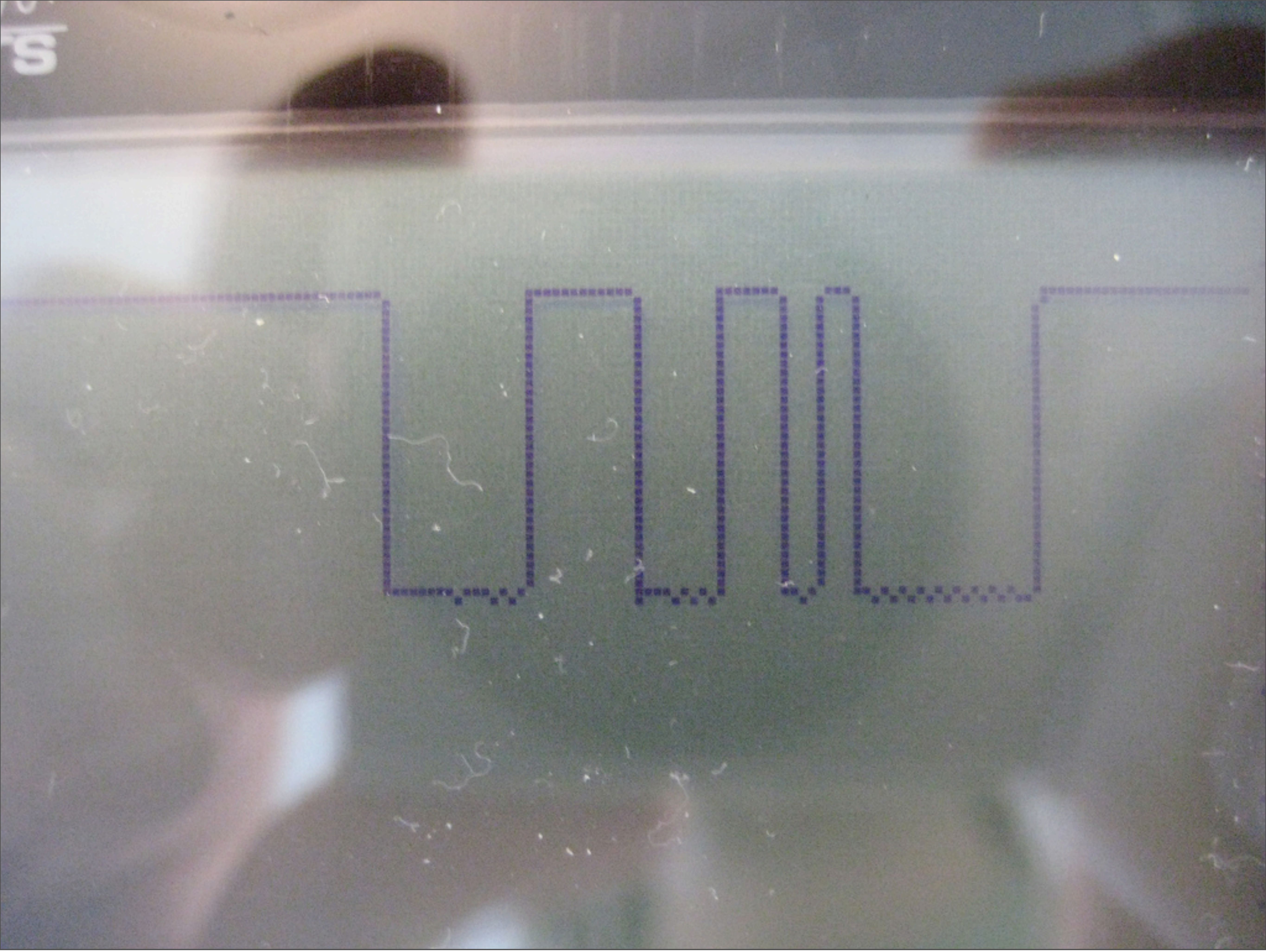


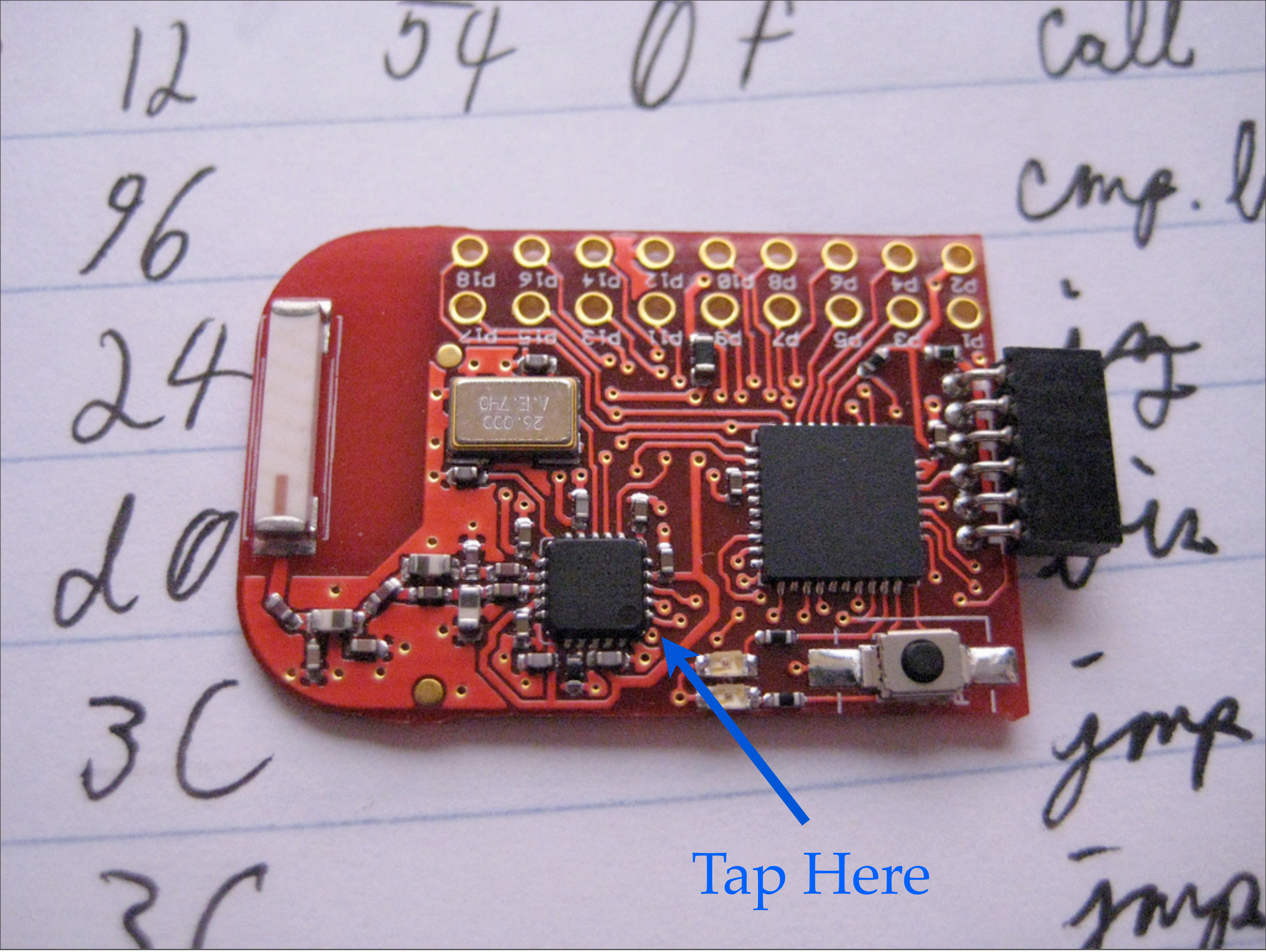

MOSI
SCLK
MISO


12 54 0F call
96 cmp.l
24
d0
3C jmp
3C jmp
P18 P16 P14 P12 P10 P8 P6 P4 P2
P17 P15 P13 P11 P9 P7 P5 P3 P1
26.000
A.IE.7H0

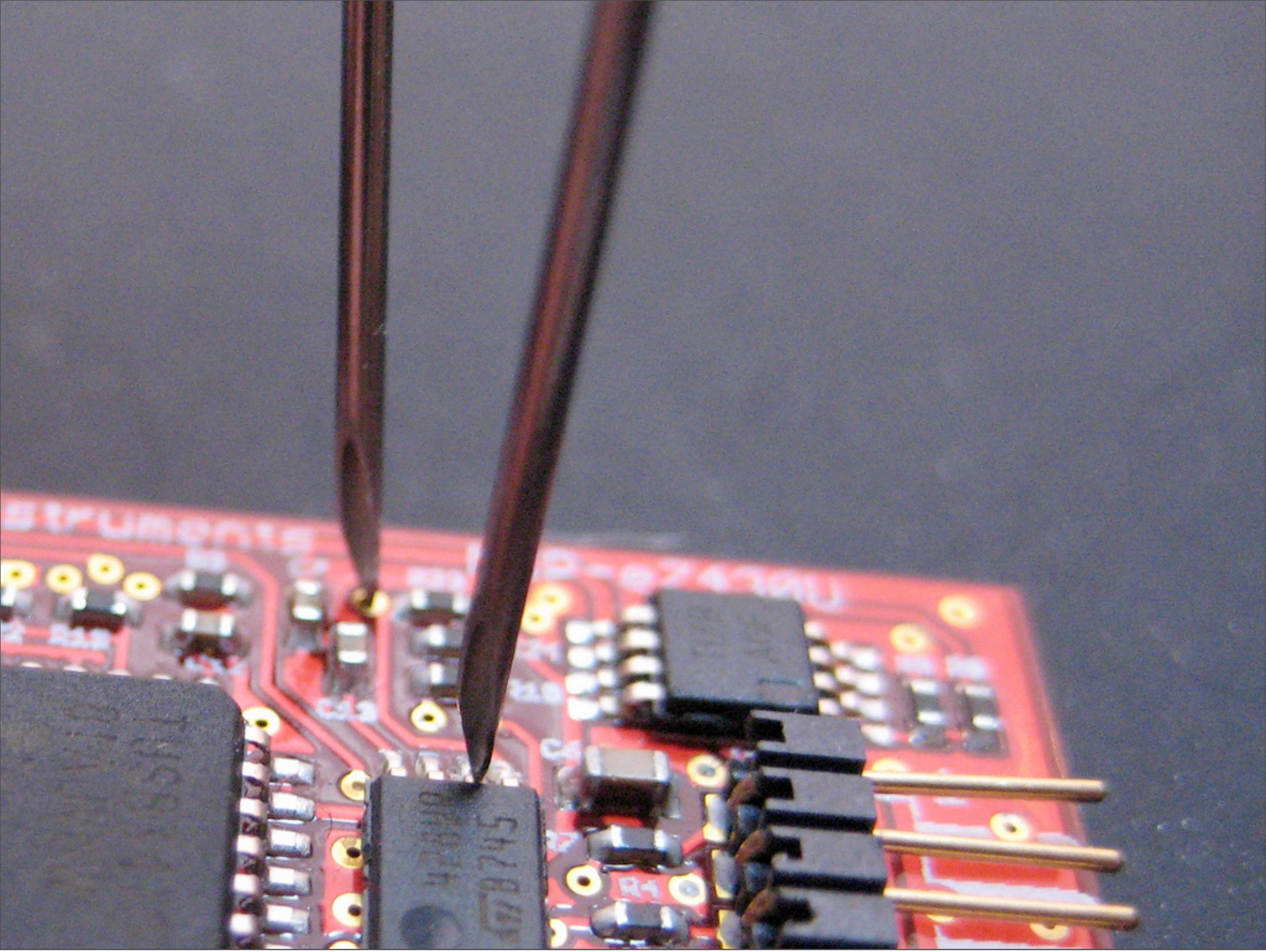
Tap Here





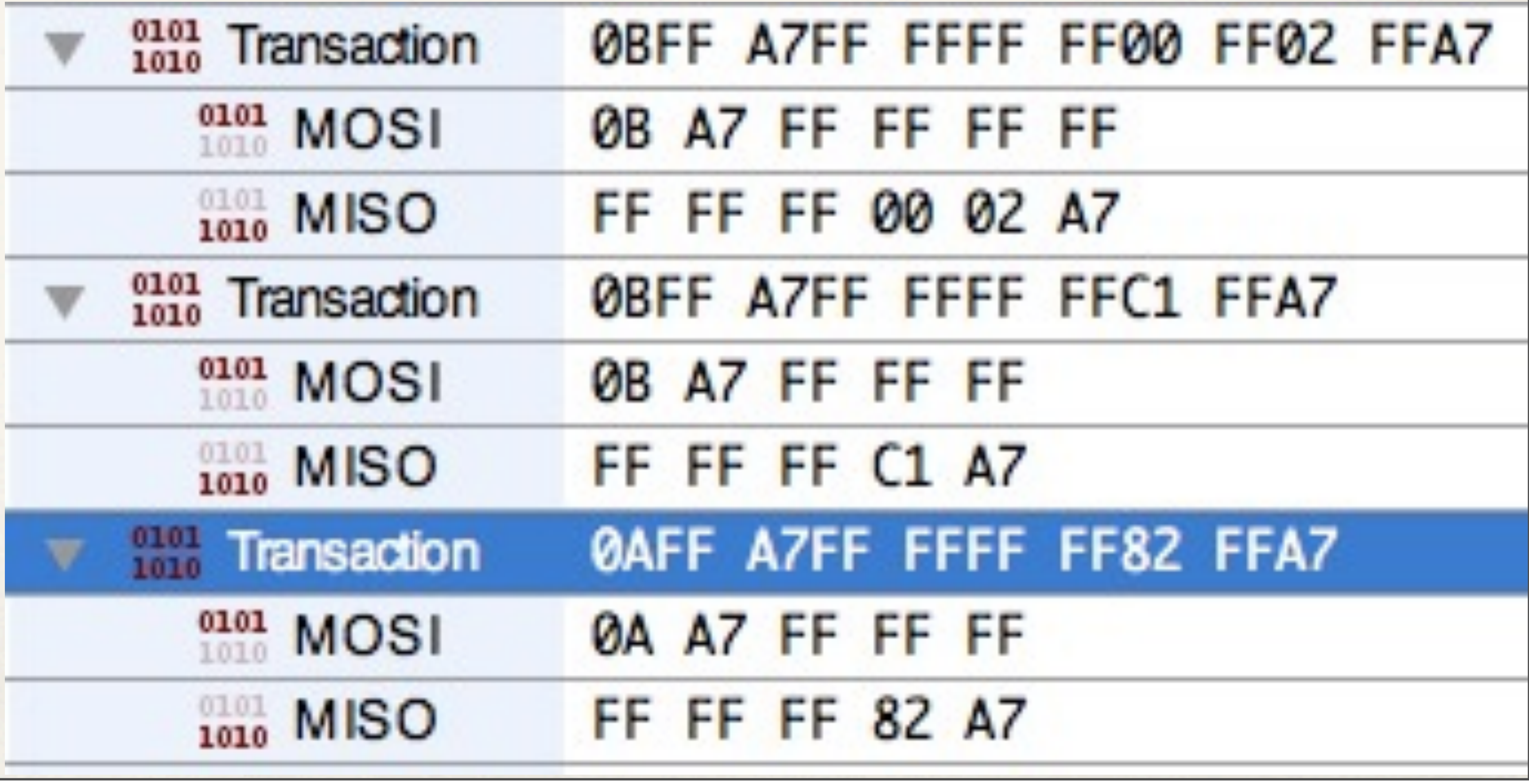
12
54
0F
call
96
cmp.l
24
20
3C
jmp
3C
jmp
P18
P16
P14
P12
P10
P8
P6
P4
P2
P17
P15
P13
P11
P9
P7
P5
P3
P1
26.000

Tap Here


struments
C12
C4
R1

SPI Radio Bus Tap

- Sort of like tapping a driver.
- Commands vary by chip.
  - Read/Write Register
  - TX Packet
  - RX Packet

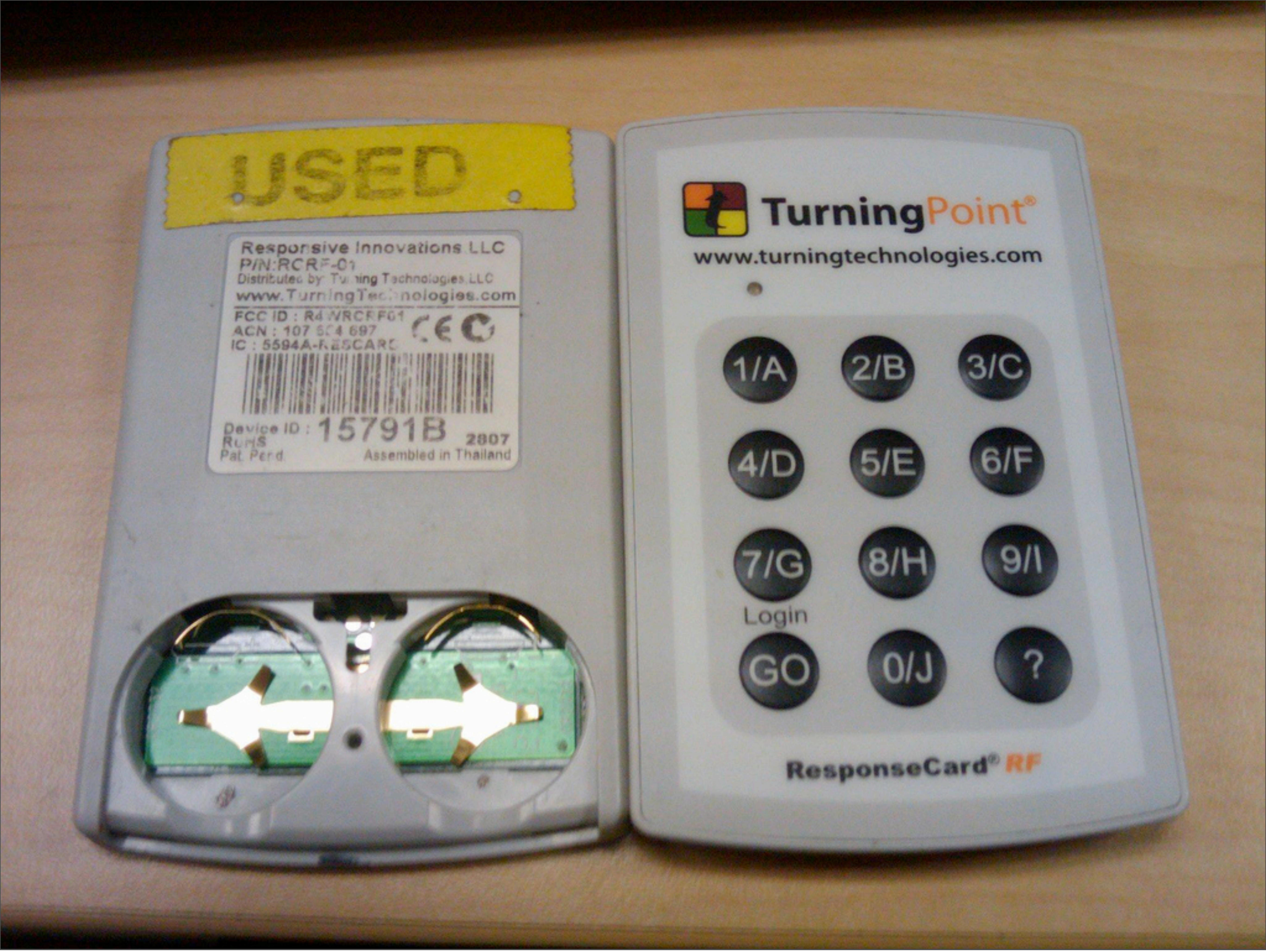


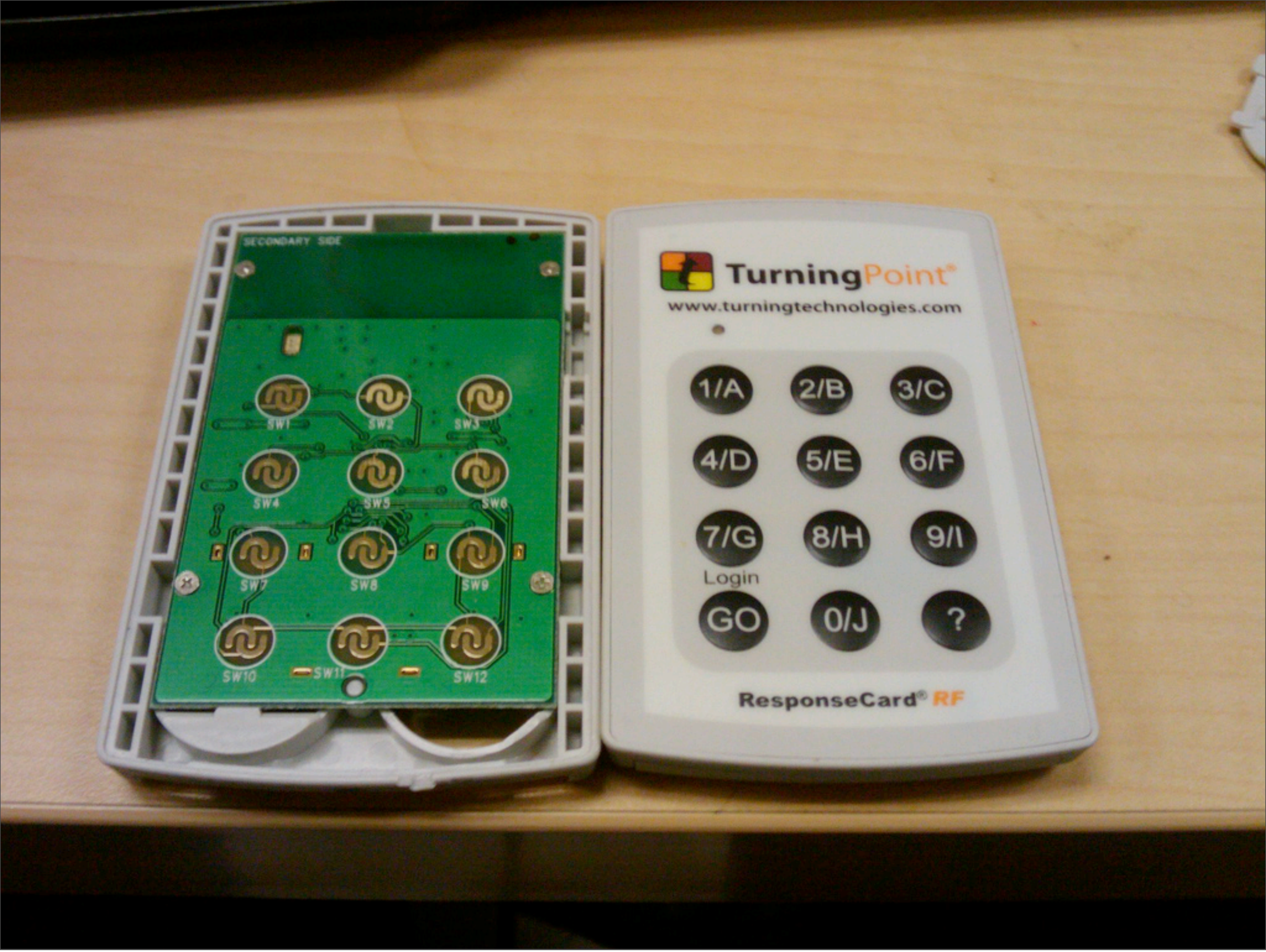
```
▼ 0101 Transaction   0BFF A7FF FFFF FF00 FF02 FFA7
  1010
       0101 MOSI     0B A7 FF FF FF FF
       0101 MISO     FF FF FF 00 02 A7
▼ 0101 Transaction   0BFF A7FF FFFF FFC1 FFA7
  1010
       0101 MOSI     0B A7 FF FF FF
       0101 MISO     FF FF FF C1 A7
▼ 0101 Transaction   0AFF A7FF FFFF FF82 FFA7
  1010
       0101 MOSI     0A A7 FF FF FF
       0101 MISO     FF FF FF 82 A7
```

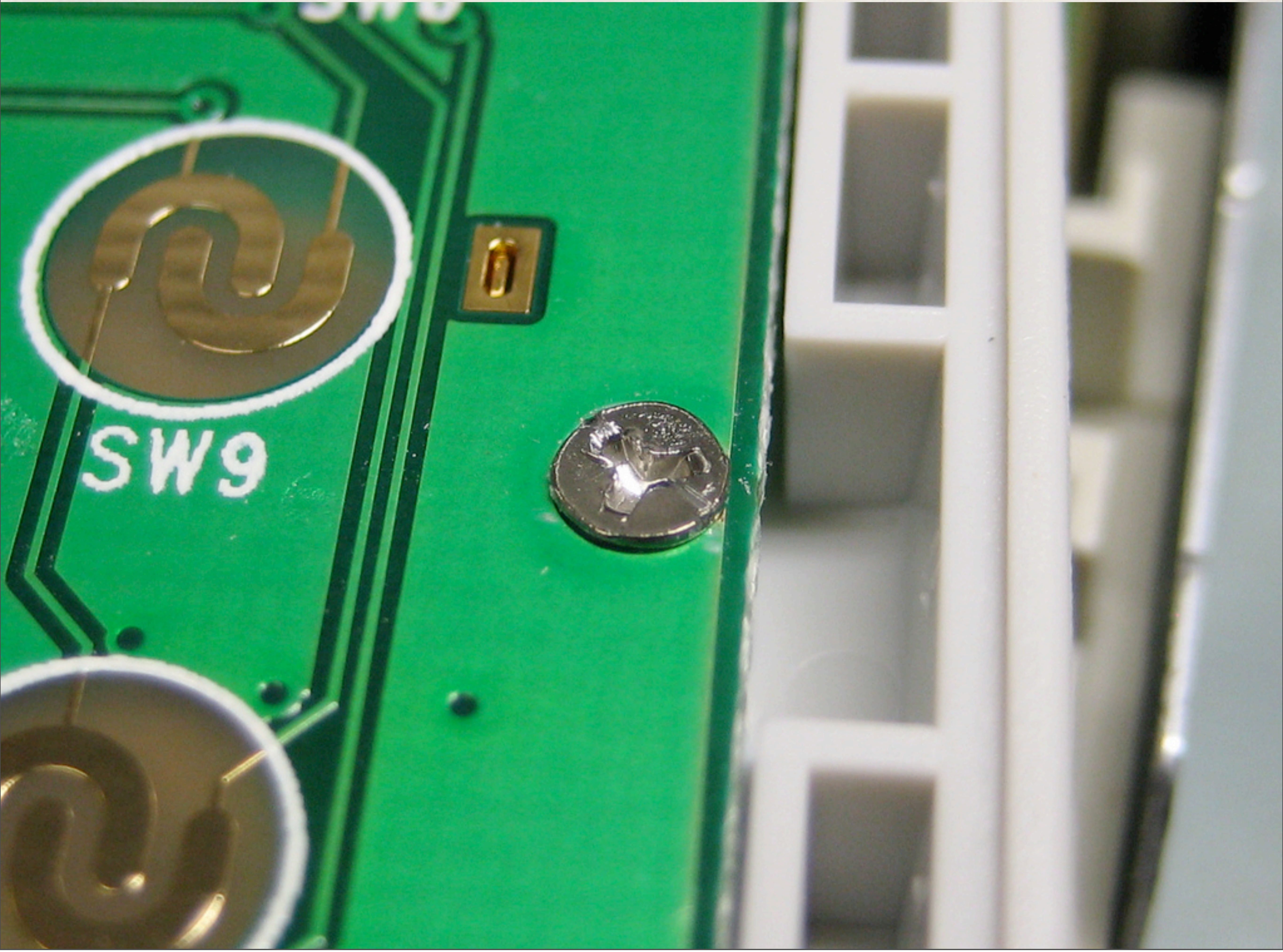
SPI Bus Tap Results

- Which frequency, modulation, MAC addresses, etc are used.
  - Enough to packet sniff, usually.
- Which AES keys are used.
  - KEY[0]=98aceb47c26450ee85292d0c8ce55292
  - KEY[1]=7b8397ddacac7e429ba6f49cbd2c69b1
- Very useful for channel hopping devices.

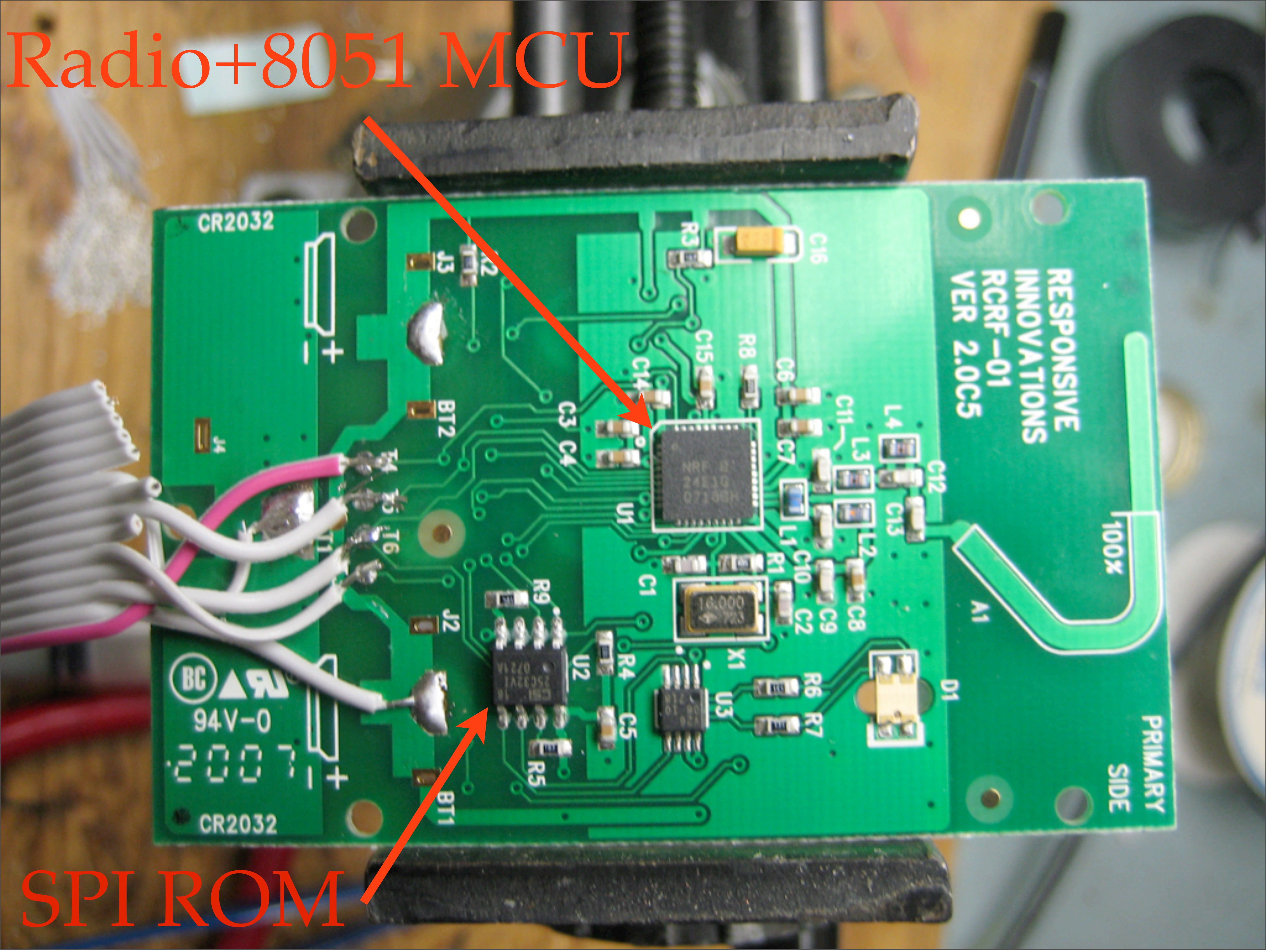
Part 2: Reversing a Clicker


USED
Responsive Innovations LLC
P/N:RCRF-01
Distributed by: Turning Technologies LLC
www.TurningTechnologies.com
FCC ID : R4WRCRF01
ACN : 107 504 697
IC : 5594A-RESCARD
Device ID : 15791B
2807
RoHS
Pat. Pend.
Assembled in Thailand
TurningPoint®
www.turningtechnologies.com
1/A
2/B
3/C
4/D
5/E
6/F
7/G
8/H
9/I
Login
GO
0/J
?
ResponseCard® RF


SECONDARY SIDE
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
SW10
SW11
SW12
TurningPoint®
www.turningtechnologies.com
1/A
2/B
3/C
4/D
5/E
6/F
7/G
8/H
9/I
Login
GO
0/J
?
ResponseCard® RF



SW9



Radio+8051 MCU

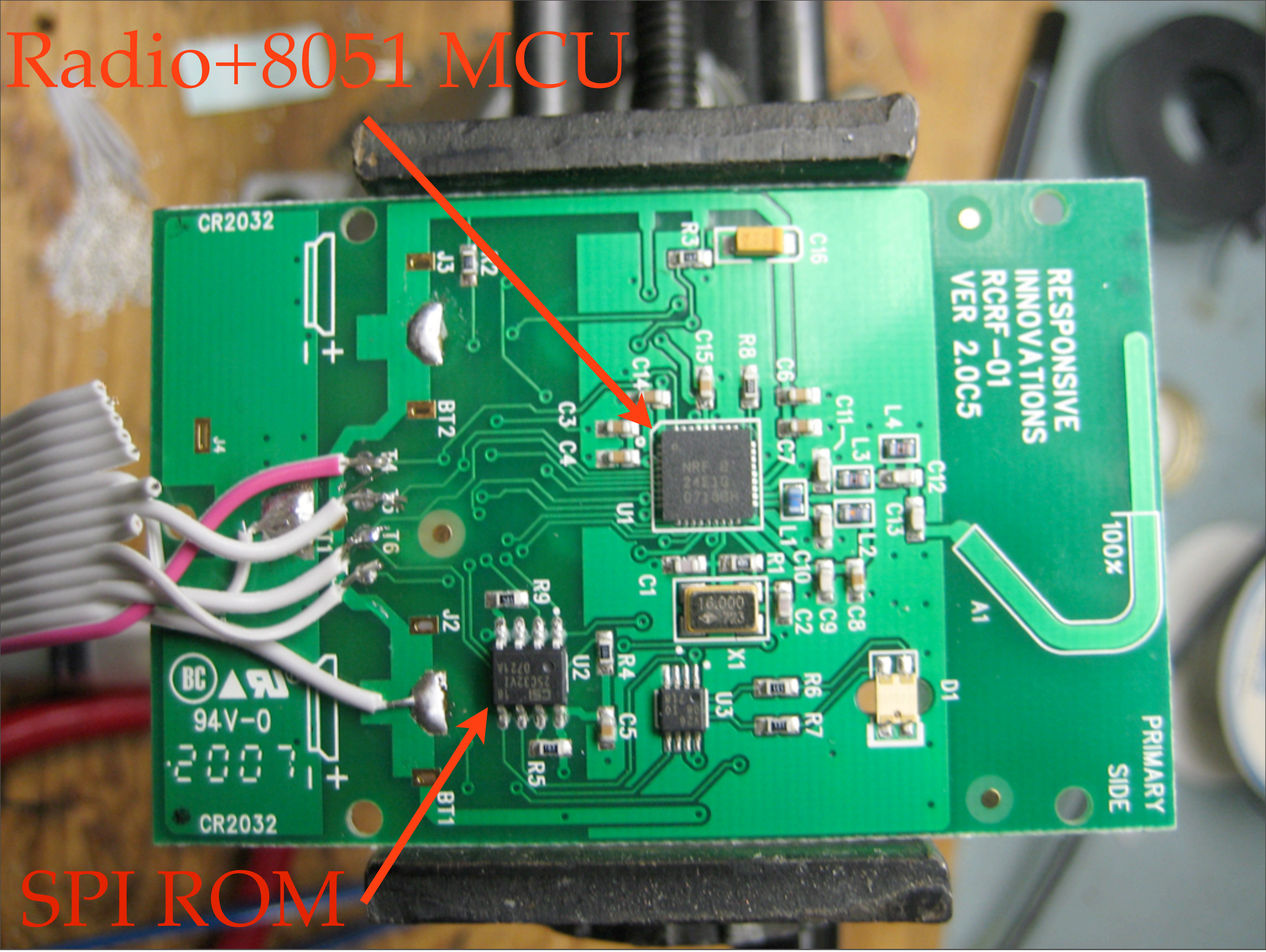
SPI ROM

Dumping Firmware

- Chips
  - nRF24E1G -- 8051 MCU + nRF2401 Radio
  - 24C32 Boot Rom
- Documentation
  - Datasheets, Reference Design

nRF24E1

- 8051 Microcontroller
  - More popular than ARM and X86.
- Internal nRF2401 Radio
  - 1Mbps GFSK Radio
  - 2.4 to 2.5 GHz, 1MHz Channel Spacing
- No internal Flash. Boots from external EEPROM.



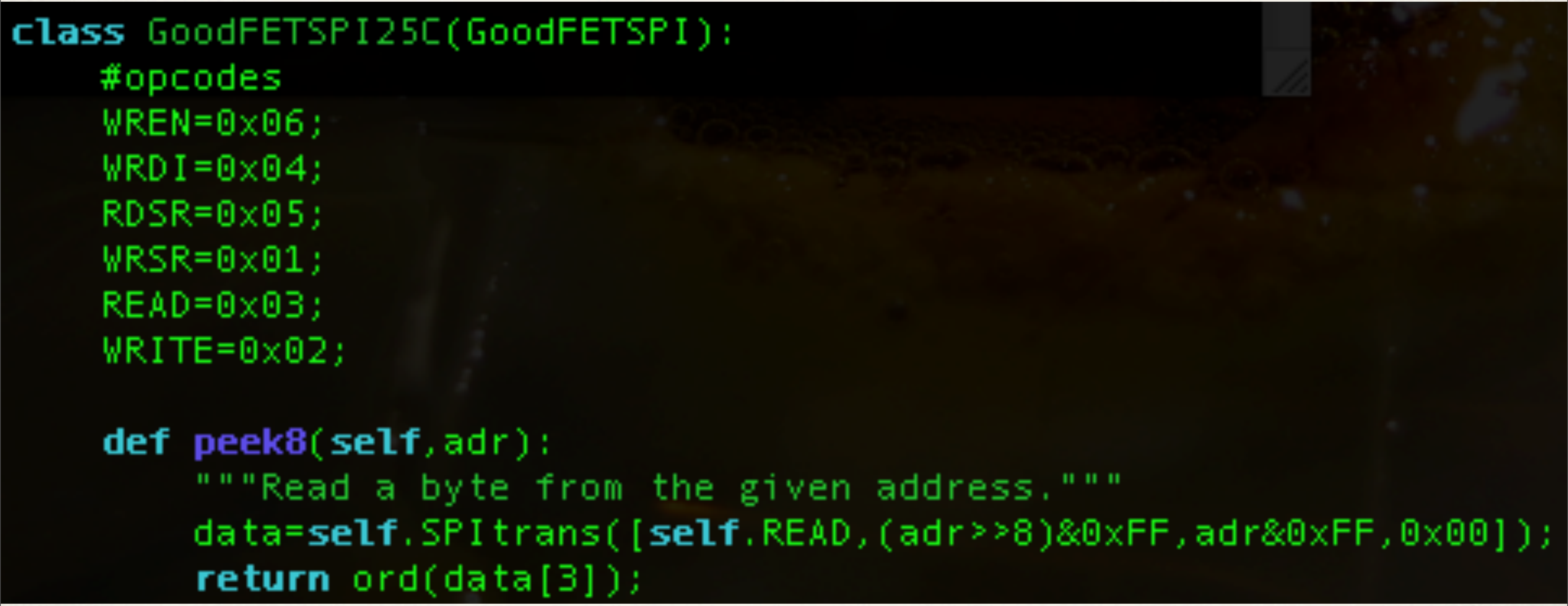
Radio+8051 MCU

SPI ROM

Dumping the 25C32 SPI EEPROM

- Serial Peripheral Interface Bus
  - START, bytes, STOP
  - Input and Output at the same time.
- To read a byte,
  - TX {0x03, LA, HA, 0x00}
  - RX {0xFF, 0xFF, 0xFF, byte}

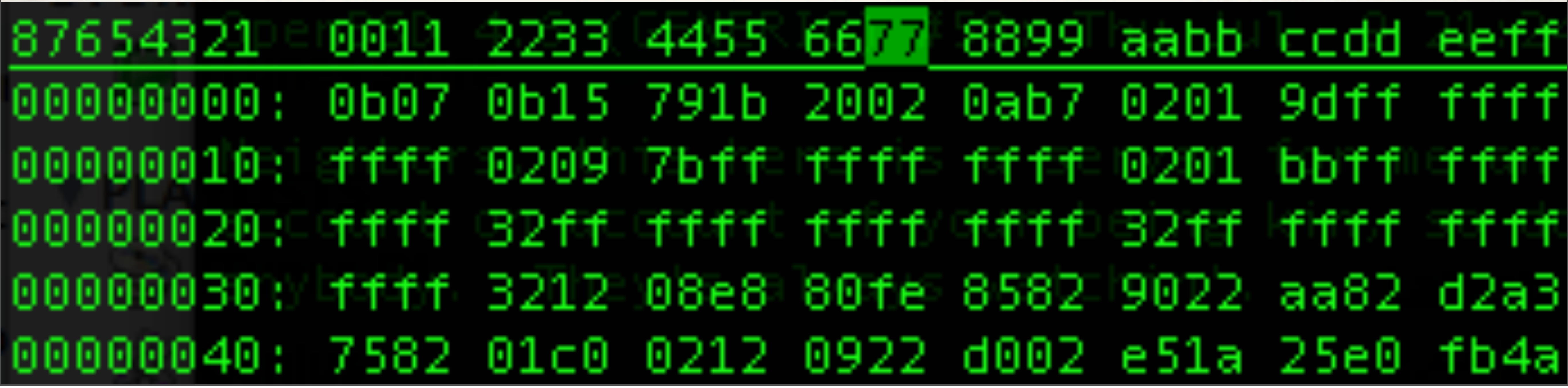
Quick and Dirty 25C32 Driver



```
class GoodFETSPI25C(GoodFETSPI):
    #opcodes
    WREN=0x06;
    WRDI=0x04;
    RDSR=0x05;
    WRSR=0x01;
    READ=0x03;
    WRITE=0x02;

    def peek8(self,adr):
        """Read a byte from the given address."""
        data=self.SPItrans([self.READ,(adr>>8)&0xFF,adr&0xFF,0x00]);
        return ord(data[3]);
```

EEPROM Basics

- Serial Number 15791B, bytes[3,4,5]
- Channel at byte[6].
- 8051 code begins at byte[7], loaded to CODE[0].

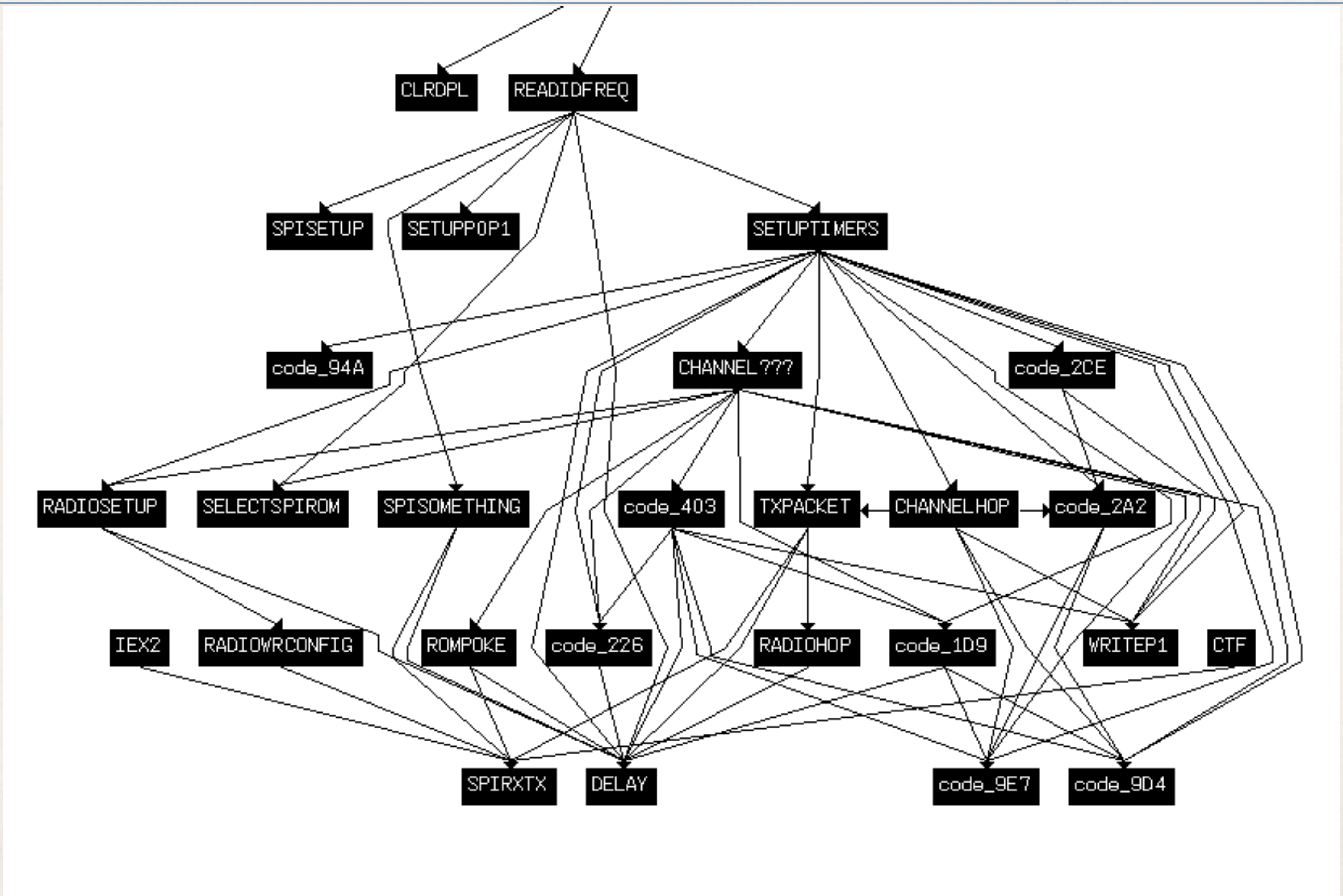


```
87654321  0011 2233 4455 6677 8899 aabb ccdd eeff
00000000: 0b07 0b15 791b 2002 0ab7 0201 9dff ffff
00000010: ffff 0209 7bff ffff ffff 0201 bbff ffff
00000020: ffff 32ff ffff ffff ffff 32ff ffff ffff
00000030: ffff 3212 08e8 80fe 8582 9022 aa82 d2a3
00000040: 7582 01c0 0212 0922 d002 e51a 25e0 fb4a
```

nRF24E1 Firmware in IDA

- ``goodfet.spi25c dump clicker.hex”
- Copy all but first 7 bytes to clicker.bin.
- Load clicker.bin to CODE memory at 0x0000.

Just 3kB of Code

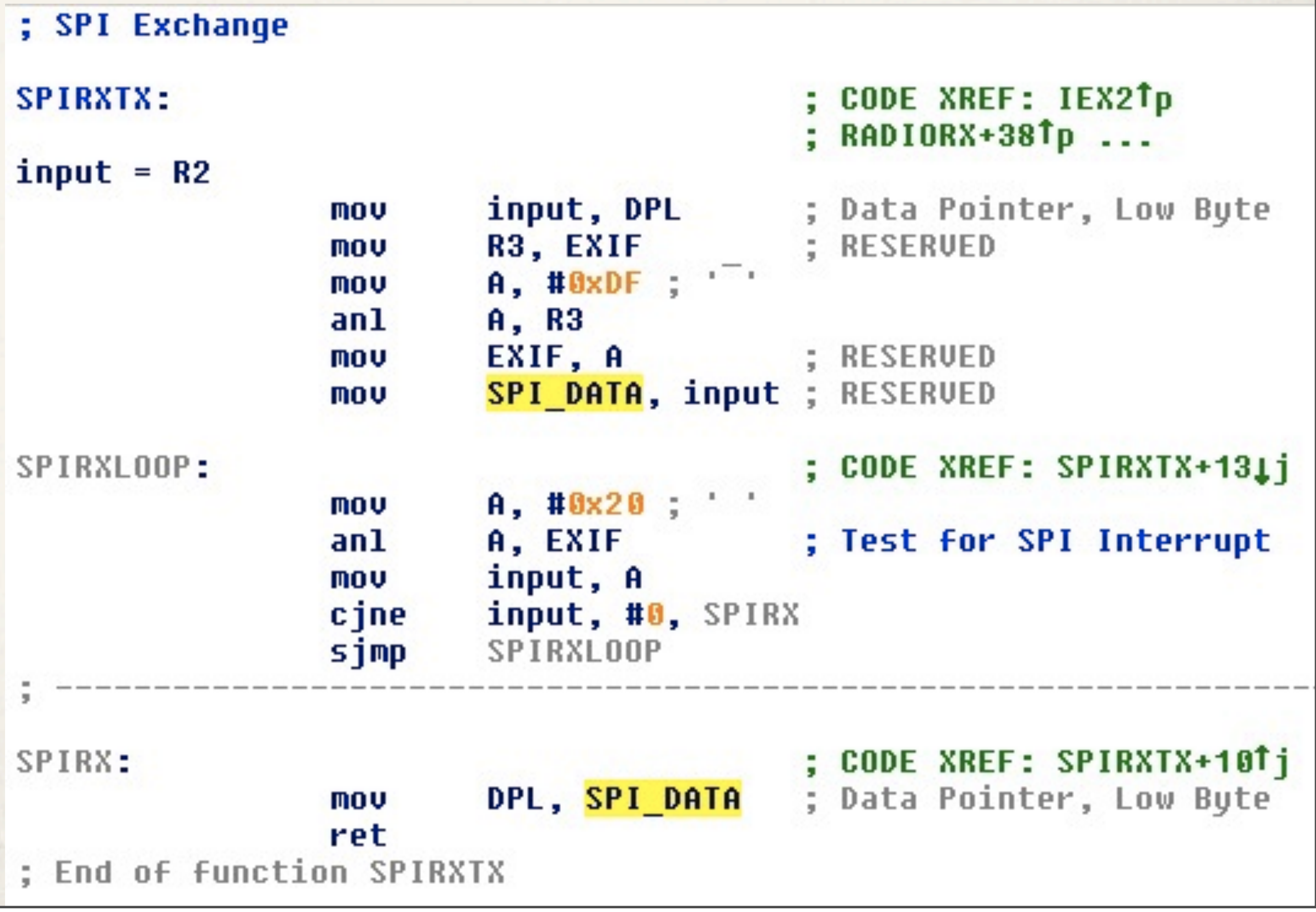


Identifying Ports, Functions

- No operating system.
- No function symbol names.
- I/O ports do have names.
    - These names are documented in the datasheet.
    - Can quickly be imported to IDA.

SPI Exchange Function

- mov SPI_DATA, input
- while(!READY);
- mov output, SPI_DATA



```
; SPI Exchange

SPIRXTX:                                ; CODE XREF: IEX2↑p
                                        ; RADIORX+38↑p ...
input = R2
                mov     input, DPL      ; Data Pointer, Low Byte
                mov     R3, EXIF        ; RESERVED
                mov     A, #0xDF ; '¯'
                anl     A, R3
                mov     EXIF, A         ; RESERVED
                mov     SPI_DATA, input ; RESERVED

SPIRXLOOP:                              ; CODE XREF: SPIRXTX+13↓j
                mov     A, #0x20 ; ' '
                anl     A, EXIF         ; Test for SPI Interrupt
                mov     input, A
                cjne    input, #0, SPIRX
                sjmp    SPIRXLOOP
; ---------------------------------------------------------------------------

SPIRX:                                  ; CODE XREF: SPIRXTX+10↑j
                mov     DPL, SPI_DATA   ; Data Pointer, Low Byte
                ret
; End of function SPIRXTX
```

nRF24E1 Internal Arrangement

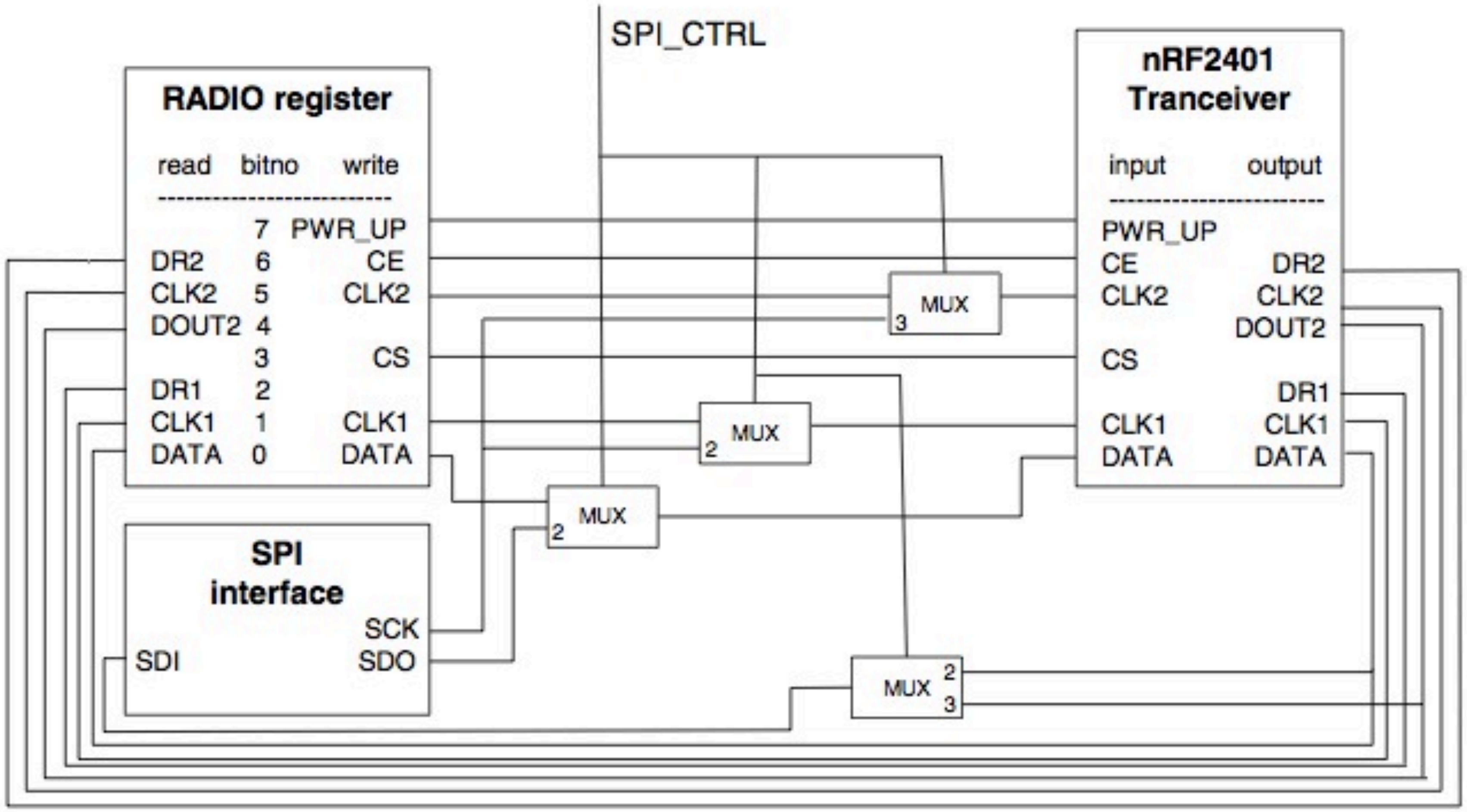


Figure 4-1 : Transceiver interface

- 8051 MCU
- Internal SPI Bus
- RADIO register #0x80

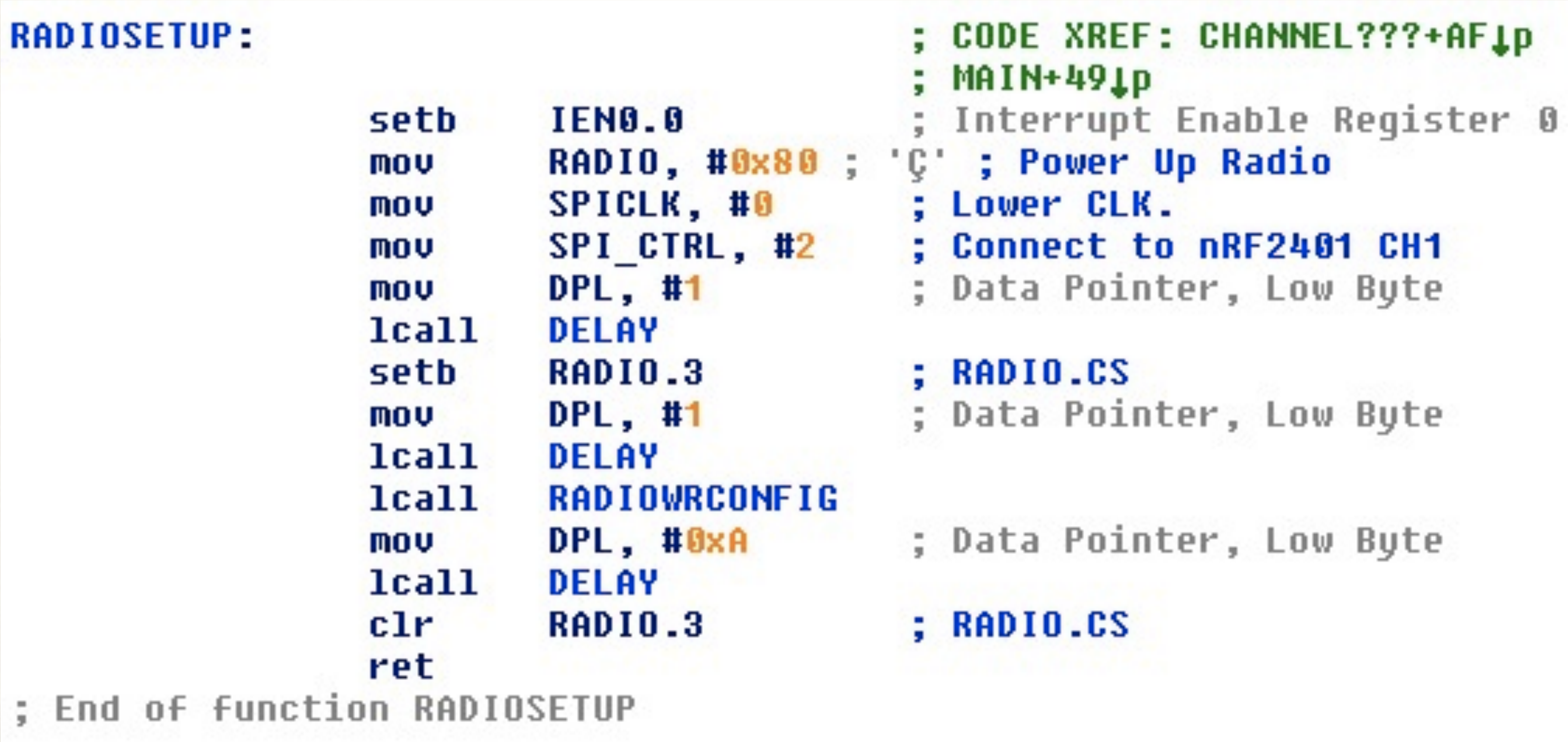
Useful Registers

- SPI_DATA, SPICLK, SPI_CNTRL, EXIF
- P1 LED Port
- P0.0 SPI EEPROM Slave Select
- RADIO #0x80
    - RADIO.3 is Radio Slave Select
    - RADIO.7 is Power Up

- Radio SPI
  - SETB RADIO.3
  - for(...) SPIRXTX(...)
  - CLRB RADIO.3

- EEPROM SPI
  - CLRB P0.0
  - for(...) SPIRXTX(...)
  - SETB P0.0

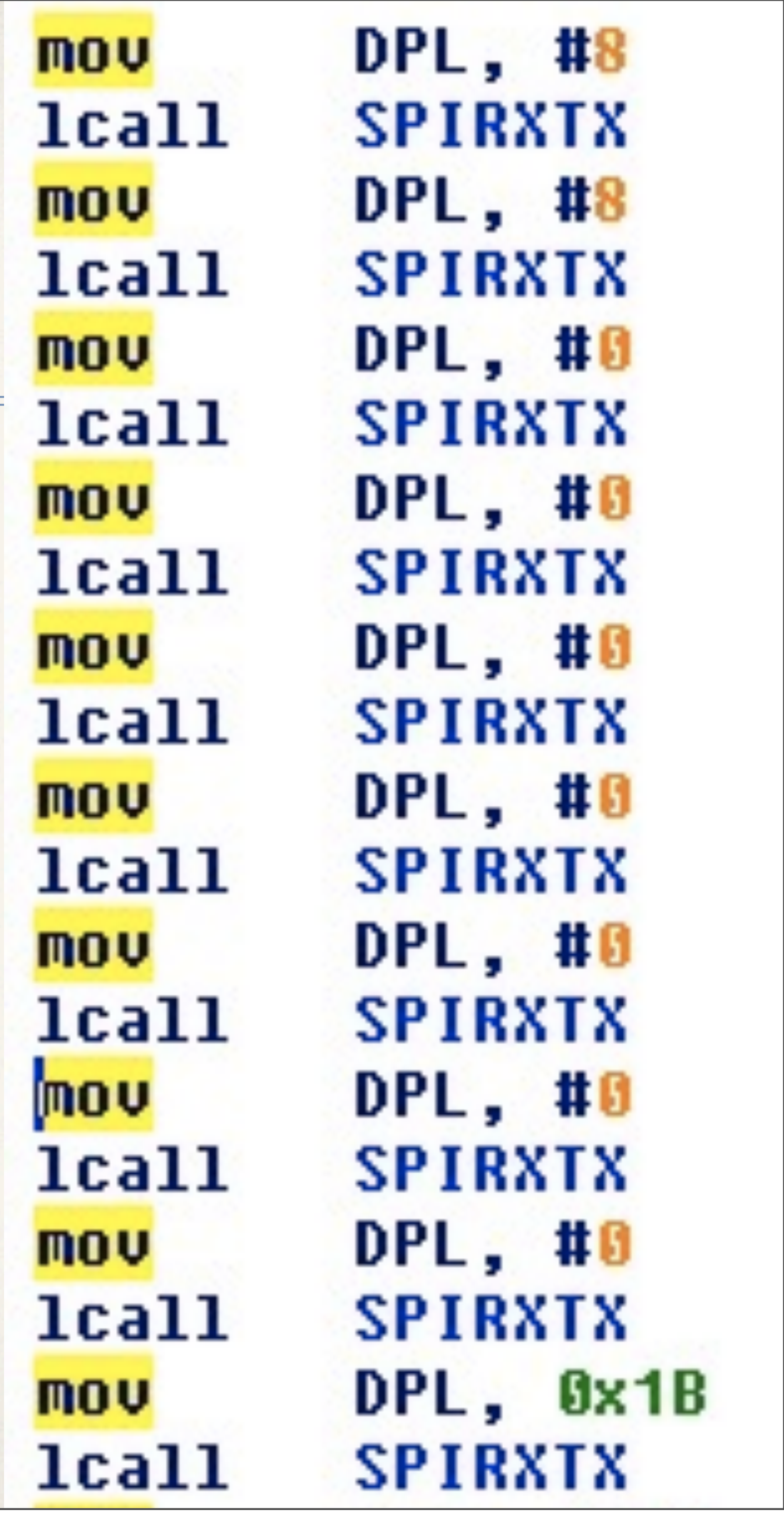
From Registers to Functions



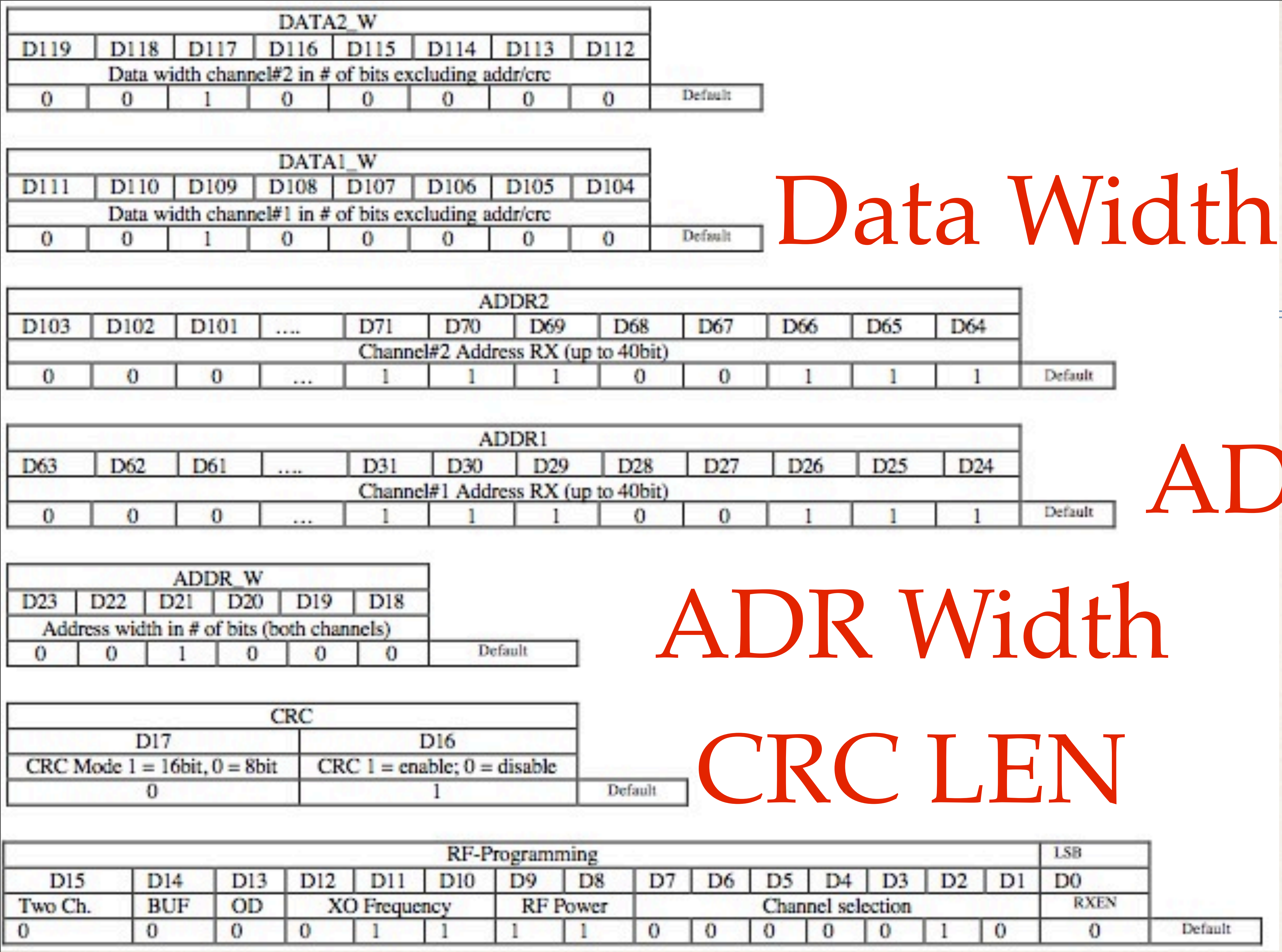
```
RADIOSETUP:                                ; CODE XREF: CHANNEL???+AF↓p
                                           ; MAIN+49↓p
                setb    IEN0.0             ; Interrupt Enable Register 0
                mov     RADIO, #0x80 ; 'Ç'  ; Power Up Radio
                mov     SPICLK, #0         ; Lower CLK.
                mov     SPI_CTRL, #2       ; Connect to nRF2401 CH1
                mov     DPL, #1            ; Data Pointer, Low Byte
                lcall   DELAY
                setb    RADIO.3            ; RADIO.CS
                mov     DPL, #1            ; Data Pointer, Low Byte
                lcall   DELAY
                lcall   RADIOWRCONFIG
                mov     DPL, #0xA          ; Data Pointer, Low Byte
                lcall   DELAY
                clr     RADIO.3            ; RADIO.CS
                ret
; End of function RADIOSETUP
```

RADIOWRCONFIG

- Just a lot of SPIRXTX.
  - 08 08 00 00 00 00 00 00 00
  - (1B) (1C) (1D)
  - 63 6F
  - (1A)+1



```
mov     DPL, #8
lcall   SPIRXTX
mov     DPL, #8
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, #0
lcall   SPIRXTX
mov     DPL, 0x1B
lcall   SPIRXTX
```



DATA2_W								
D119	D118	D117	D116	D115	D114	D113	D112	
Data width channel#2 in # of bits excluding addr/crc								
0	0	1	0	0	0	0	0	Default

DATA1_W								
D111	D110	D109	D108	D107	D106	D105	D104	
Data width channel#1 in # of bits excluding addr/crc								
0	0	1	0	0	0	0	0	Default

Data Width

ADDR2												
D103	D102	D101	….	D71	D70	D69	D68	D67	D66	D65	D64	
Channel#2 Address RX (up to 40bit)												
0	0	0	…	1	1	1	0	0	1	1	1	Default

ADDR1												
D63	D62	D61	….	D31	D30	D29	D28	D27	D26	D25	D24	
Channel#1 Address RX (up to 40bit)												
0	0	0	…	1	1	1	0	0	1	1	1	Default

ADR

ADDR_W						
D23	D22	D21	D20	D19	D18	
Address width in # of bits (both channels)						
0	0	1	0	0	0	Default

ADR Width

CRC		
D17	D16	
CRC Mode 1 = 16bit, 0 = 8bit	CRC 1 = enable; 0 = disable	
0	1	Default

CRC LEN

RF-Programming															LSB	
D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	
Two Ch.	BUF	OD	XO Frequency			RF Power		Channel selection							RXEN	
0	0	0	0	1	1	1	1	0	0	0	0	0	1	0	0	Default

Config

Channel

RADIOWRCONFIG

- Just a lot of SPIRXTX.
  - 08 08 00 00 00 00 00 00 00
  - (1B) (1C) (1D)
  - 63 6F
  - (1A)+1

- Channel at 0x1A
- MAC at 0x1B, 0x1C, 0x1D
- 4 bytes of data
- 1 byte checksum

Transmission

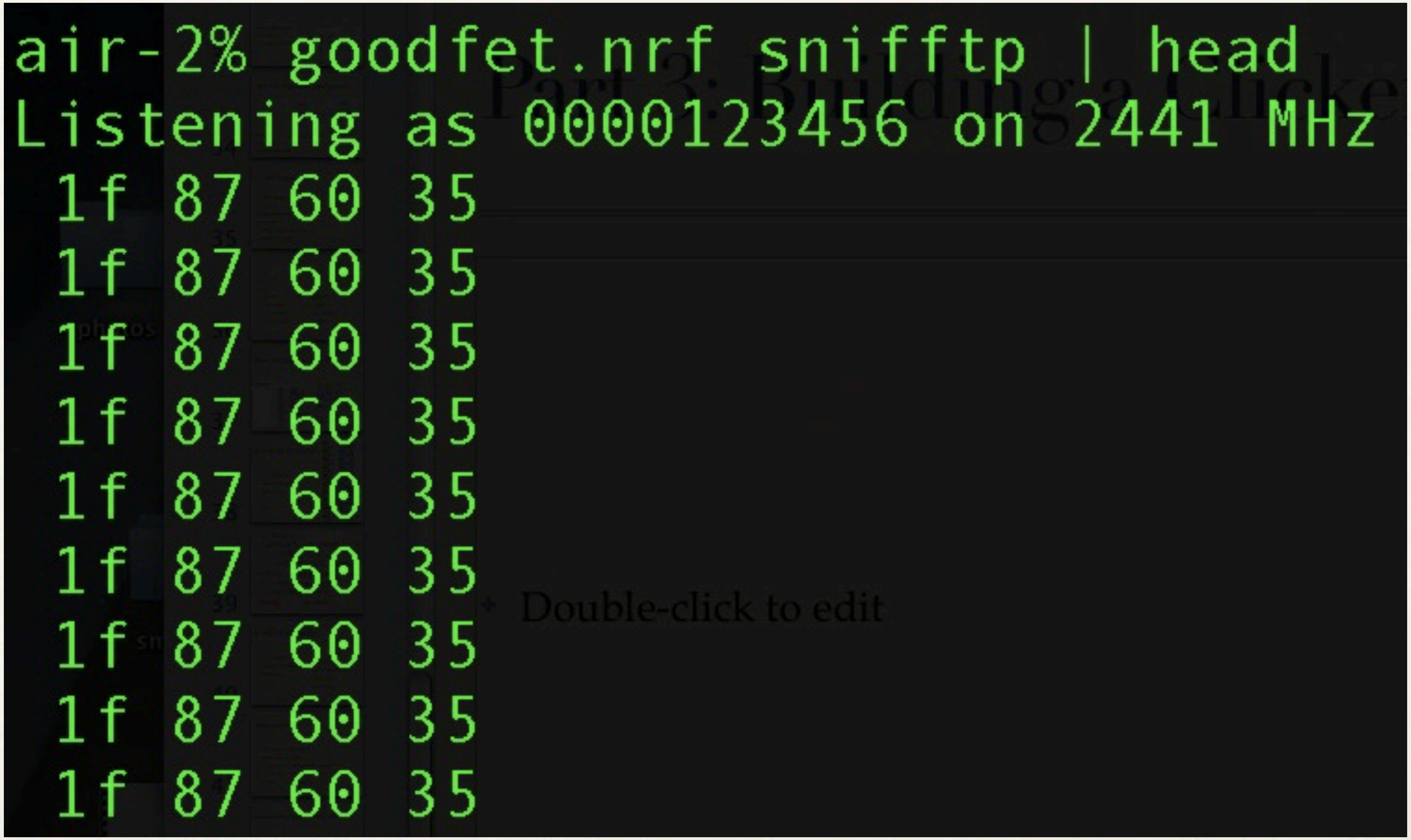
- Function takes one byte of input.
- Repeated calls to SPITXRX
  - (1E) (1F) (20)   //Destination MAC Address
  - (1B) (1C) (1D)   //Source MAC Address
  - (input)   //Button Code

Destination MAC at 1E, 1F, 20

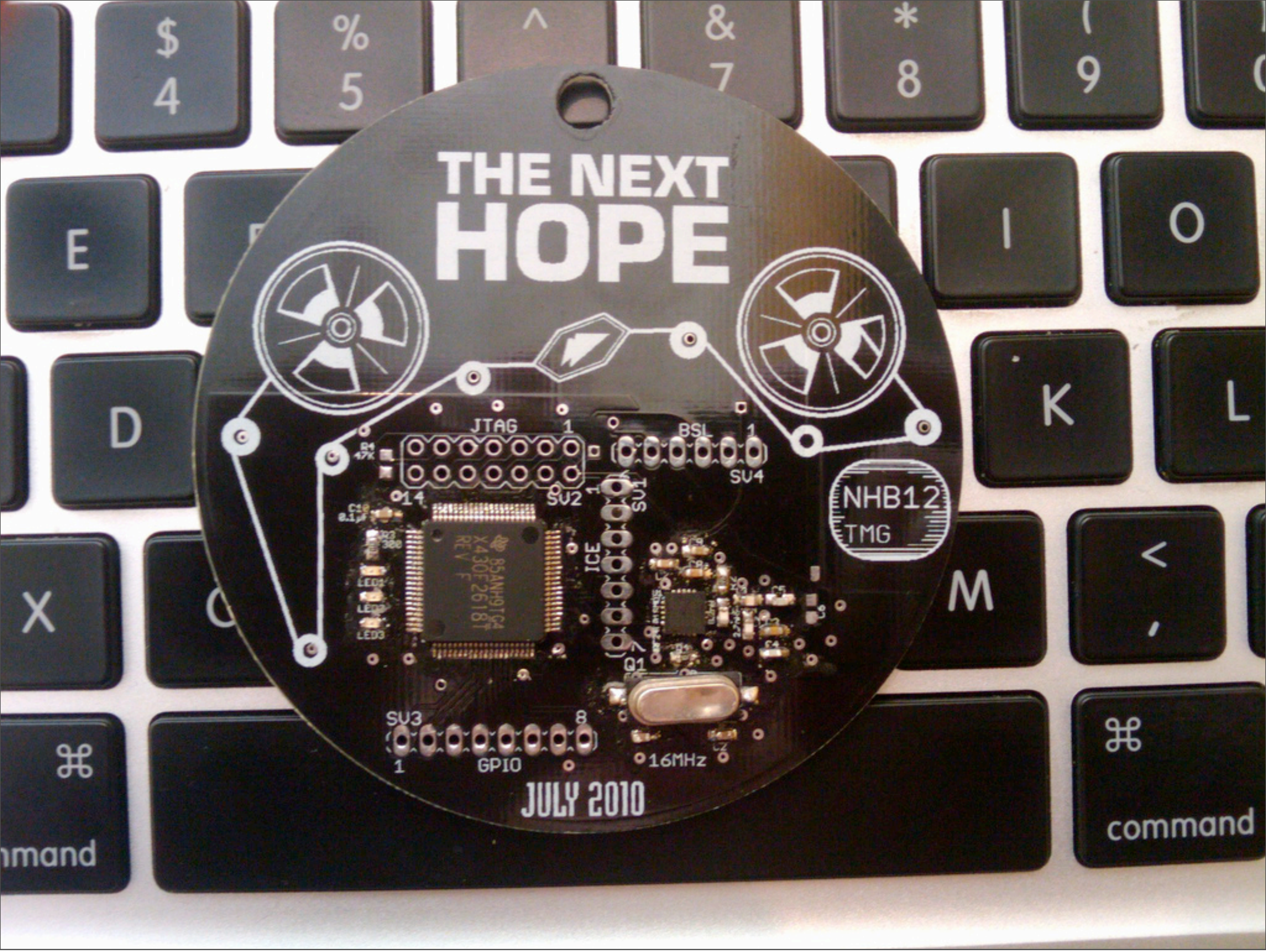
- MOV 0x1E, #0x12
- MOV 0x1F, #0x34
- MOV 0x20, #0x56

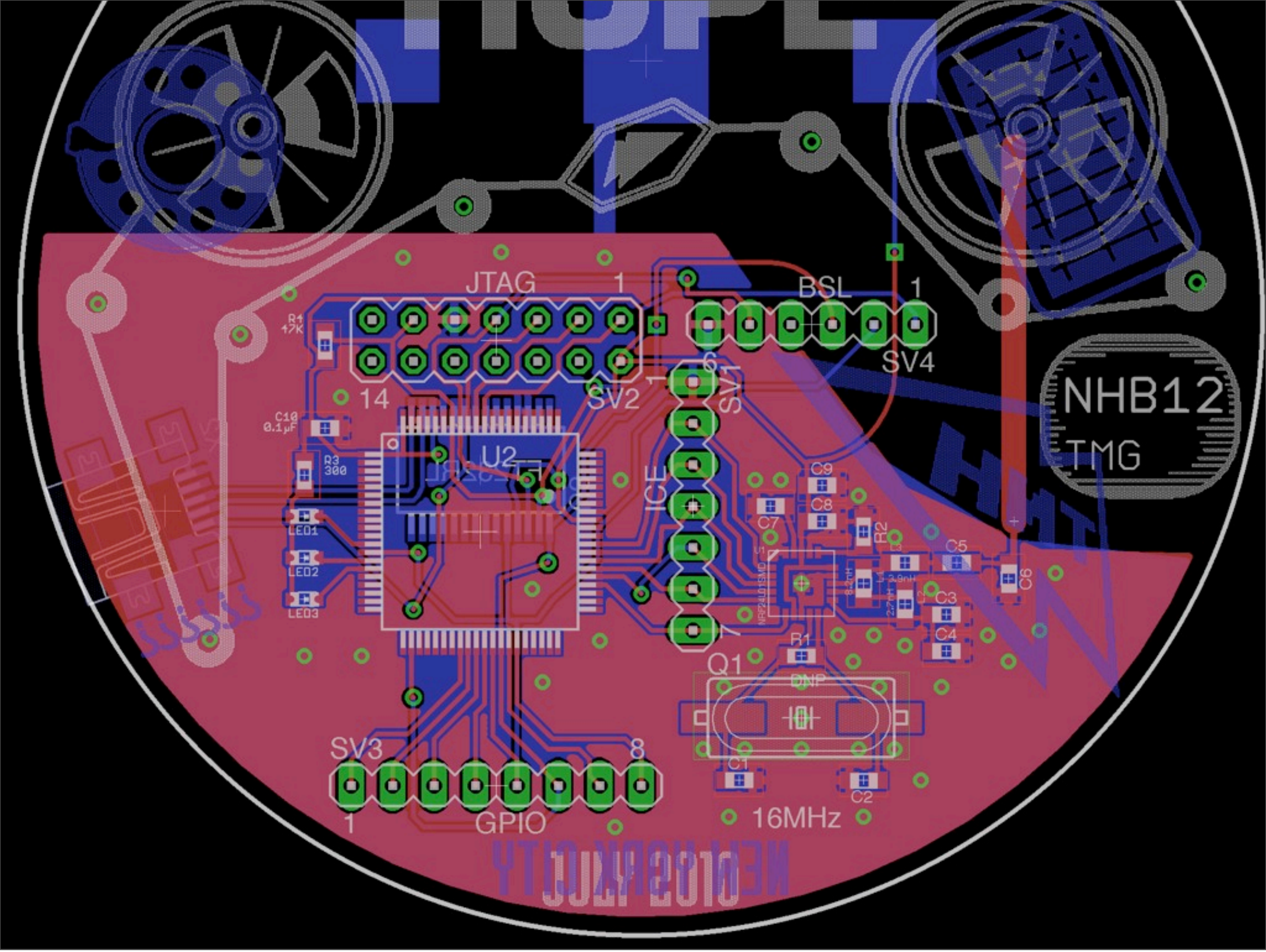
- DMAC is 0x123456
- Payload length is 4 bytes.
- One byte checksum.

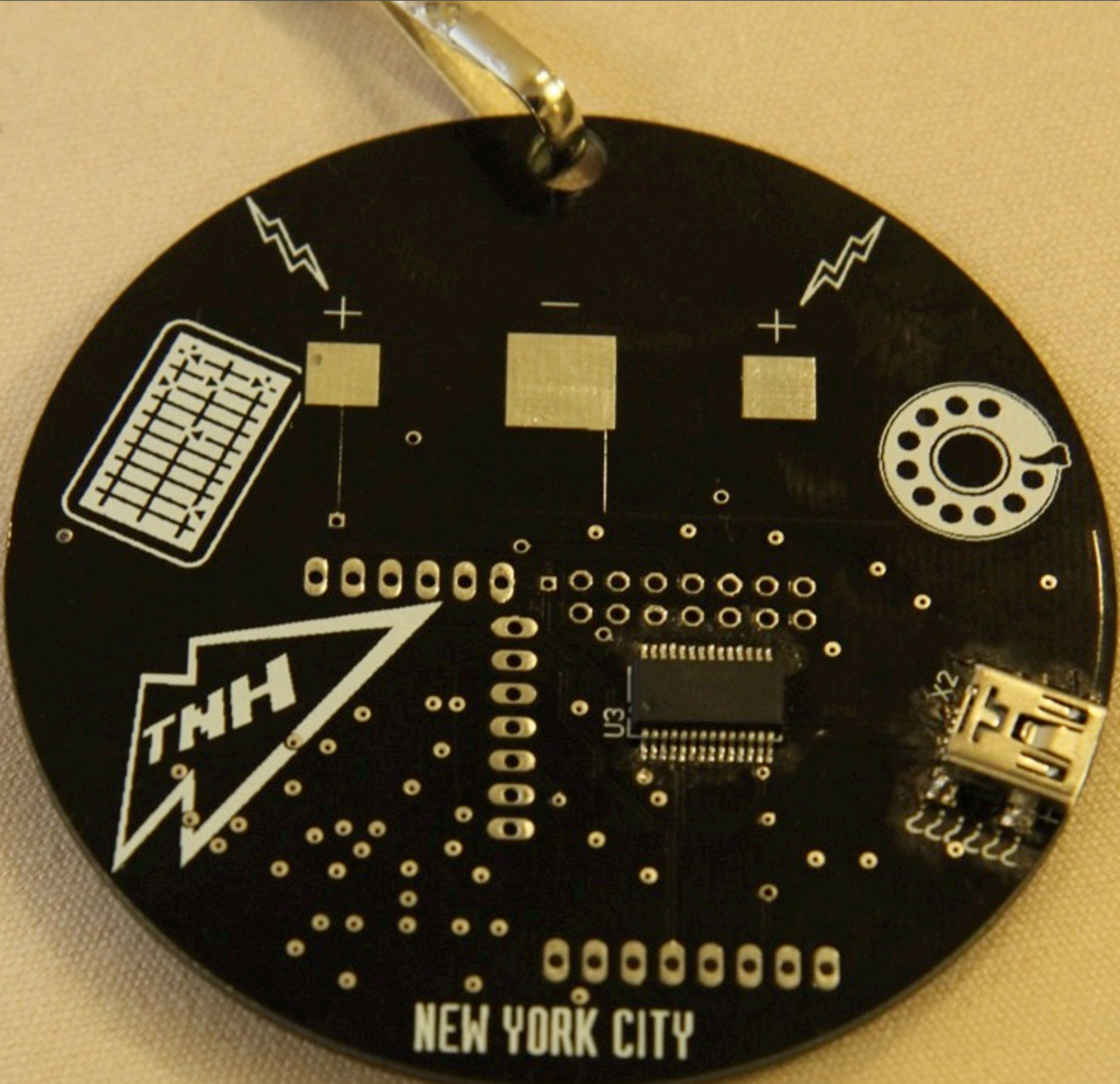
Part 3: Building a Clicker Sniffer



```
air-2% goodfet.nrf snifftp | head
Listening as 0000123456 on 2441 MHz
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
 1f 87 60 35
```


THE NEXT
HOPE
JTAG
1
BSL
1
SV4
14
SV2
SV1
1
ICE
7
NHB12
TMG
85ANH9IG4
X430F2618T
REV F
LED1
LED2
LED3
Q1
SV3
8
1
GPIO
16MHz
JULY 2010


JTAG
1
14
SV2
BSL
1
SV4
6
SV1
1
7
ICE
R1
47K
C10
0.1µF
R3
300
LED1
LED2
LED3
U2
NHB12
TMG
C9
C8
C7
R2
C5
C6
C3
C4
R1
Q1
DNP
C1
C2
16MHz
SV3
8
1
GPIO


+
−
+
TNH
U3
X2
NEW YORK CITY

Next Hope Badge Hardware

- Texas Instruments MSP430 Microcontroller
    - 16-bit RISC, GNU toolchain.
- Nordic nRF24L01+ Radio
    - Radio chain from reference design.
- Runs either OpenBeacon or GoodFET Firmware

NHBadge+GoodFET

- GoodFET firmware exposes radio by USB.
- GoodFET client provides Python libraries for nRF24L01+ Radio.

Radio Settings

- 2.441 GHz
- 1Mbps GFSK
- MAC 0x123456
- 4 byte payload, CRC16

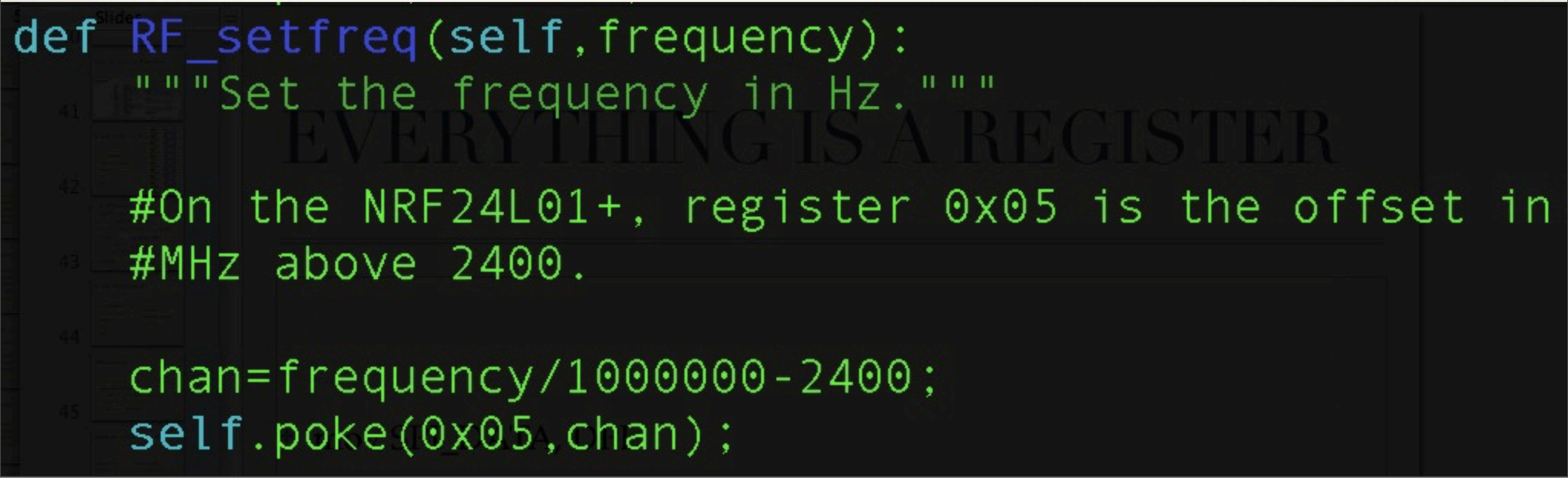
- 2.481 GHz
- 2Mbps GFSK
- MAC 0x0102030201
- 16 byte payload, CRC8

GoodFET Python Client

- Separate class for most protocols.
- Some classes share a hardware module.
  - SPI EEPROM needs no additional C code

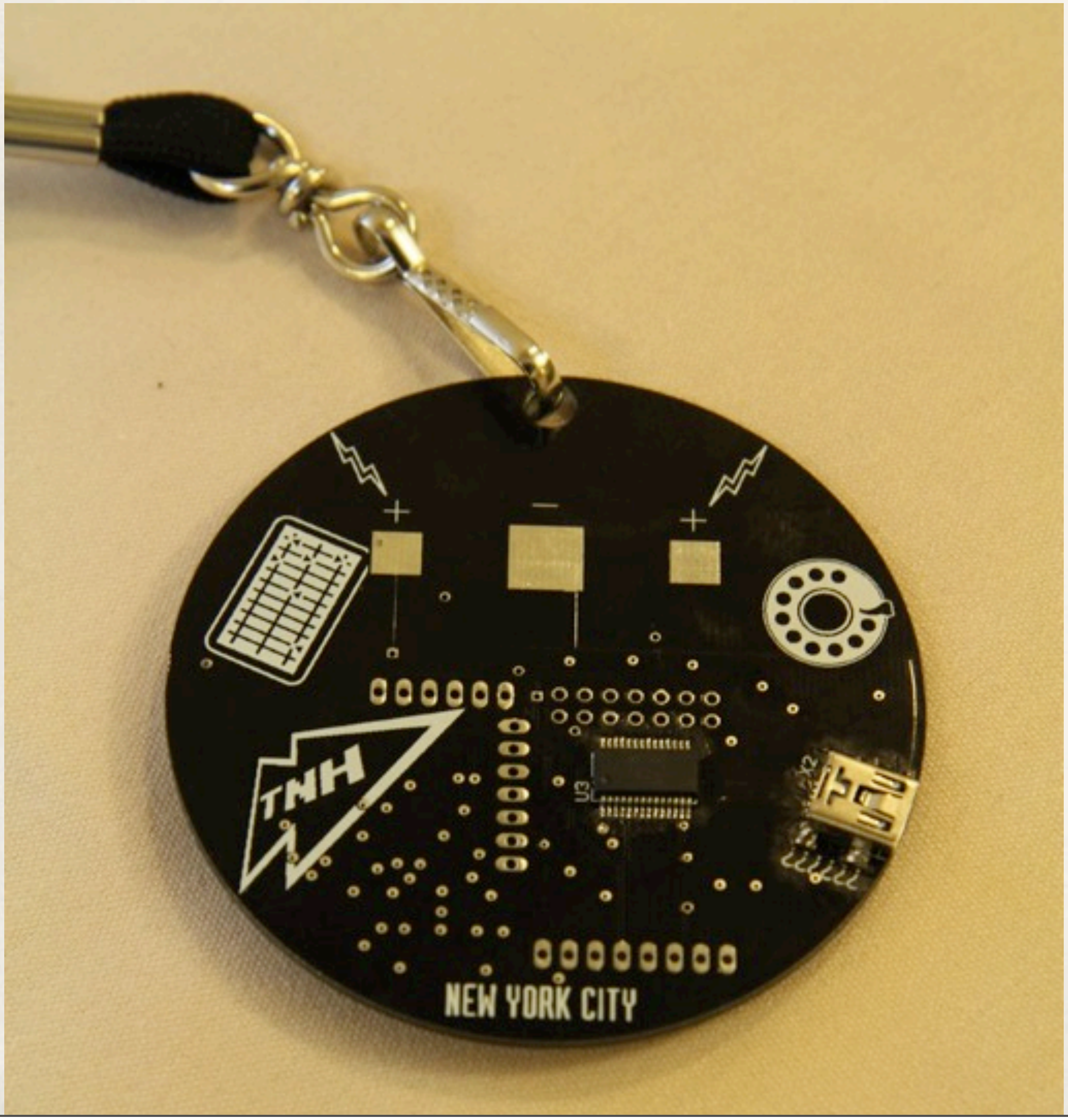
EVERYTHING IS A REGISTER

- mov SPI_DATA, DPL
- mov DPL, SPI_DATA

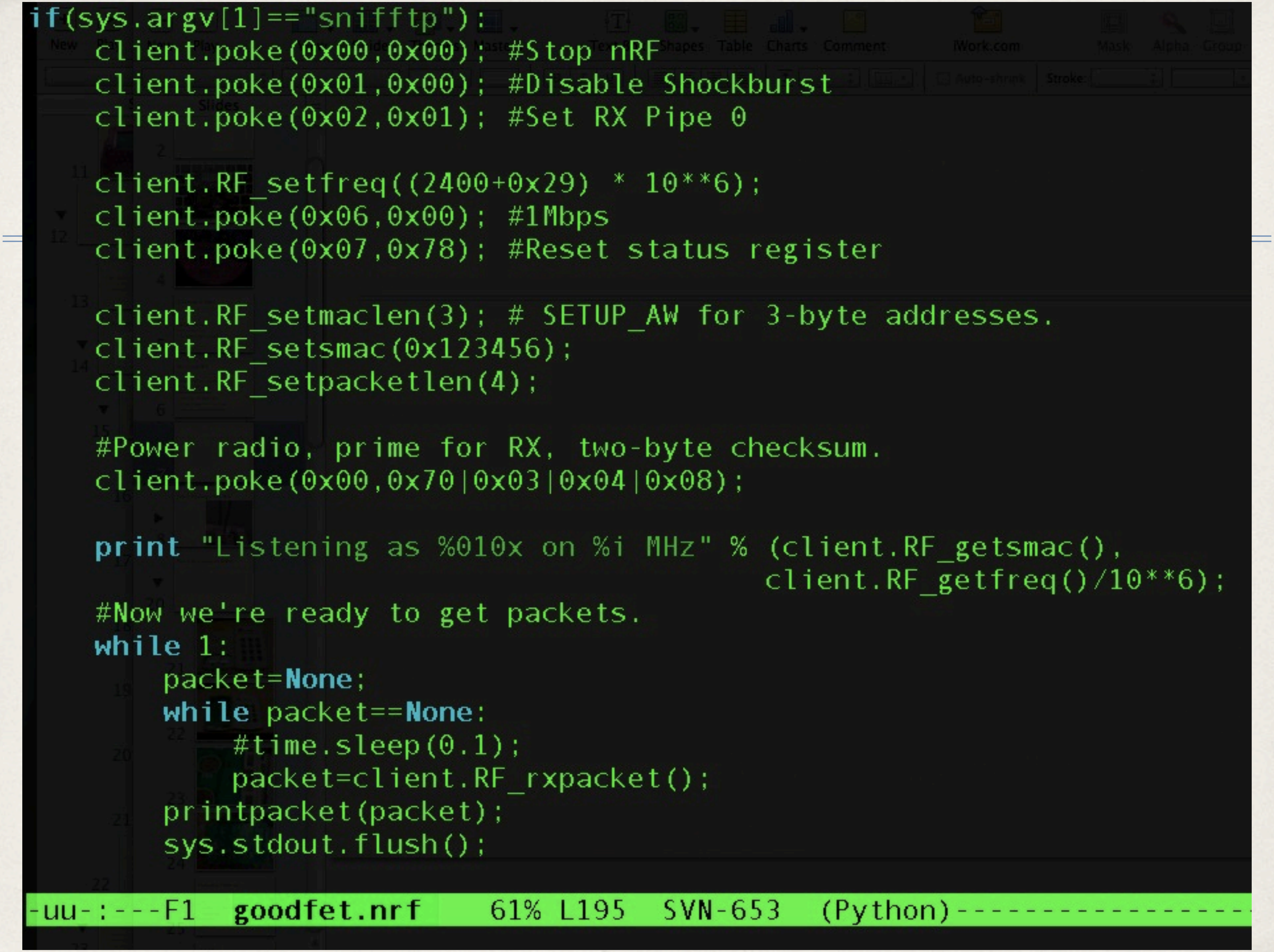


```
def RF_setfreq(self,frequency):
    """Set the frequency in Hz."""

    #On the NRF24L01+, register 0x05 is the offset in
    #MHz above 2400.

    chan=frequency/1000000-2400;
    self.poke(0x05,chan);
```

Client Driver



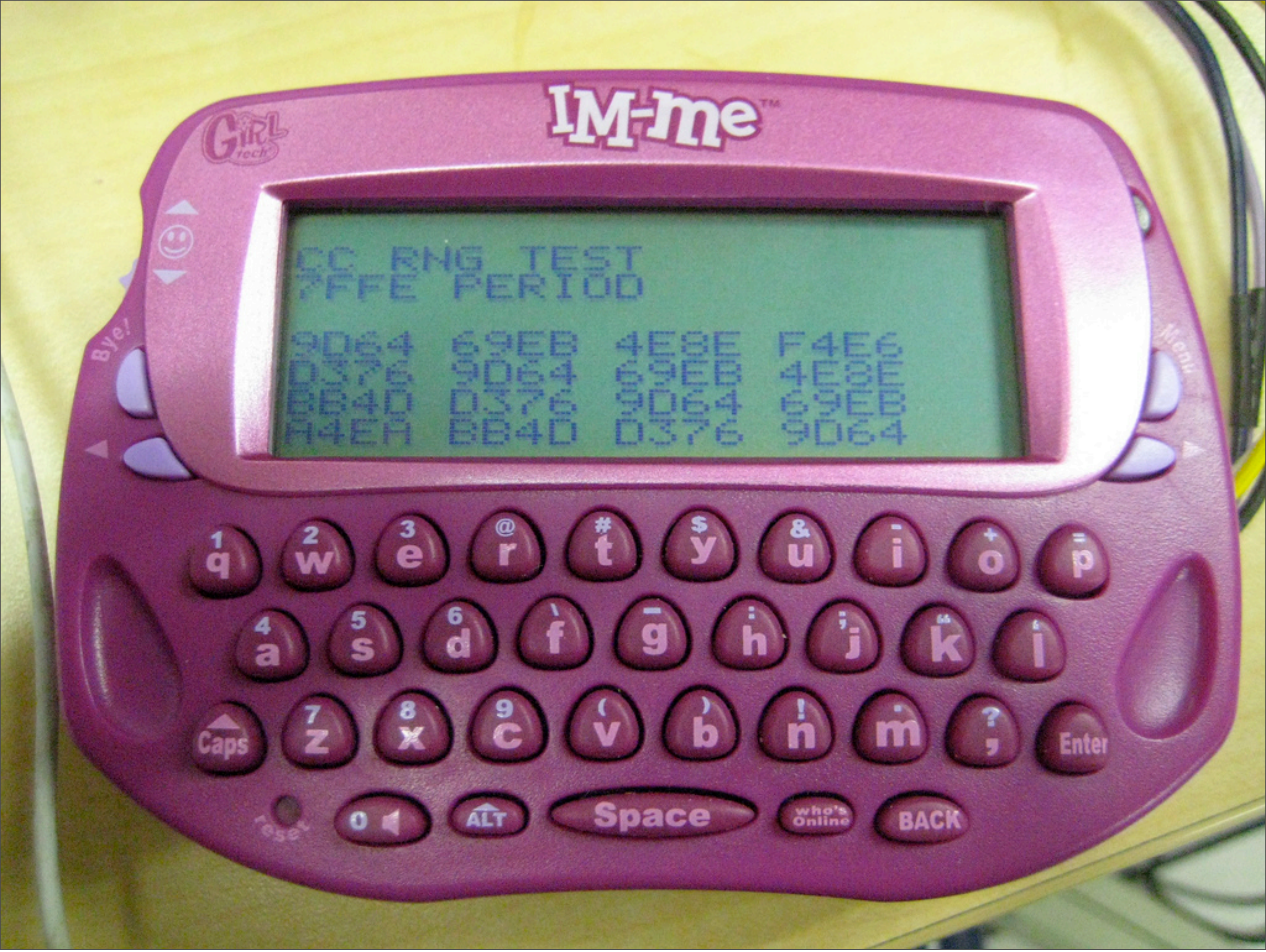
- GoodFETNRF
  - poke(register,value);
  - RF_setfreq(Hz)
  - RF_setsmac(mac)
  - RF_setpacketlen(len)



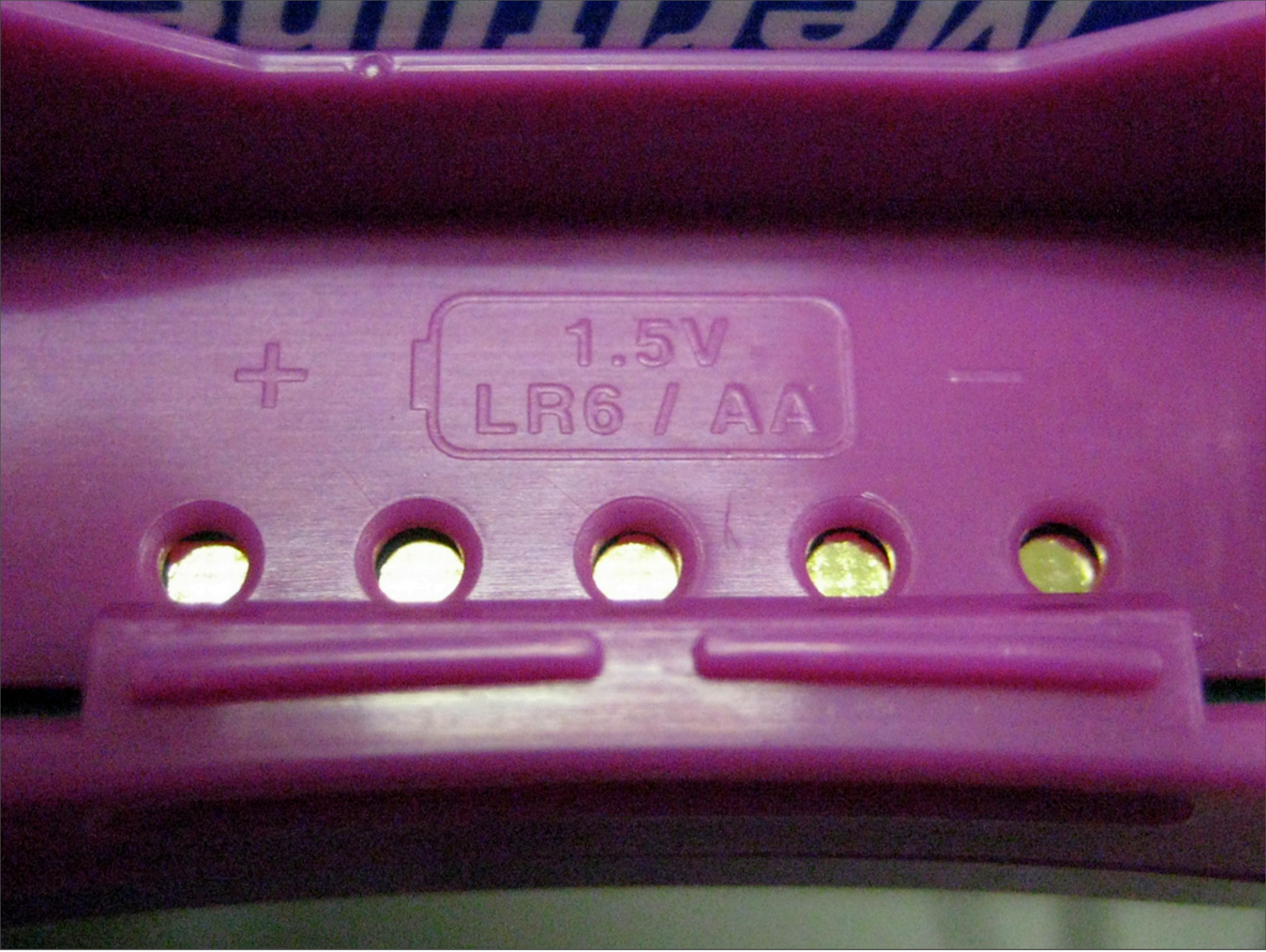
```
if(sys.argv[1]=="snifftp"):
   client.poke(0x00,0x00); #Stop nRF
   client.poke(0x01,0x00); #Disable Shockburst
   client.poke(0x02,0x01); #Set RX Pipe 0

   client.RF_setfreq((2400+0x29) * 10**6);
   client.poke(0x06,0x00); #1Mbps
   client.poke(0x07,0x78); #Reset status register

   client.RF_setmaclen(3); # SETUP_AW for 3-byte addresses.
   client.RF_setsmac(0x123456);
   client.RF_setpacketlen(4);

   #Power radio, prime for RX, two-byte checksum.
   client.poke(0x00,0x70|0x03|0x04|0x08);

   print "Listening as %010x on %i MHz" % (client.RF_getsmac(),
                                            client.RF_getfreq()/10**6);
   #Now we're ready to get packets.
   while 1:
       packet=None;
       while packet==None:
           #time.sleep(0.1);
           packet=client.RF_rxpacket();
       printpacket(packet);
       sys.stdout.flush();

-uu-:---F1  goodfet.nrf      61% L195   SVN-653   (Python)------------------
```

Other Targets

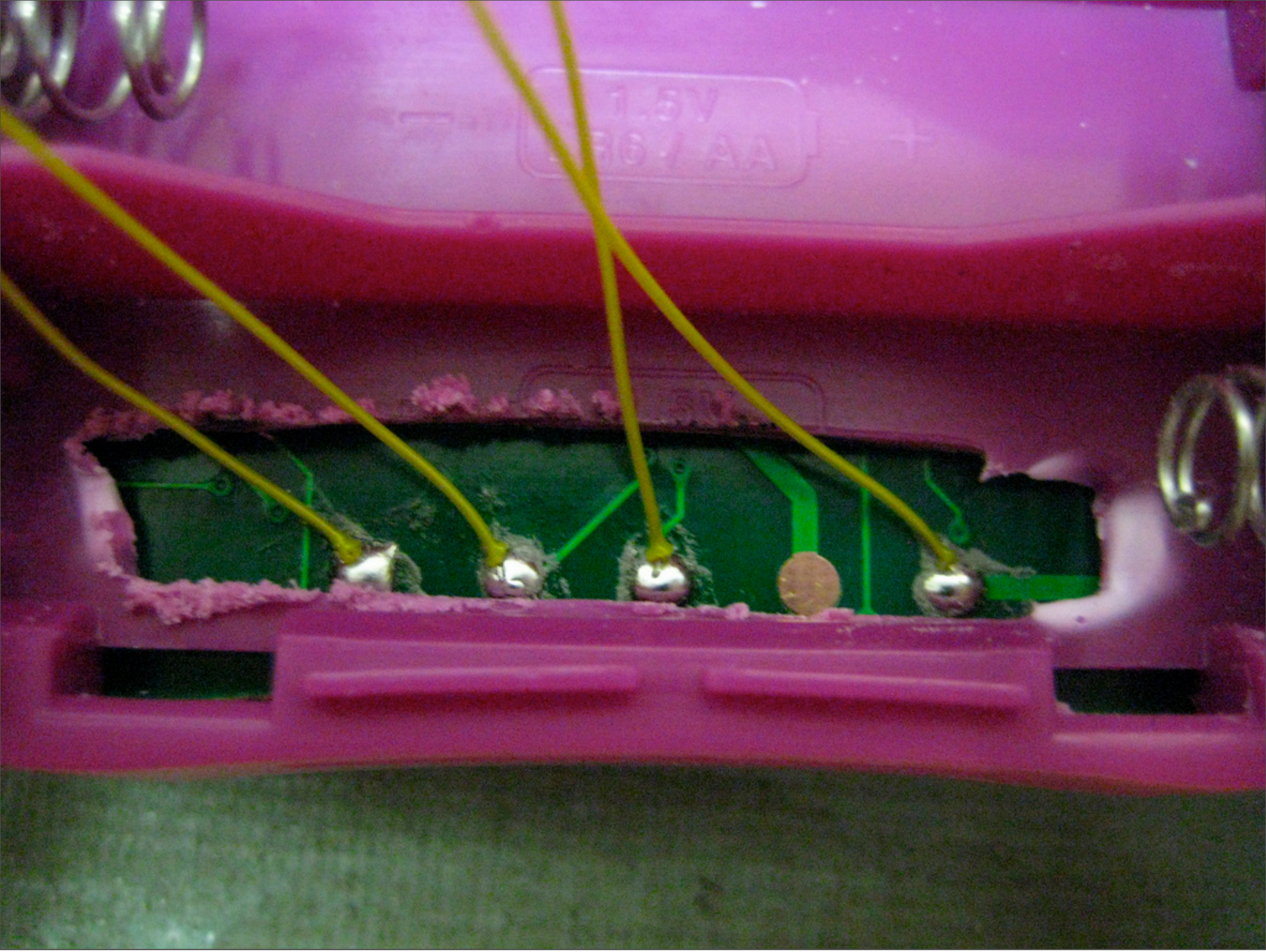
- Toys
- Smart Grid
- Sports
- Medical

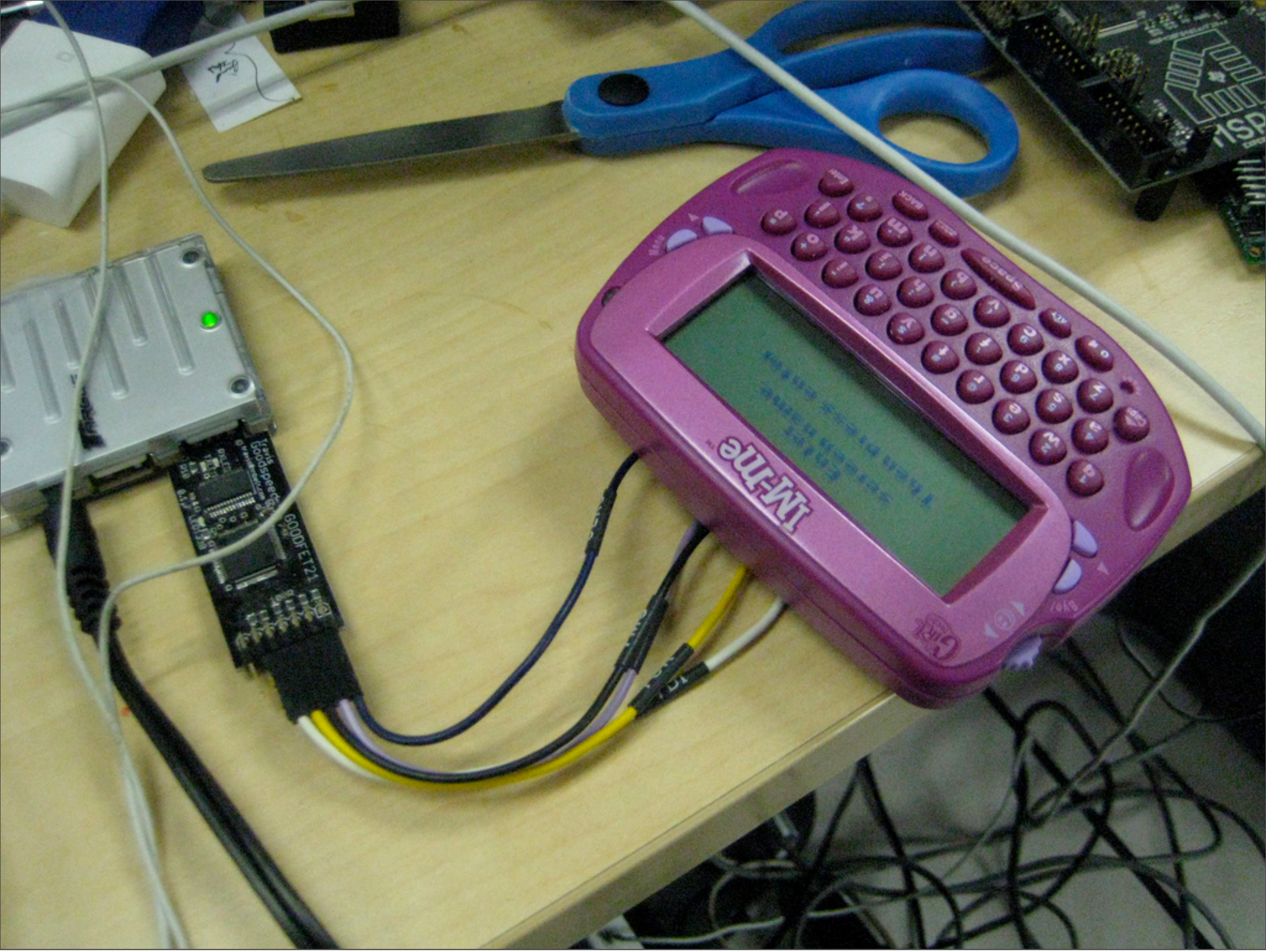


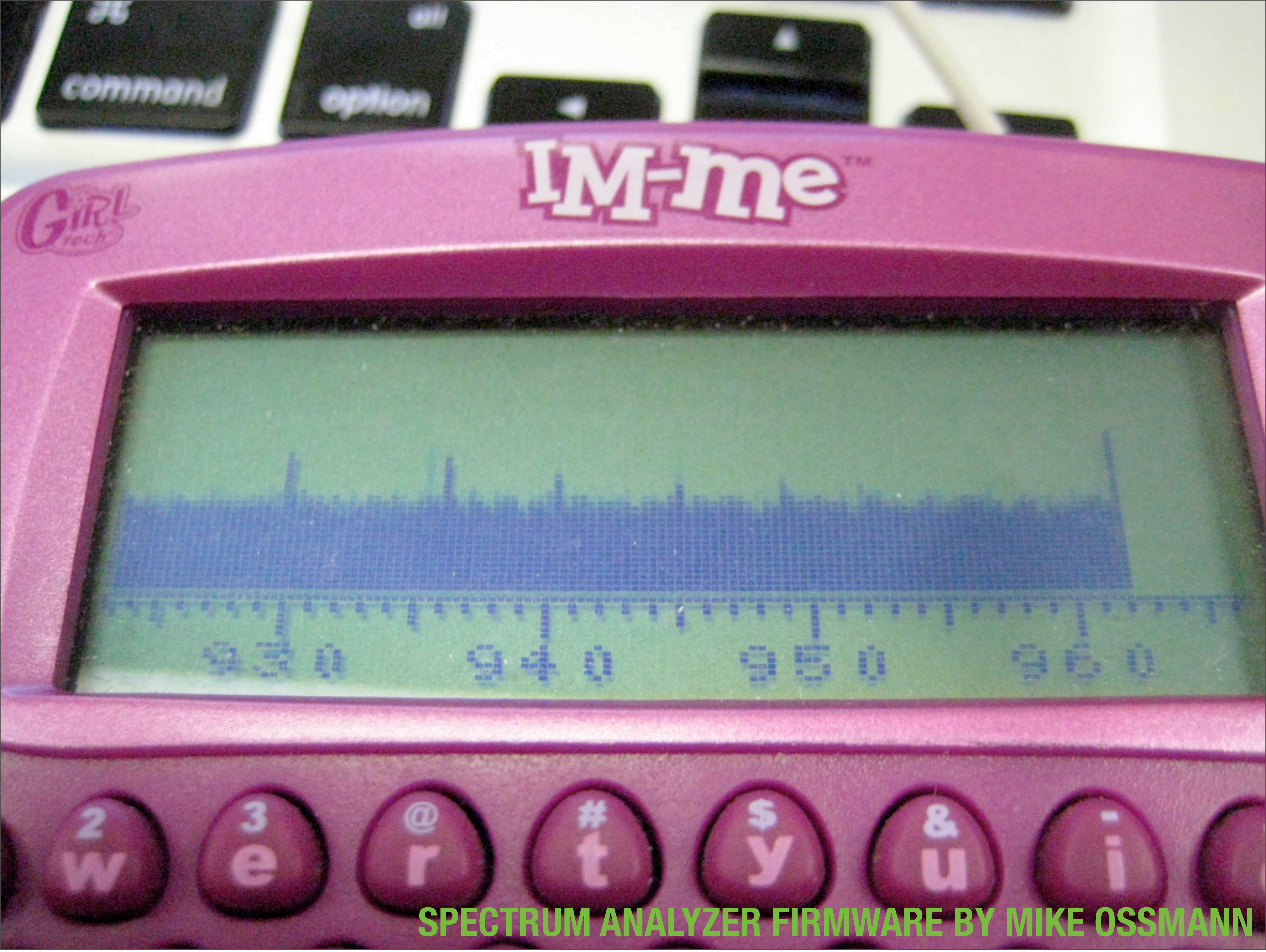
```
CC RNG TEST
7FFE PERIOD

9D64 69EB 4E8E F4E6
D376 9D64 69EB 4E8E
BB4D D376 9D64 69EB
A4EA BB4D D376 9D64
```



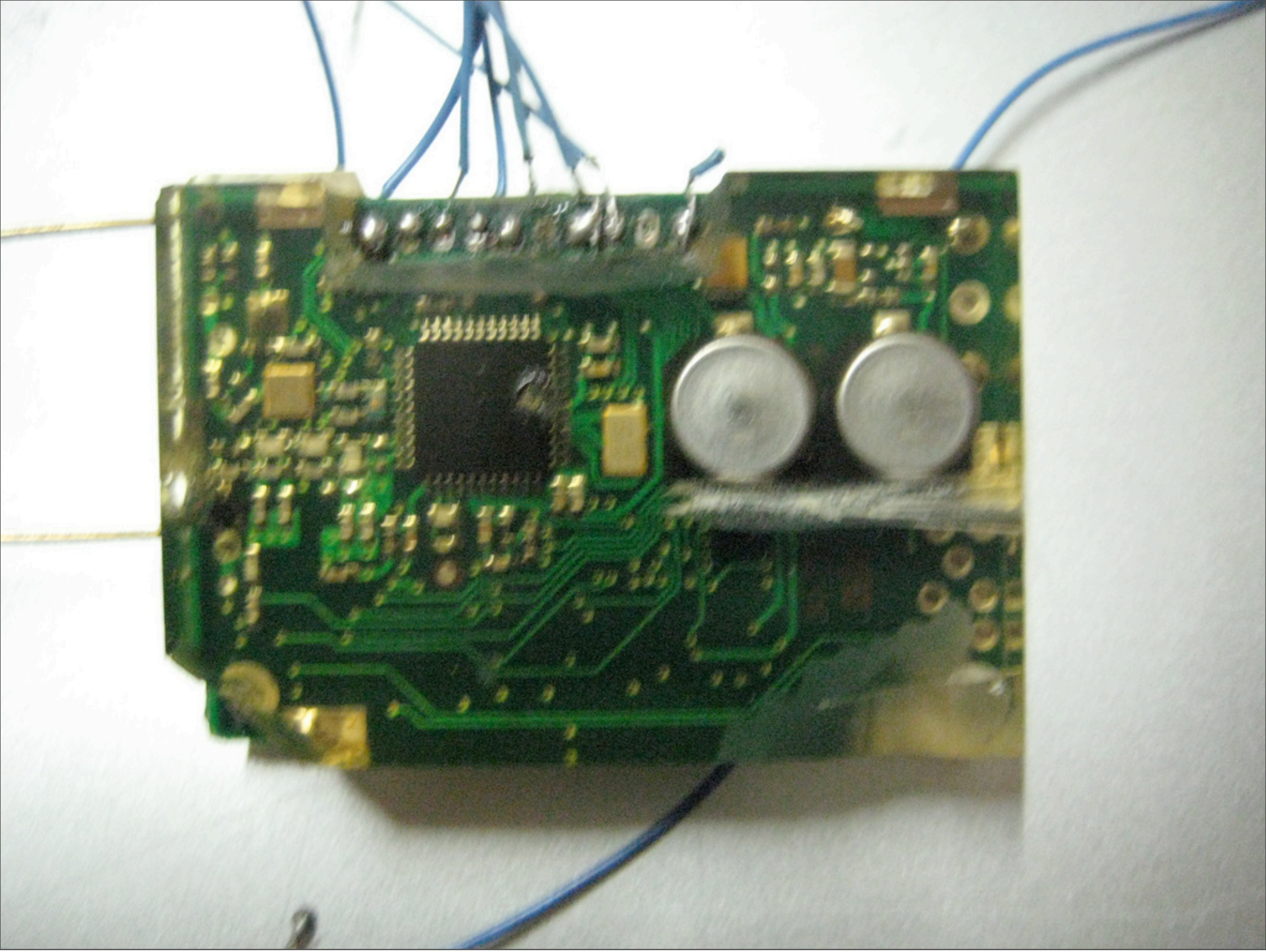
+
1.5V
LR6 / AA
—


1.5V
R6 / AA


IM-me
Enter
screen name
Then press enter
GOODFET21
Travis Goodspeed



SPECTRUM ANALYZER FIRMWARE BY MIKE OSSMANN



ANT Protocol

- Proprietary LPAN protocol.
- Compatible with NHBadge.
- Not yet reversed.
    - (Hardware is waiting at my apartment. :)

Neat Tricks

- Vulnerabilities are chip-wise, not application-wise.
  - Every EM2xx chip exposes full memory to an external debugger.
  - Every Chipcon 8051 chip exposes RAM to a debugger, but not Flash.
- Most ZigBee SEP devices have bad random number generators.
  - ECMQV exposes private keys when the nonce is recoverable!

Memory Exposure

- Access controls exist for protecting CODE, not DATA.
- Reprogramming is almost always allowed.
- Erase, then dump. RAM and keys will be intact.
  - goodfet.cc erase
  - goodfet.cc dumpdata ram.hex 0 0xFFFF

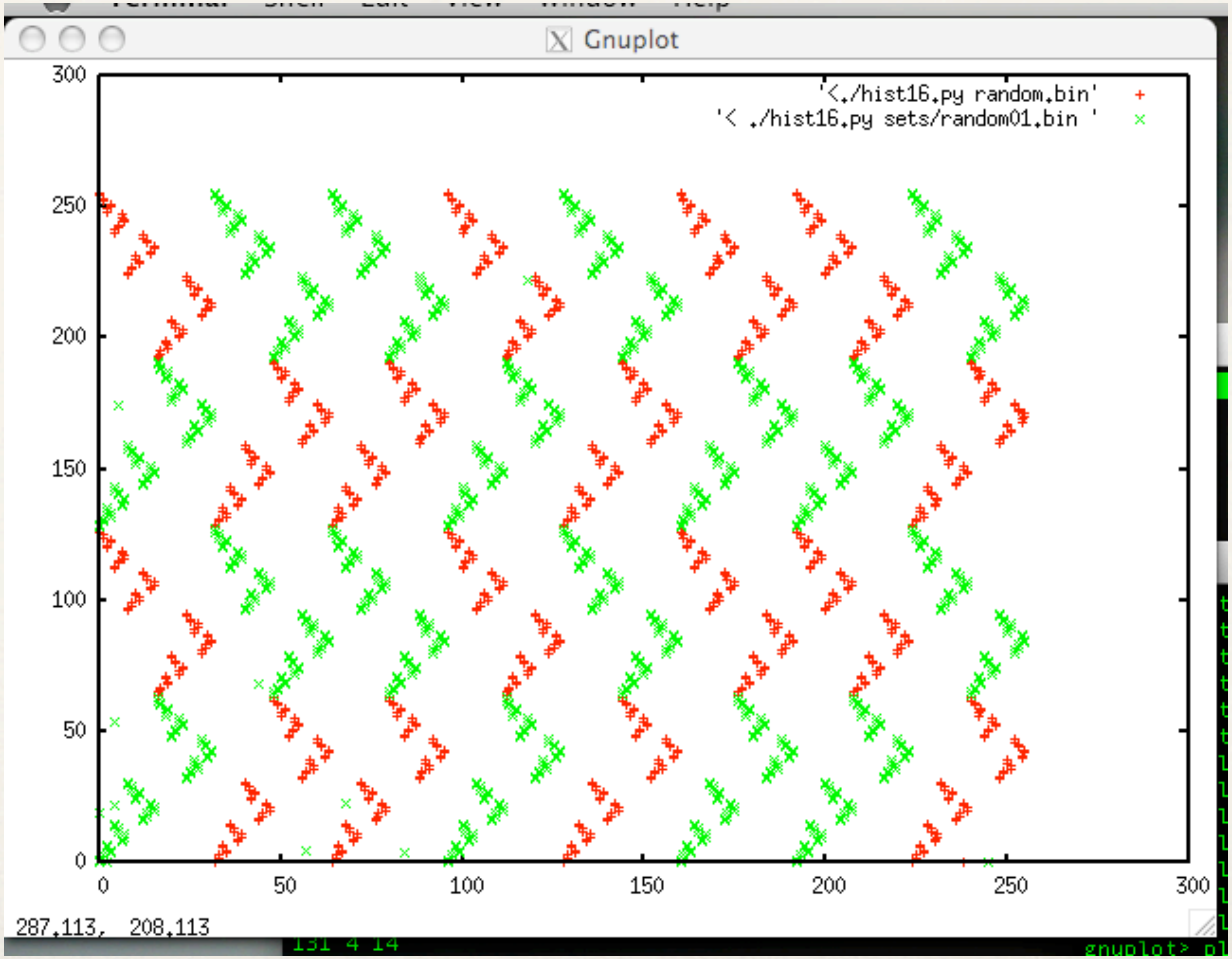
Stack Buffer Overflow Exploits

- Standard overflows work, but sometimes RAM is not executable.
- Further, the goal of an exploit is often to get code.
  - No image to work from, just a guess and a crash.
- ``Return to ROM'' like ``Return to LibC''
- Aurélien Francillon has implemented Return-Oriented-Programming for AVR microcontrollers.

Bus Usurping

- 1) Connect a GoodFET to a SPI Bus.
- 2) Boot the target device.
- 3) Halt the target MCU, leaving radio online.
- In the case of application processors (EM260, CC2480), sockets remain open and accessible!

Random Number Generators



Tools

- GoodFET for everything.
  - http://goodfet.sf.net
- Next Hope Conference Badge
  - `Hackers on a Train, eh?' this Thursday by Amtrak
  - http://amd.hope.net
- Total Phase Beagle for SPI Sniffing.

Conclusions

- Deeply Embedded Systems are a lot of fun to hack.
    - The only impediment is your fear of a soldering iron.
    - Grab a GoodFET and dump some firmware.

- A special thanks to the neighbors at Texas Instruments.

Acknowledgements

- IMME Spectrum Analyzer firmware by Mike Ossmann.
- IMME Keyboard/LCD Wiring by Dave.
- NHBadge design based upon the PIC OpenBeacon.

- Contact me if your name is Bryan and you have done related work.

Questions?

Travis Goodspeed
<Travis at RadiantMachines.com>
http://goodfet.sf.net
http://travisgoodspeed.blogspot.com