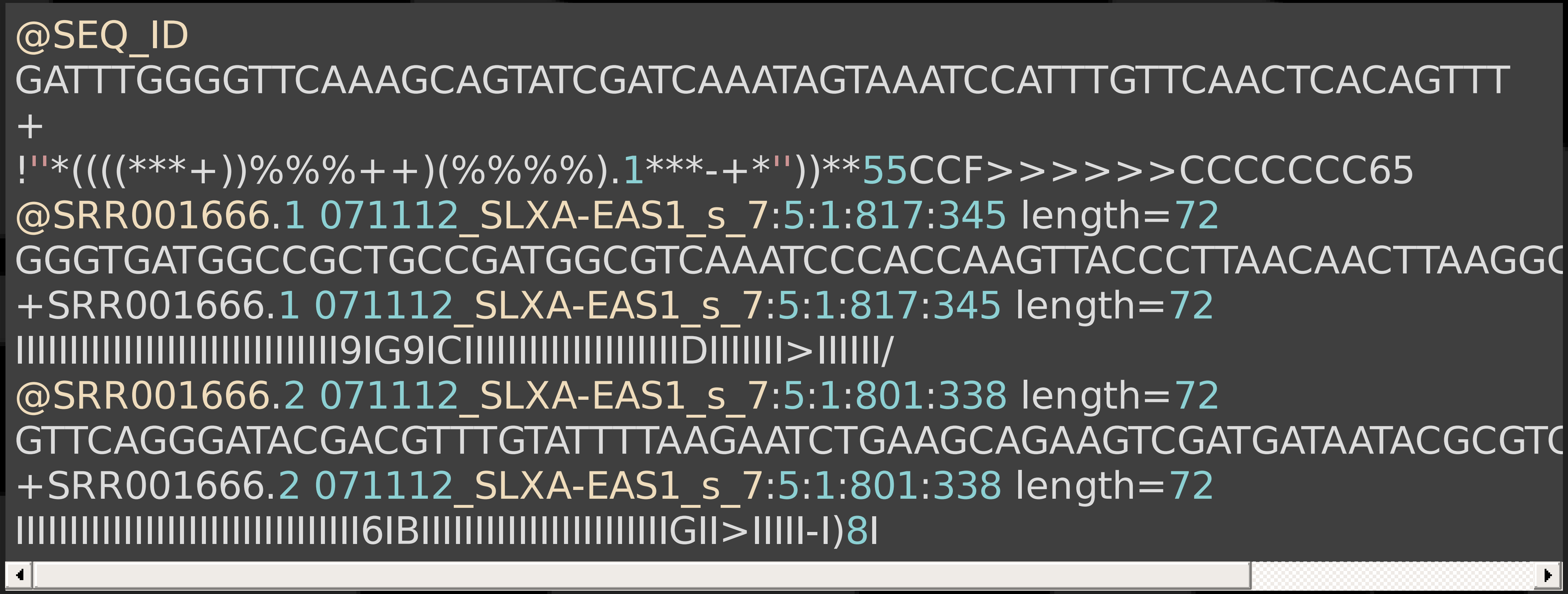
REVERSING AN AVR-BASED ESC

by @braincode

$ whoami

- @braincode on twitter, @brainstorm on github.
- An ~~outsider~~ newbie in the HW/RE community.
- Bioinformatician, so python is awesome (hugs @trufae).
- Recent hobbies: playing with MCUs such as Atmega8 and the mighty ESP8266/ESP32.
- Swedish, catalan, (heading towards Australia?).

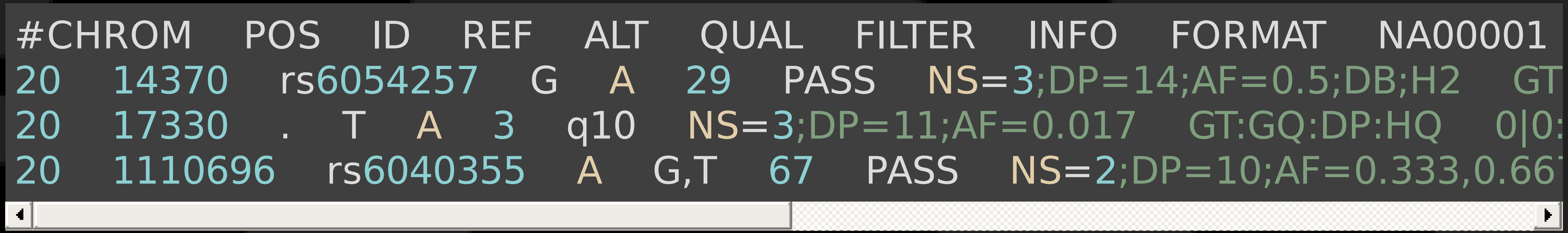
I DEAL WITH THIS...



```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
@SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
GGGTGATGGCCGCTGCCGATGGCGTCAAATCCCACCAAGTTACCCTTAACAACTTAAGGG
+SRR001666.1 071112_SLXA-EAS1_s_7:5:1:817:345 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII9IG9ICIIIIIIIIIIIIIIIIIIIIDIIIIIII>IIIIII/
@SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
GTTCAGGGATACGACGTTTGTATTTTAAGAATCTGAAGCAGAAGTCGATGATAATACGCGTC
+SRR001666.2 071112_SLXA-EAS1_s_7:5:1:801:338 length=72
IIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIIII6IBIIIIIIIIIIIIIIIIIIIIIIIGII>IIIII-I)8I
```

(FASTQ format)

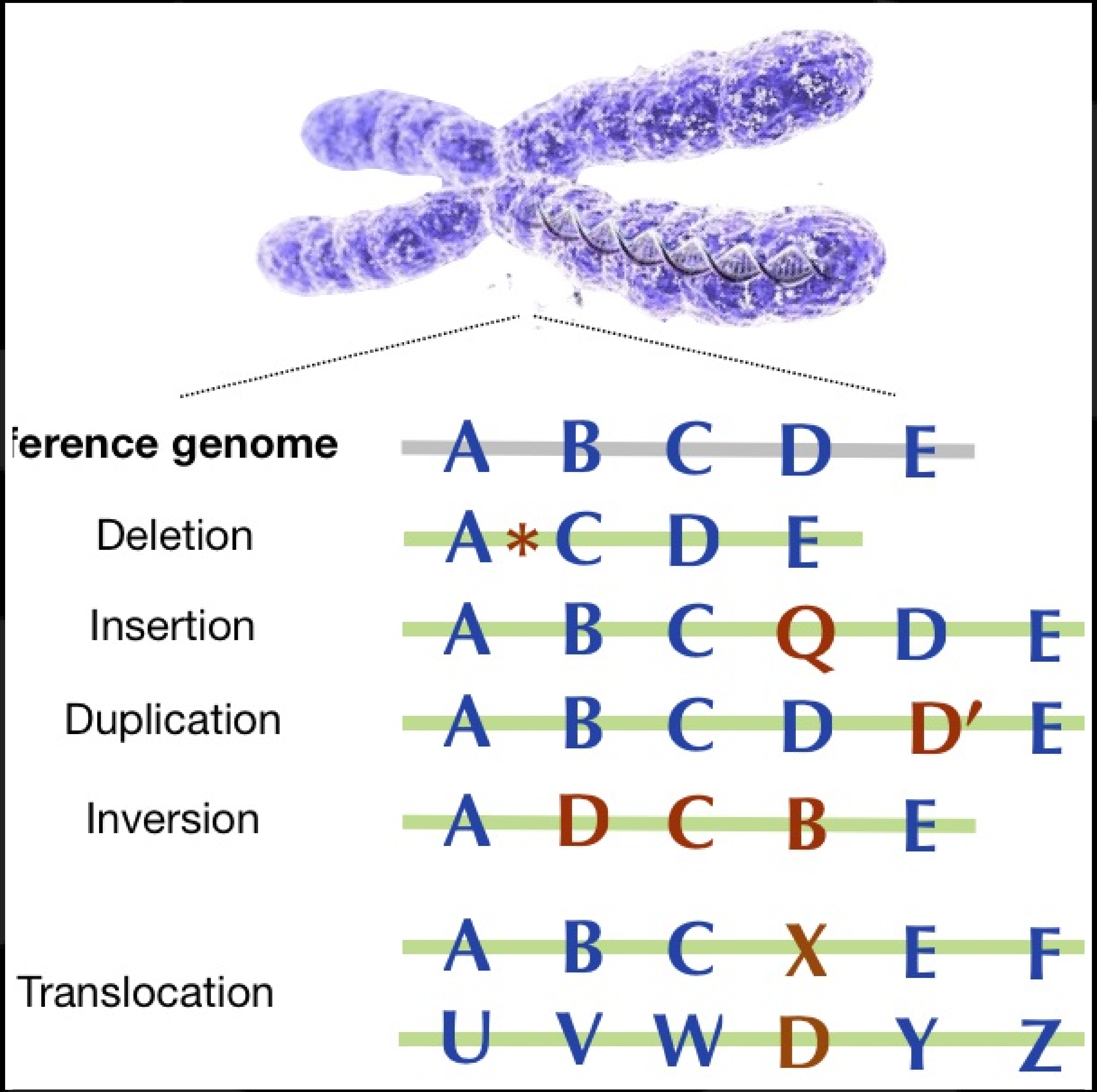
...AND THIS



```
#CHROM    POS    ID    REF    ALT    QUAL    FILTER    INFO    FORMAT    NA00001
20    14370    rs6054257    G    A    29    PASS    NS=3;DP=14;AF=0.5;DB;H2    GT
20    17330    .    T    A    3    q10    NS=3;DP=11;AF=0.017    GT:GQ:DP:HQ    0|0:
20    1110696    rs6040355    A    G,T    67    PASS    NS=2;DP=10;AF=0.333,0.66
```

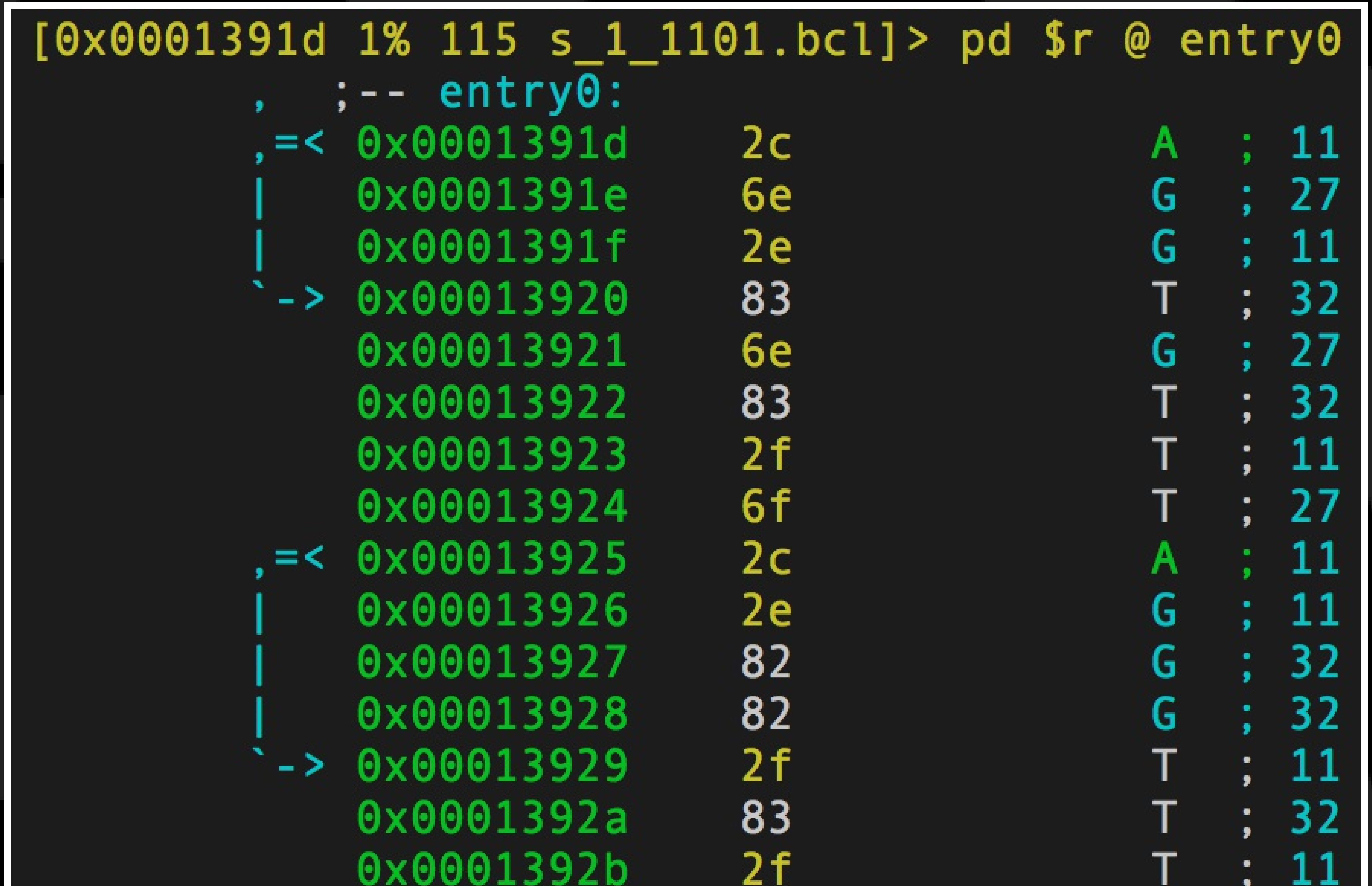
(genomic) Variant Call Format

… AND THIS TOO



ference genome
A B C D E
Deletion
A * C D E
Insertion
A B C Q D E
Duplication
A B C D D′ E
Inversion
A D C B E
Translocation
A B C X E F
U V W D Y Z

SOMETIMES THIS AS SEEN IN RADARE-EXTRAS BCL SUPPORT



```
[0x0001391d 1% 115 s_1_1101.bcl]> pd $r @ entry0
            ,   ;-- entry0:
            ,=< 0x0001391d     2c                       A   ; 11
            |   0x0001391e     6e                       G   ; 27
            |   0x0001391f     2e                       G   ; 11
            `-> 0x00013920     83                       T   ; 32
                0x00013921     6e                       G   ; 27
                0x00013922     83                       T   ; 32
                0x00013923     2f                       T   ; 11
                0x00013924     6f                       T   ; 27
            ,=< 0x00013925     2c                       A   ; 11
            |   0x00013926     2e                       G   ; 11
            |   0x00013927     82                       G   ; 32
            |   0x00013928     82                       G   ; 32
            `-> 0x00013929     2f                       T   ; 11
                0x0001392a     83                       T   ; 32
                0x0001392b     2f                       T   ; 11
```

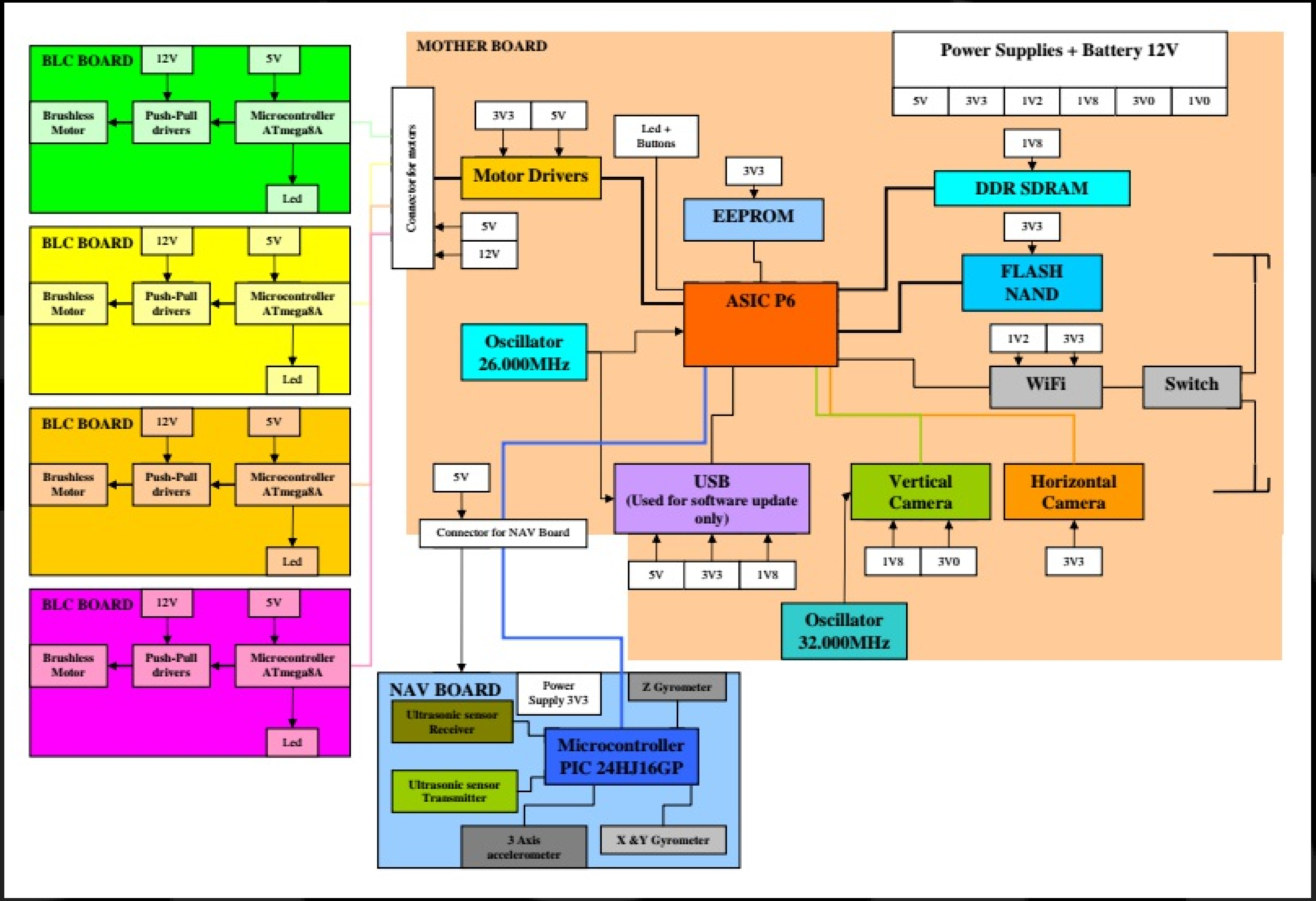
why am I here in 3 steps

1. A thing I own stopped working. I **wanted to know why**.
2. I inquired @trufae on radare's ARMv5 and AVR support.
3. Radare pullrequests #5701 #5453 #5444 #5247 #5060 and a few more issues. Impossible without @trufae and community patience and attention.

PLAN

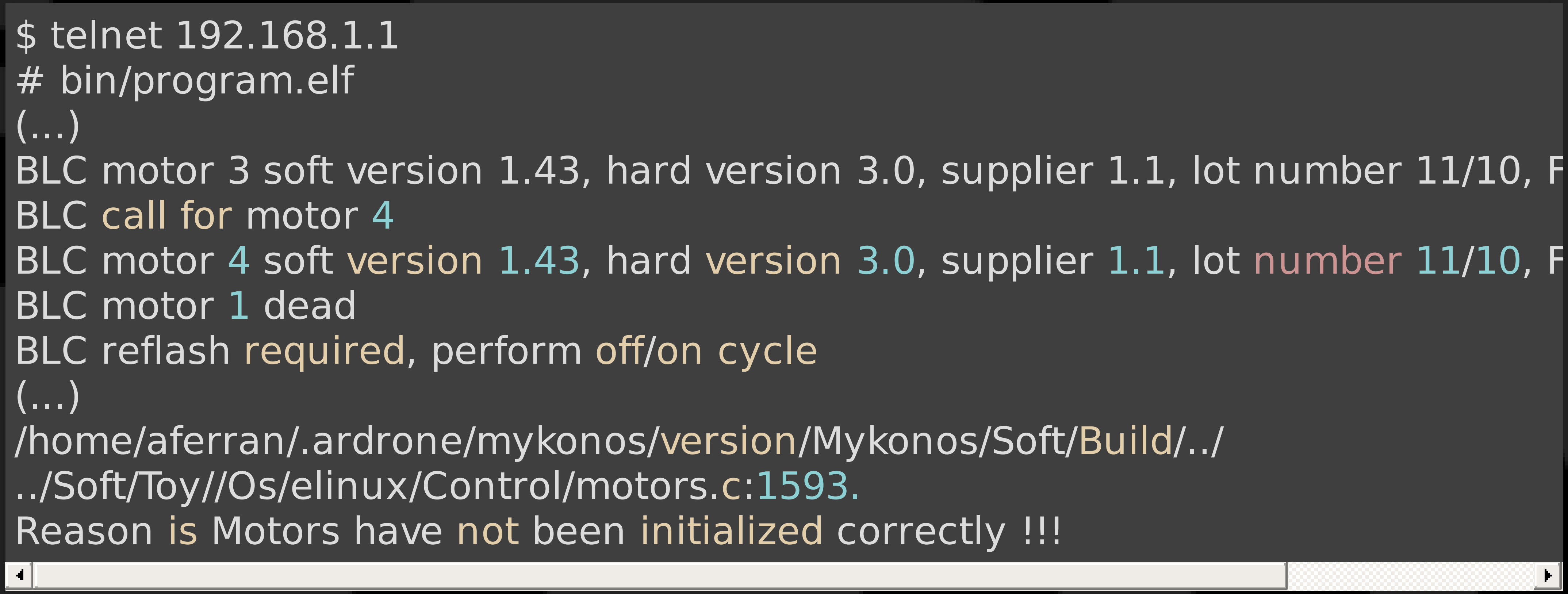
1. Identify drone parts and ICs without opening anything (thanks to Hugo Perquin and FCCID.io).
2. Discard MOSFETs failure. Focus on Atmega8 ICs onboard.
3. Docker cross-compile support for radare2, avrdude, gdb.
4. Read flash, lockbits and EEPROM from the motor AVRs.
5. (Auto-)annotate the AVR interrupt vector table.
6. Try to figure out what does what, why and how.
7. Report and help with some radare2 bugs along the way, discussing with @trufae.

FCCID.IO SCHEMATIC



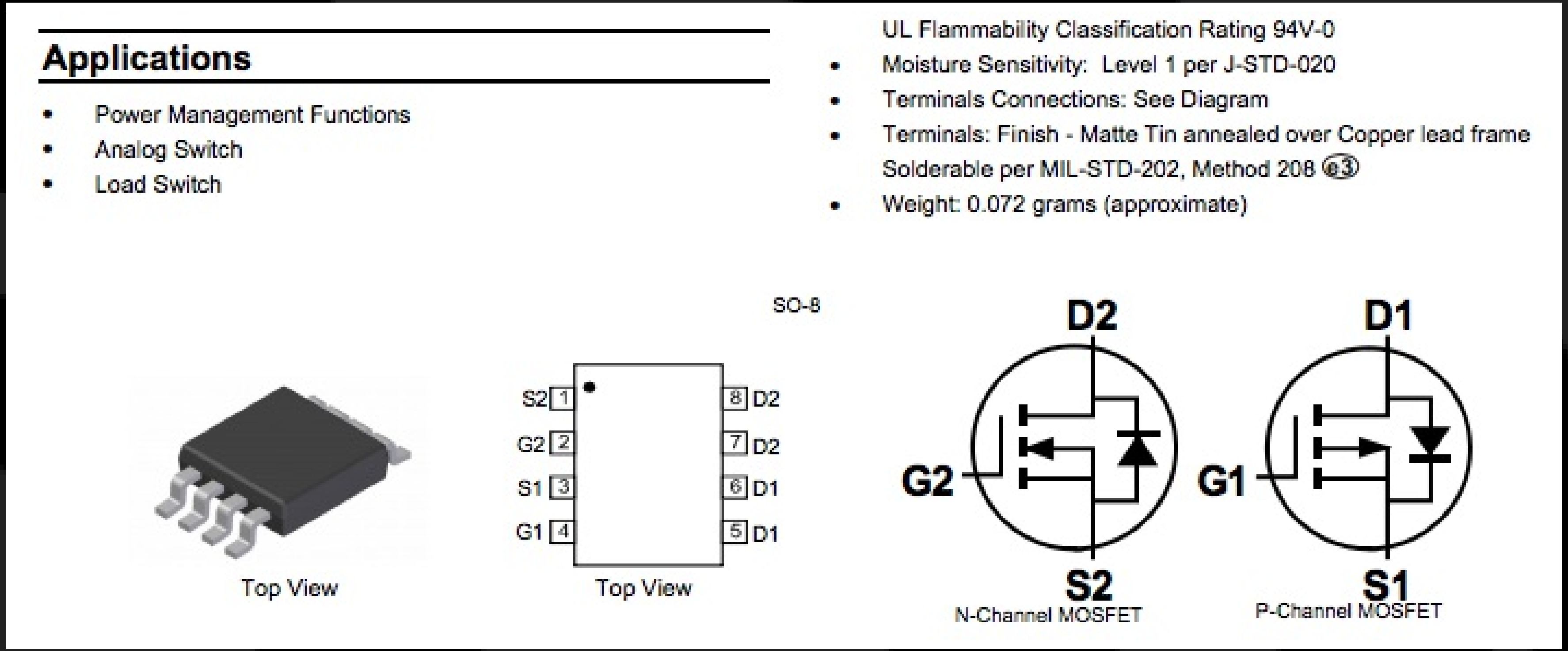
but what is wrong?

With the MOSFETs tested and healthy, there seems to be a software-only issue preventing to get one of the motors up and running:



```
$ telnet 192.168.1.1
# bin/program.elf
(...)
BLC motor 3 soft version 1.43, hard version 3.0, supplier 1.1, lot number 11/10, F
BLC call for motor 4
BLC motor 4 soft version 1.43, hard version 3.0, supplier 1.1, lot number 11/10, F
BLC motor 1 dead
BLC reflash required, perform off/on cycle
(...)
/home/aferran/.ardrone/mykonos/version/Mykonos/Soft/Build/../
../Soft/Toy//Os/elinux/Control/motors.c:1593.
Reason is Motors have not been initialized correctly !!!
```

MOSFETS



Applications

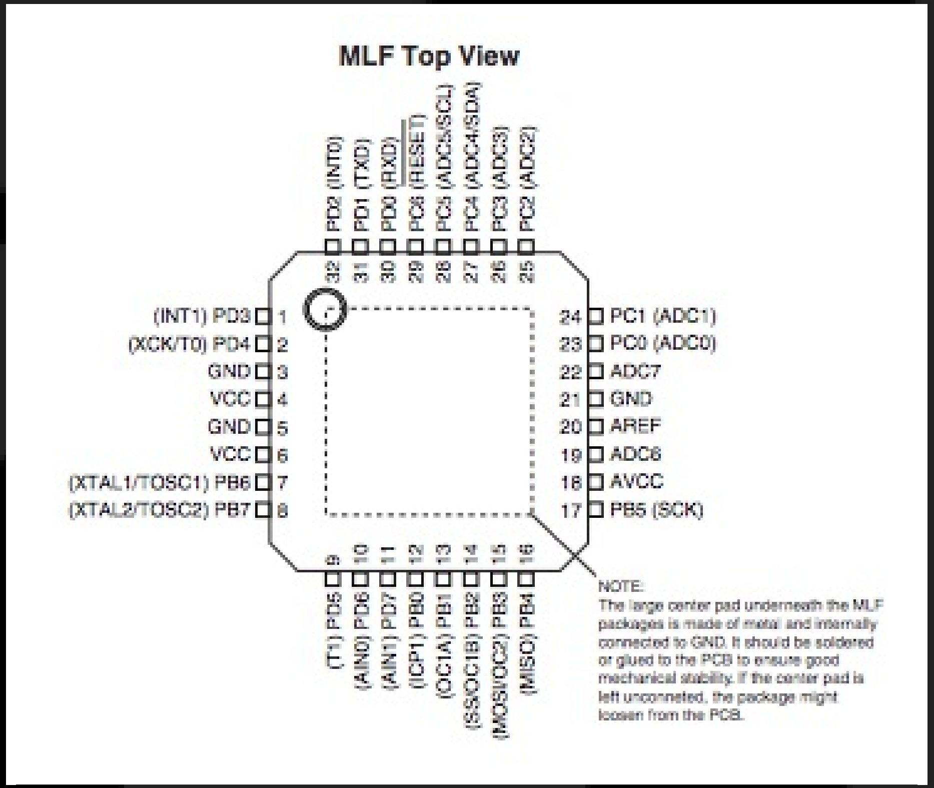
- Power Management Functions
- Analog Switch
- Load Switch

UL Flammability Classification Rating 94V-0

- Moisture Sensitivity: Level 1 per J-STD-020
- Terminals Connections: See Diagram
- Terminals: Finish - Matte Tin annealed over Copper lead frame Solderable per MIL-STD-202, Method 208 e3
- Weight: 0.072 grams (approximate)

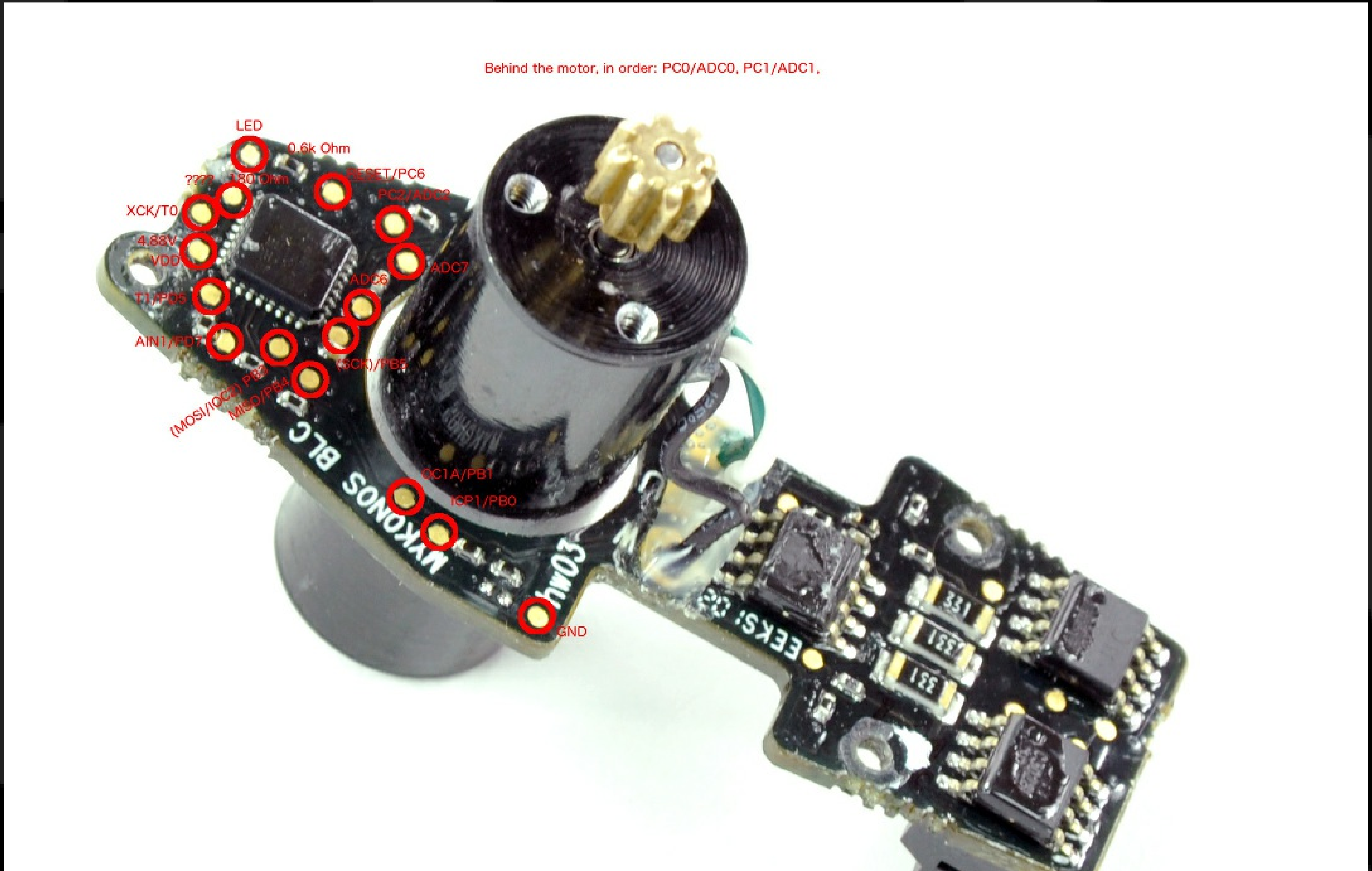
SO-8
Top View
S2 1
G2 2
S1 3
G1 4
8 D2
7 D2
6 D1
5 D1
Top View
D2
G2
S2
N-Channel MOSFET
D1
G1
S1
P-Channel MOSFET

ATMEL ATMEGA8A



NOTE:
The large center pad underneath the MLF packages is made of metal and internally connected to GND. It should be soldered or glued to the PCB to ensure good mechanical stability. If the center pad is left unconneted, the package might loosen from the PCB.

DRONE BLC


Behind the motor, in order: PC0/ADC0, PC1/ADC1,
LED
0.6k Ohm
????
180 Ohm
RESET/PC6
PC2/ADC2
XCK/T0
4.88V
VDD
ADC7
ADC6
T1/PD5
AIN1/PD7
(SCK)/PB5
(MOSI/IOC2) PB3
MISO/PB4
WYKONOS BLC
OC1A/PB1
ICP1/PB0
GND
EEKS: 03
331

Hugo Perquin points out that setting GPIOs that talk to the different motors go from 68 to 71:

```
gpio 68 -d i = select motor1
gpio 68 -d ho 0 = deselect motor1
gpio 69 -d i = select motor2
gpio 69 -d ho 0 = deselect motor2
gpio 70 -d i = select motor3
gpio 70 -d ho 0 = deselect motor3
gpio 71 -d i = select motor4
gpio 71 -d ho 0 = deselect motor4
```

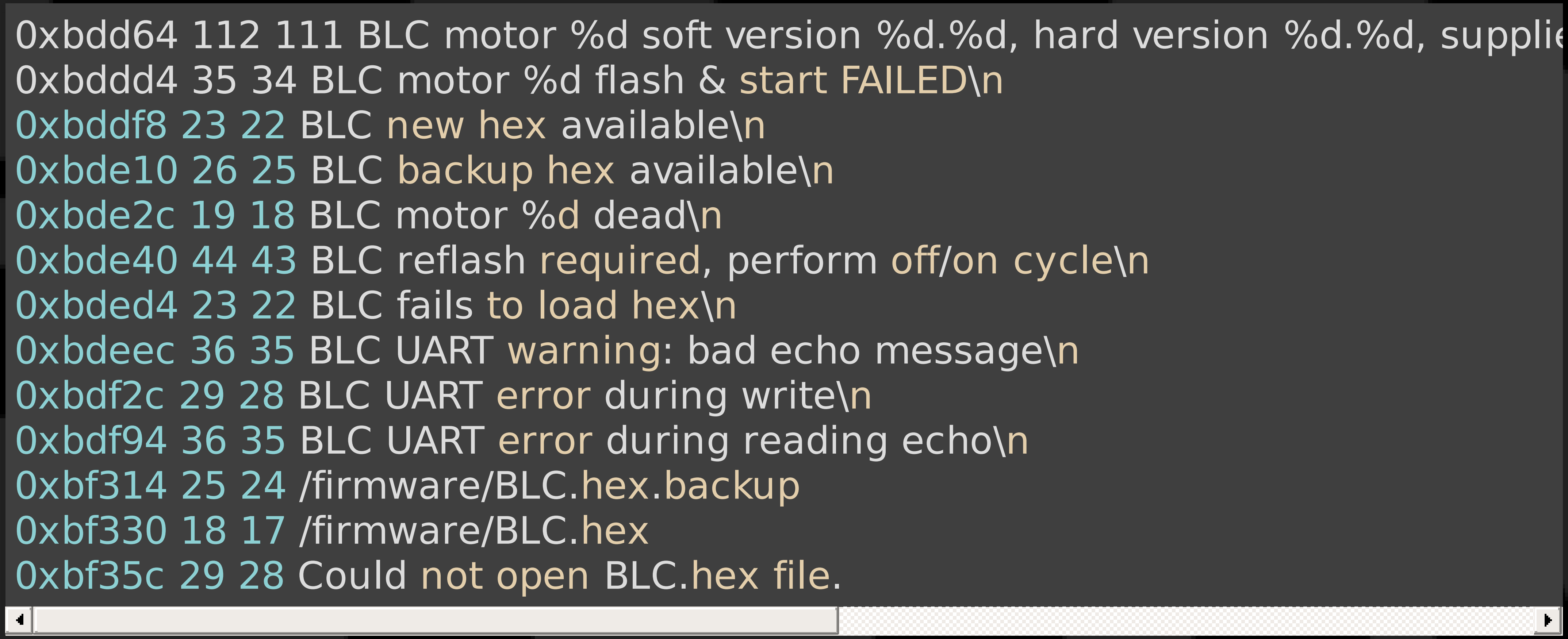
So after typing a few commands in radare against **program.elf...**:

```
[0x00000000]> izzq~BLC     # looking for "BLC" (motor-related) str
[0x00000000]> s 0xbde40    # what is there around those strings?
[0x000bde40]> V
[0x000bde40]> s 0x28fcc    # hm, that function looks interesting
[0x00028fcc]> V
[0x00028fcc]> s 0x28b5c    # yeah, let's look at the callgraph
[0x00028b5c]> VV
```

STRINGS

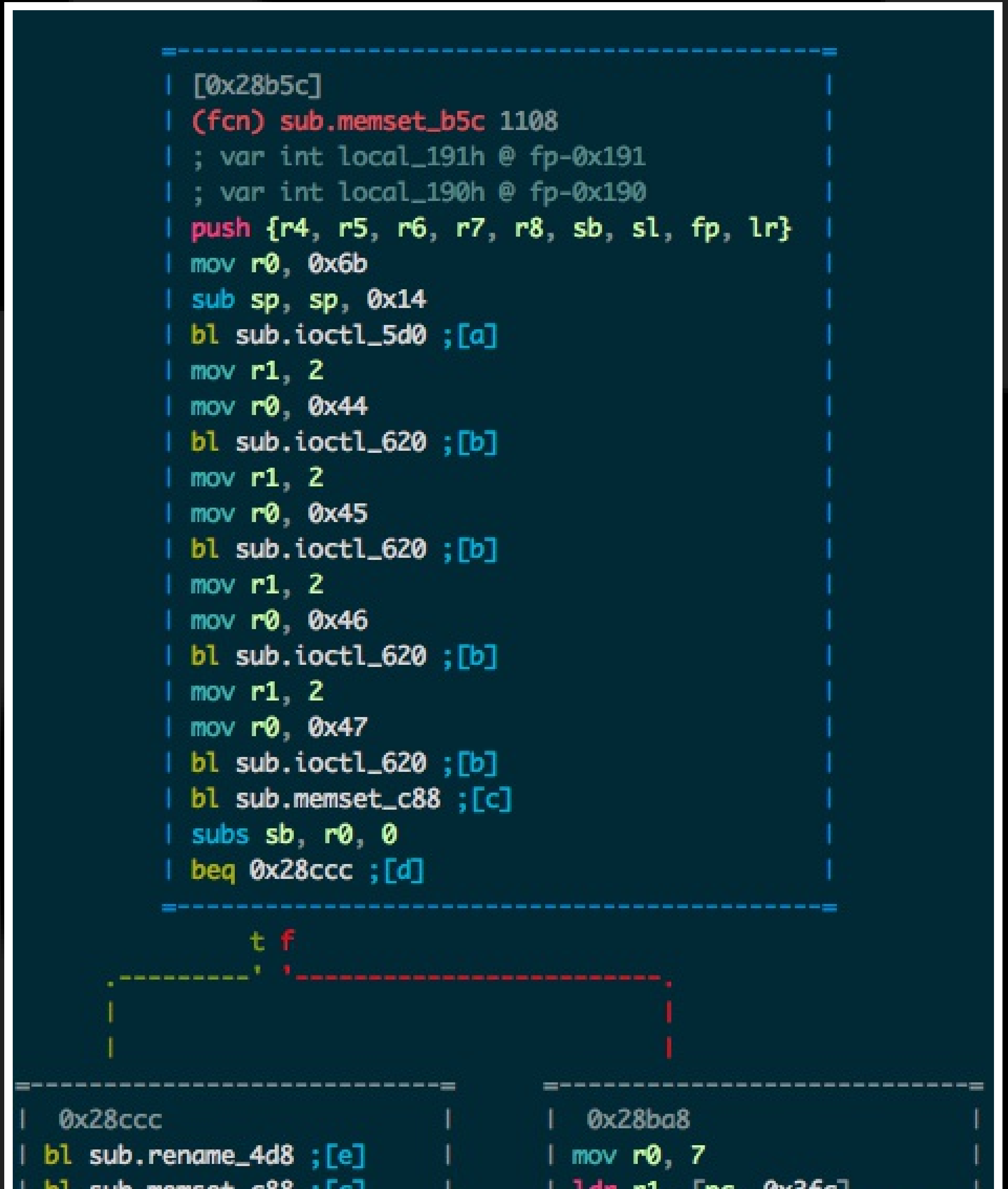
```
[0x00000040]> izzq~BLC
0xbdbfc 38 37 BLC failed to allocate parsed memory\n
0xbdc30 19 18 BLC.hex too large\n
0xbdc44 12 11 BLC verify\n
0xbdc50 30 29 BLC verify FAILED - page %d \n
0xbdc70 15 14 BLC verify OK\n
0xbdc80 17 16 BLC start flash\n
0xbdc94 19 18 BLC flash aborted\n
0xbdca8 16 15 BLC flash done\n
0xbdcb8 23 22 BLC call for motor %d\n
0xbdcd0 45 44 BLC check BLC memory corruption on motor %d\n
0xbdd00 34 33 BLC memory corrupted on motor %d\n
0xbdd24 30 29 BLC erase memory on motor %d\n
0xbdd44 30 29 BLC start FAILED on motor %d\n
```

STRINGS CONT'D



```
0xbdd64 112 111 BLC motor %d soft version %d.%d, hard version %d.%d, supplie
0xbddd4 35 34 BLC motor %d flash & start FAILED\n
0xbddf8 23 22 BLC new hex available\n
0xbde10 26 25 BLC backup hex available\n
0xbde2c 19 18 BLC motor %d dead\n
0xbde40 44 43 BLC reflash required, perform off/on cycle\n
0xbded4 23 22 BLC fails to load hex\n
0xbdeec 36 35 BLC UART warning: bad echo message\n
0xbdf2c 29 28 BLC UART error during write\n
0xbdf94 36 35 BLC UART error during reading echo\n
0xbf314 25 24 /firmware/BLC.hex.backup
0xbf330 18 17 /firmware/BLC.hex
0xbf35c 29 28 Could not open BLC.hex file.
```

... we have the motors IOCTL/GPIO main logic from the ARMv5 IC. A.k.a "program.elf" inside the drone:

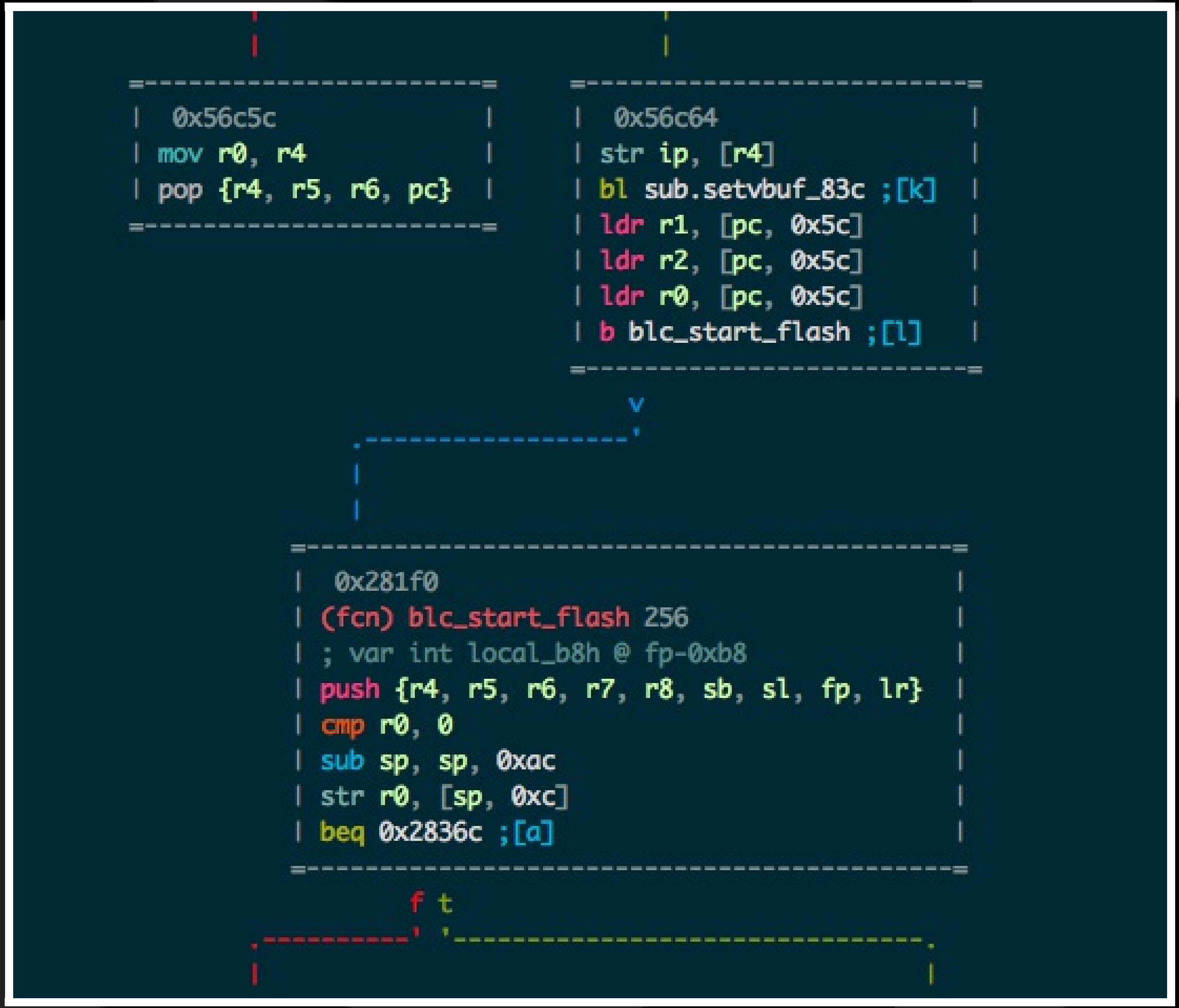


```
          =---------------------------------------------=
          | [0x28b5c]                                   |
          | (fcn) sub.memset_b5c 1108                   |
          | ; var int local_191h @ fp-0x191             |
          | ; var int local_190h @ fp-0x190             |
          | push {r4, r5, r6, r7, r8, sb, sl, fp, lr}   |
          | mov r0, 0x6b                                |
          | sub sp, sp, 0x14                            |
          | bl sub.ioctl_5d0 ;[a]                       |
          | mov r1, 2                                   |
          | mov r0, 0x44                                |
          | bl sub.ioctl_620 ;[b]                       |
          | mov r1, 2                                   |
          | mov r0, 0x45                                |
          | bl sub.ioctl_620 ;[b]                       |
          | mov r1, 2                                   |
          | mov r0, 0x46                                |
          | bl sub.ioctl_620 ;[b]                       |
          | mov r1, 2                                   |
          | mov r0, 0x47                                |
          | bl sub.ioctl_620 ;[b]                       |
          | bl sub.memset_c88 ;[c]                      |
          | subs sb, r0, 0                              |
          | beq 0x28ccc ;[d]                            |
          =---------------------------------------------=
                t f
       .---------' '--------------------------.
       |                                      |
       |                                      |
=-----------------------------=     =-----------------------------=
|  0x28ccc                    |     |  0x28ba8                    |
| bl sub.rename_4d8 ;[e]      |     | mov r0, 7                   |
```

IOCTL'S RAX2'D

```
$ rax2 0x44 0x45 0x46 0x47
68
69
70
71
```

BLC_MOTORS_MAIN

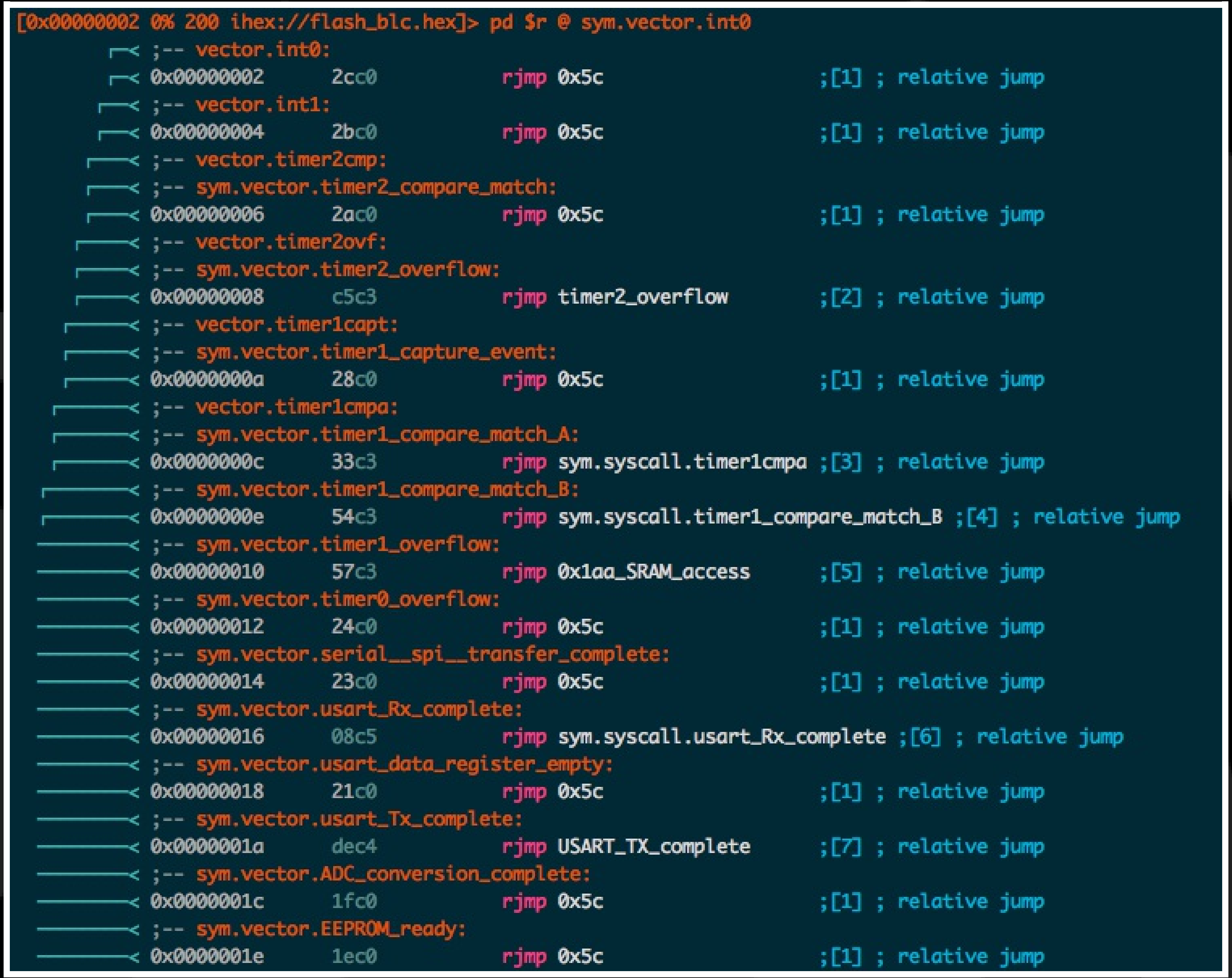


```
           |                             |
    =------------------------=     =---------------------------=
    |  0x56c5c               |     |  0x56c64                  |
    | mov r0, r4             |     | str ip, [r4]              |
    | pop {r4, r5, r6, pc}   |     | bl sub.setvbuf_83c ;[k]   |
    =------------------------=     | ldr r1, [pc, 0x5c]        |
                                   | ldr r2, [pc, 0x5c]        |
                                   | ldr r0, [pc, 0x5c]        |
                                   | b blc_start_flash ;[l]    |
                                   =---------------------------=
                                         v
                     .------------------'
                     |
                     |
          =-------------------------------------------------=
          |  0x281f0                                         |
          | (fcn) blc_start_flash 256                        |
          | ; var int local_b8h @ fp-0xb8                    |
          | push {r4, r5, r6, r7, r8, sb, sl, fp, lr}        |
          | cmp r0, 0                                        |
          | sub sp, sp, 0xac                                 |
          | str r0, [sp, 0xc]                                |
          | beq 0x2836c ;[a]                                 |
          =-------------------------------------------------=
                  f t
        .----------' '--------------------------------.
        |                                              |
```

BLC_MOTORS_* FUNCTIONS TL;DR

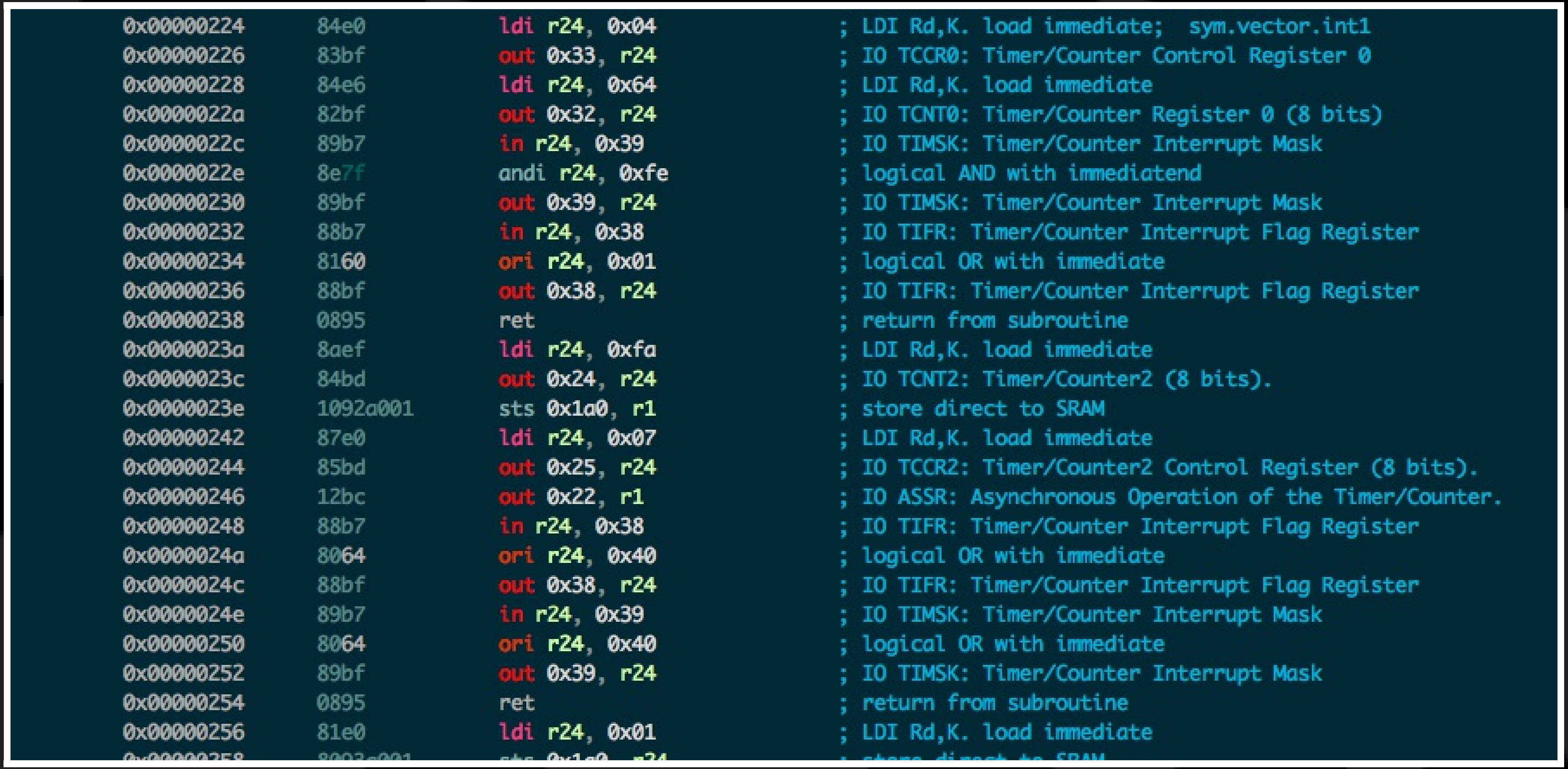
1. Read and/or backup the AVR ihex file for the motor(s) in the drone's linux UBIFS filesystem.
2. **radiff2 atmega8.ihex BLC.hex** was broken. @trufae fixed it.
3. Make sure motors are ok and reflash them as needed.
4. Send commands to the motors, check status. Cutout flip-flops when drone crashes.

ATMEGA8A IN EACH BLC



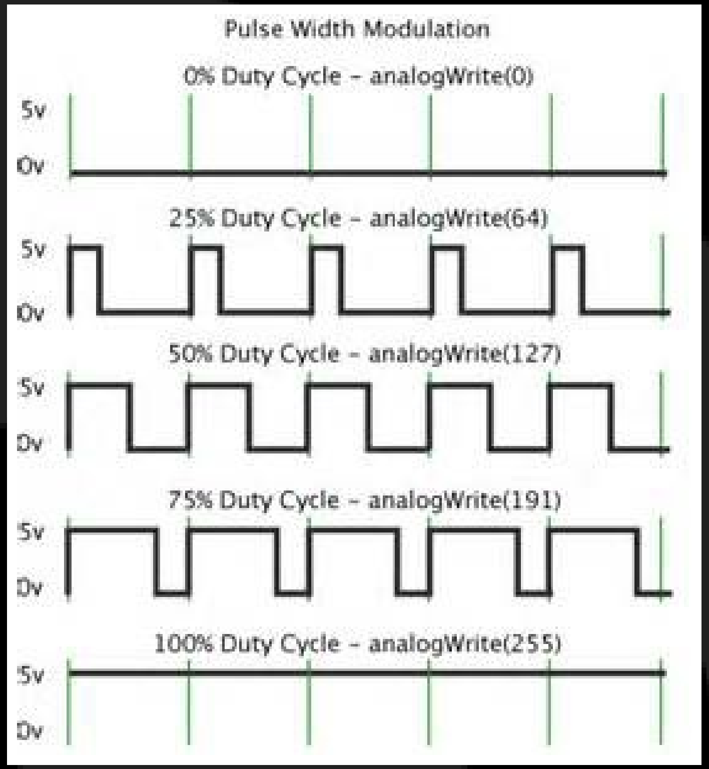
```
[0x00000002 0% 200 ihex://flash_blc.hex]> pd $r @ sym.vector.int0
        ┌─< ;-- vector.int0:
        ┌─< 0x00000002      2cc0           rjmp 0x5c                     ;[1] ; relative jump
       ┌──< ;-- vector.int1:
       ┌──< 0x00000004      2bc0           rjmp 0x5c                     ;[1] ; relative jump
      ┌───< ;-- vector.timer2cmp:
      ┌───< ;-- sym.vector.timer2_compare_match:
      ┌───< 0x00000006      2ac0           rjmp 0x5c                     ;[1] ; relative jump
     ┌────< ;-- vector.timer2ovf:
     ┌────< ;-- sym.vector.timer2_overflow:
     ┌────< 0x00000008      c5c3           rjmp timer2_overflow          ;[2] ; relative jump
    ┌─────< ;-- vector.timer1capt:
    ┌─────< ;-- sym.vector.timer1_capture_event:
    ┌─────< 0x0000000a      28c0           rjmp 0x5c                     ;[1] ; relative jump
   ┌──────< ;-- vector.timer1cmpa:
   ┌──────< ;-- sym.vector.timer1_compare_match_A:
   ┌──────< 0x0000000c      33c3           rjmp sym.syscall.timer1cmpa ;[3] ; relative jump
  ┌───────< ;-- sym.vector.timer1_compare_match_B:
  ┌───────< 0x0000000e      54c3           rjmp sym.syscall.timer1_compare_match_B ;[4] ; relative jump
  ────────< ;-- sym.vector.timer1_overflow:
  ────────< 0x00000010      57c3           rjmp 0x1aa_SRAM_access        ;[5] ; relative jump
  ────────< ;-- sym.vector.timer0_overflow:
  ────────< 0x00000012      24c0           rjmp 0x5c                     ;[1] ; relative jump
  ────────< ;-- sym.vector.serial__spi__transfer_complete:
  ────────< 0x00000014      23c0           rjmp 0x5c                     ;[1] ; relative jump
  ────────< ;-- sym.vector.usart_Rx_complete:
  ────────< 0x00000016      08c5           rjmp sym.syscall.usart_Rx_complete ;[6] ; relative jump
  ────────< ;-- sym.vector.usart_data_register_empty:
  ────────< 0x00000018      21c0           rjmp 0x5c                     ;[1] ; relative jump
  ────────< ;-- sym.vector.usart_Tx_complete:
  ────────< 0x0000001a      dec4           rjmp USART_TX_complete        ;[7] ; relative jump
  ────────< ;-- sym.vector.ADC_conversion_complete:
  ────────< 0x0000001c      1fc0           rjmp 0x5c                     ;[1] ; relative jump
  ────────< ;-- sym.vector.EEPROM_ready:
  ────────< 0x0000001e      1ec0           rjmp 0x5c                     ;[1] ; relative jump
```

TIMERS FOR PWM



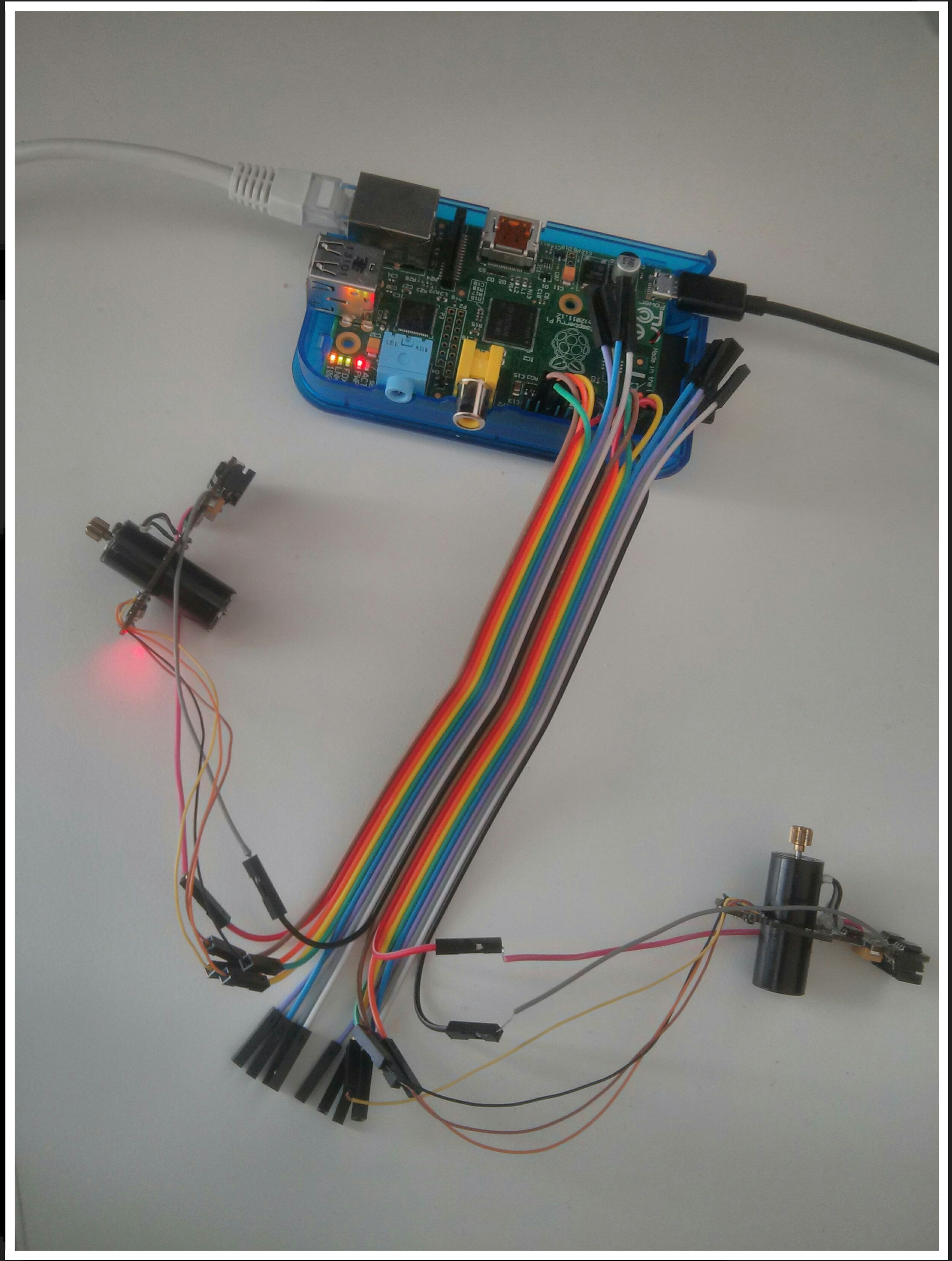
```
0x00000224      84e0            ldi r24, 0x04               ; LDI Rd,K. load immediate;  sym.vector.int1
0x00000226      83bf            out 0x33, r24               ; IO TCCR0: Timer/Counter Control Register 0
0x00000228      84e6            ldi r24, 0x64               ; LDI Rd,K. load immediate
0x0000022a      82bf            out 0x32, r24               ; IO TCNT0: Timer/Counter Register 0 (8 bits)
0x0000022c      89b7            in r24, 0x39                ; IO TIMSK: Timer/Counter Interrupt Mask
0x0000022e      8e7f            andi r24, 0xfe              ; logical AND with immediatend
0x00000230      89bf            out 0x39, r24               ; IO TIMSK: Timer/Counter Interrupt Mask
0x00000232      88b7            in r24, 0x38                ; IO TIFR: Timer/Counter Interrupt Flag Register
0x00000234      8160            ori r24, 0x01               ; logical OR with immediate
0x00000236      88bf            out 0x38, r24               ; IO TIFR: Timer/Counter Interrupt Flag Register
0x00000238      0895            ret                         ; return from subroutine
0x0000023a      8aef            ldi r24, 0xfa               ; LDI Rd,K. load immediate
0x0000023c      84bd            out 0x24, r24               ; IO TCNT2: Timer/Counter2 (8 bits).
0x0000023e      1092a001        sts 0x1a0, r1               ; store direct to SRAM
0x00000242      87e0            ldi r24, 0x07               ; LDI Rd,K. load immediate
0x00000244      85bd            out 0x25, r24               ; IO TCCR2: Timer/Counter2 Control Register (8 bits).
0x00000246      12bc            out 0x22, r1                ; IO ASSR: Asynchronous Operation of the Timer/Counter.
0x00000248      88b7            in r24, 0x38                ; IO TIFR: Timer/Counter Interrupt Flag Register
0x0000024a      8064            ori r24, 0x40               ; logical OR with immediate
0x0000024c      88bf            out 0x38, r24               ; IO TIFR: Timer/Counter Interrupt Flag Register
0x0000024e      89b7            in r24, 0x39                ; IO TIMSK: Timer/Counter Interrupt Mask
0x00000250      8064            ori r24, 0x40               ; logical OR with immediate
0x00000252      89bf            out 0x39, r24               ; IO TIMSK: Timer/Counter Interrupt Mask
0x00000254      0895            ret                         ; return from subroutine
0x00000256      81e0            ldi r24, 0x01               ; LDI Rd,K. load immediate
0x00000258      8093a001        sts 0x1a0, r24              ; store direct to SRAM
```

PWM

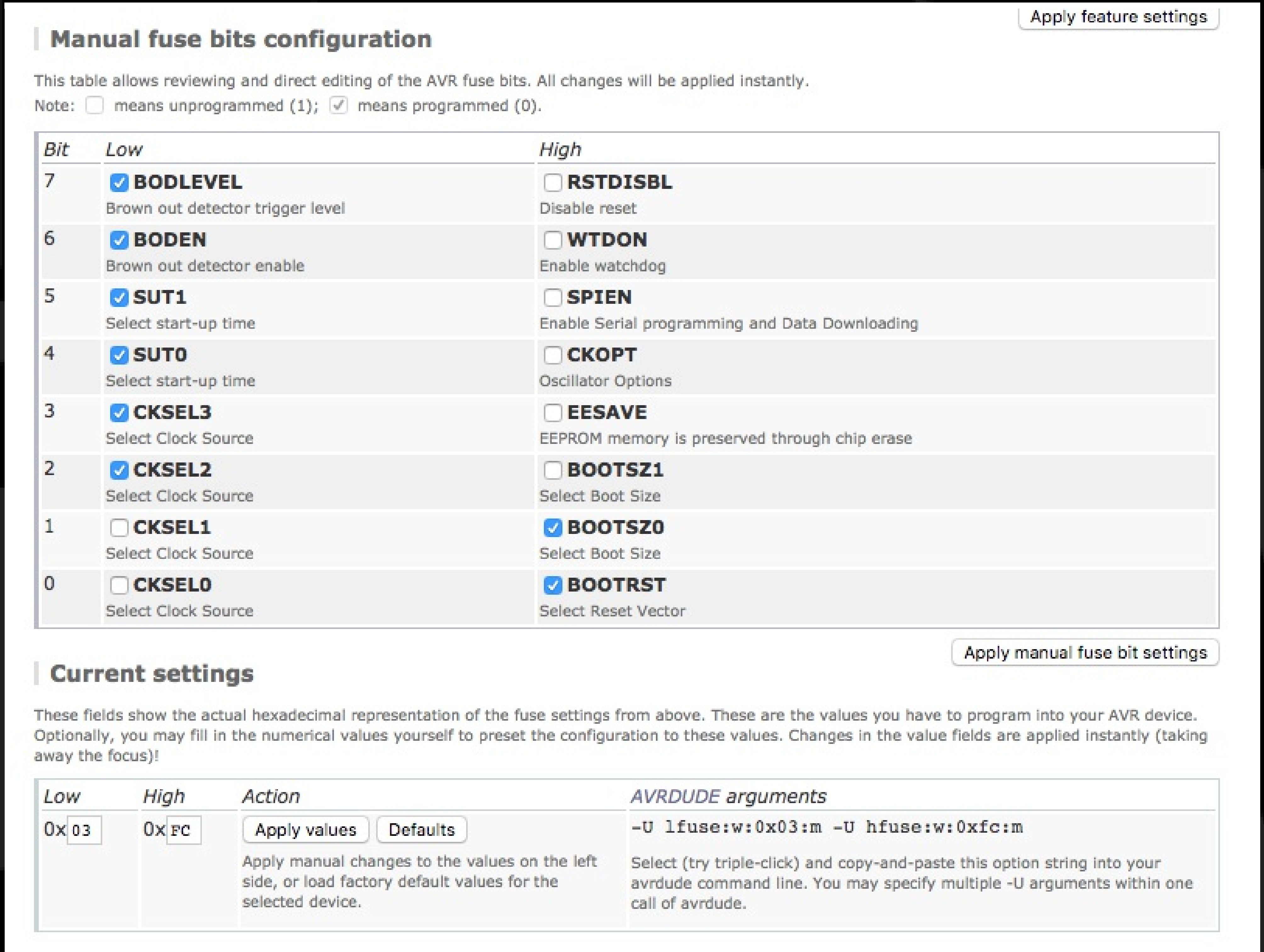

Pulse Width Modulation
0% Duty Cycle - analogWrite(0)
25% Duty Cycle - analogWrite(64)
50% Duty Cycle - analogWrite(127)
75% Duty Cycle - analogWrite(191)
100% Duty Cycle - analogWrite(255)
5v
0v

DIFFING HARDWARE

How does the "dead" motor compare to the "healthy" one?



FUSE BITS (03 FC): "DEAD" MOTOR



Apply feature settings

Manual fuse bits configuration

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.
Note: ☐ means unprogrammed (1); ✓ means programmed (0).

Bit	*Low*	*High*
7	✓ **BODLEVEL** Brown out detector trigger level	☐ **RSTDISBL** Disable reset
6	✓ **BODEN** Brown out detector enable	☐ **WTDON** Enable watchdog
5	✓ **SUT1** Select start-up time	☐ **SPIEN** Enable Serial programming and Data Downloading
4	✓ **SUT0** Select start-up time	☐ **CKOPT** Oscillator Options
3	✓ **CKSEL3** Select Clock Source	☐ **EESAVE** EEPROM memory is preserved through chip erase
2	✓ **CKSEL2** Select Clock Source	☐ **BOOTSZ1** Select Boot Size
1	☐ **CKSEL1** Select Clock Source	✓ **BOOTSZ0** Select Boot Size
0	☐ **CKSEL0** Select Clock Source	✓ **BOOTRST** Select Reset Vector

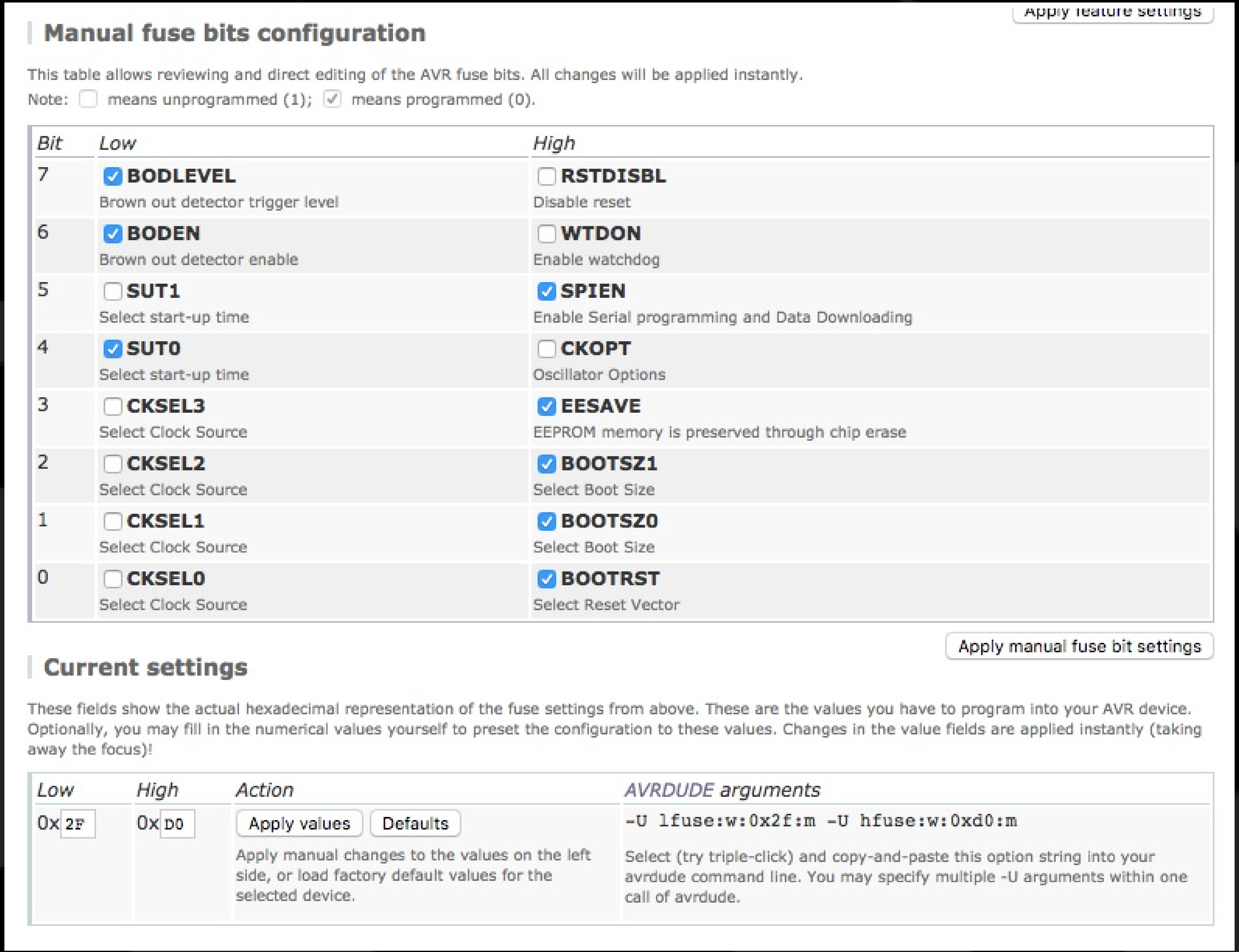
Apply manual fuse bit settings

Current settings

These fields show the actual hexadecimal representation of the fuse settings from above. These are the values you have to program into your AVR device. Optionally, you may fill in the numerical values yourself to preset the configuration to these values. Changes in the value fields are applied instantly (taking away the focus)!

Low	*High*	*Action*	*AVRDUDE arguments*
0x 03	0x FC	Apply values / Defaults Apply manual changes to the values on the left side, or load factory default values for the selected device.	`-U lfuse:w:0x03:m -U hfuse:w:0xfc:m` Select (try triple-click) and copy-and-paste this option string into your avrdude command line. You may specify multiple -U arguments within one call of avrdude.

FUSE BITS (2F D0): GOOD MOTOR



Apply feature settings

Manual fuse bits configuration

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.
Note: ☐ means unprogrammed (1); ✓ means programmed (0).

Bit	Low	High
7	✓ **BODLEVEL** Brown out detector trigger level	☐ **RSTDISBL** Disable reset
6	✓ **BODEN** Brown out detector enable	☐ **WTDON** Enable watchdog
5	☐ **SUT1** Select start-up time	✓ **SPIEN** Enable Serial programming and Data Downloading
4	✓ **SUT0** Select start-up time	☐ **CKOPT** Oscillator Options
3	☐ **CKSEL3** Select Clock Source	✓ **EESAVE** EEPROM memory is preserved through chip erase
2	☐ **CKSEL2** Select Clock Source	✓ **BOOTSZ1** Select Boot Size
1	☐ **CKSEL1** Select Clock Source	✓ **BOOTSZ0** Select Boot Size
0	☐ **CKSEL0** Select Clock Source	✓ **BOOTRST** Select Reset Vector

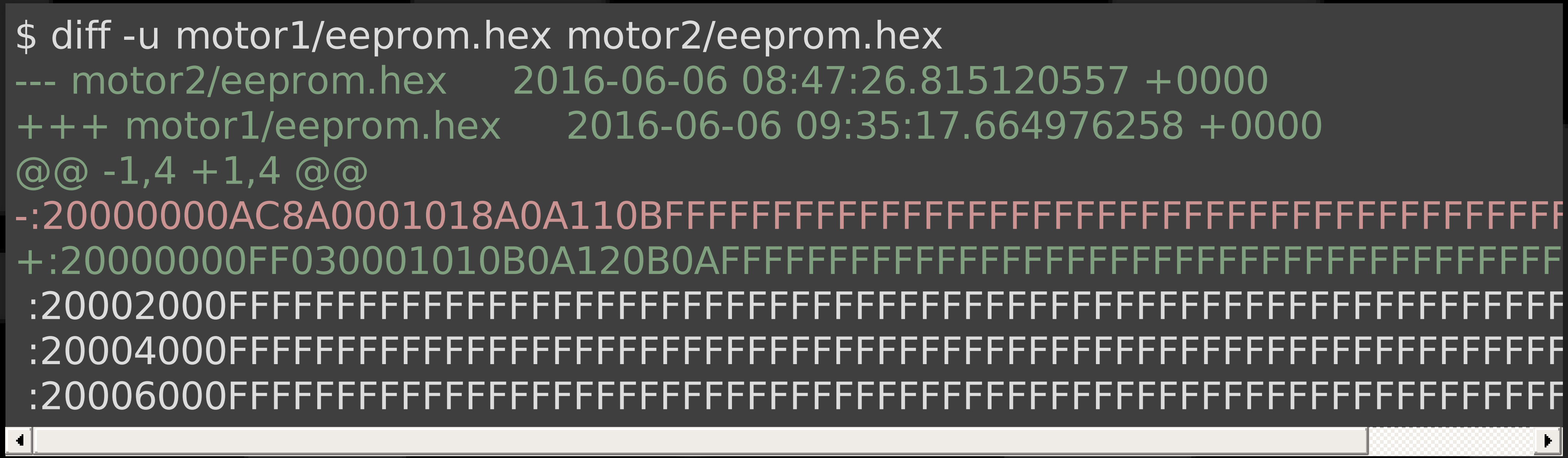
Apply manual fuse bit settings

Current settings

These fields show the actual hexadecimal representation of the fuse settings from above. These are the values you have to program into your AVR device. Optionally, you may fill in the numerical values yourself to preset the configuration to these values. Changes in the value fields are applied instantly (taking away the focus)!

Low	High	Action	AVRDUDE arguments
0x 2F	0x D0	Apply values · Defaults Apply manual changes to the values on the left side, or load factory default values for the selected device.	`-U lfuse:w:0x2f:m -U hfuse:w:0xd0:m` Select (try triple-click) and copy-and-paste this option string into your avrdude command line. You may specify multiple -U arguments within one call of avrdude.

EEPROM



```
$ diff -u motor1/eeprom.hex motor2/eeprom.hex
--- motor2/eeprom.hex     2016-06-06 08:47:26.815120557 +0000
+++ motor1/eeprom.hex     2016-06-06 09:35:17.664976258 +0000
@@ -1,4 +1,4 @@
-:20000000AC8A0001018A0A110BFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
+:20000000FF030001010B0A120B0AFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 :20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 :20004000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
 :20006000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

No idea what this is yet... did I mention we badly need ESIL for AVR to make sense of memory mappings?

AVRDUDE SAVES THE DAY

```
avrdude -p m8 -C /etc/avrdude.conf -c motor_1 -e
avrdude -p m8 -C /etc/avrdude.conf -c motor_1 -U flash:w:flash.hex:i
```

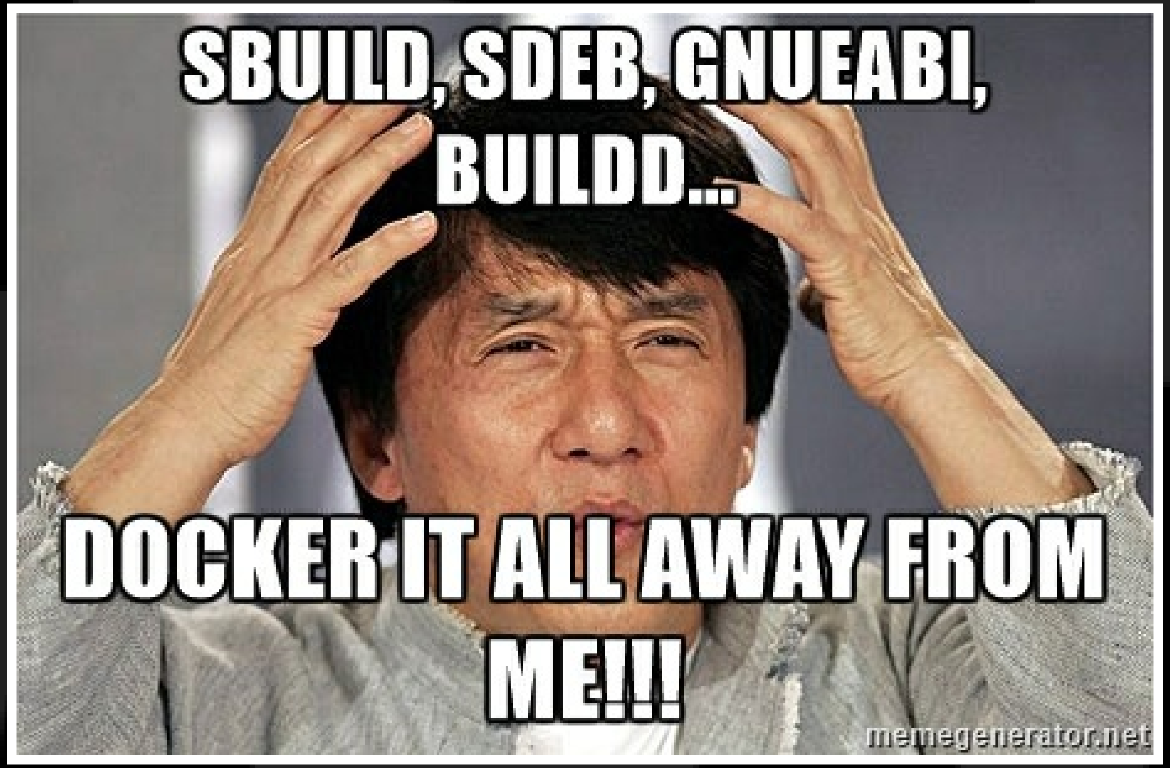
Erasing the entire chip and flashing the flash section back.

AVRDUDE INSIDE THE DRONE

Thought: why not put AVRdude inside the drone for when things go wrong again with the motor(s)?

ARMv5 does not have "hardfloat" support, so any recent cross-compiler will result into an "Illegal instruction" error.

Setting up a cross-compiling toolchain is a bit of a PITA.


SBUILD, SDEB, GNUEABI, BUILDD...
DOCKER IT ALL AWAY FROM ME!!!
memegenerator.net

WRAP THE MESS INSIDE DOCKER WITH DOCKCROSS

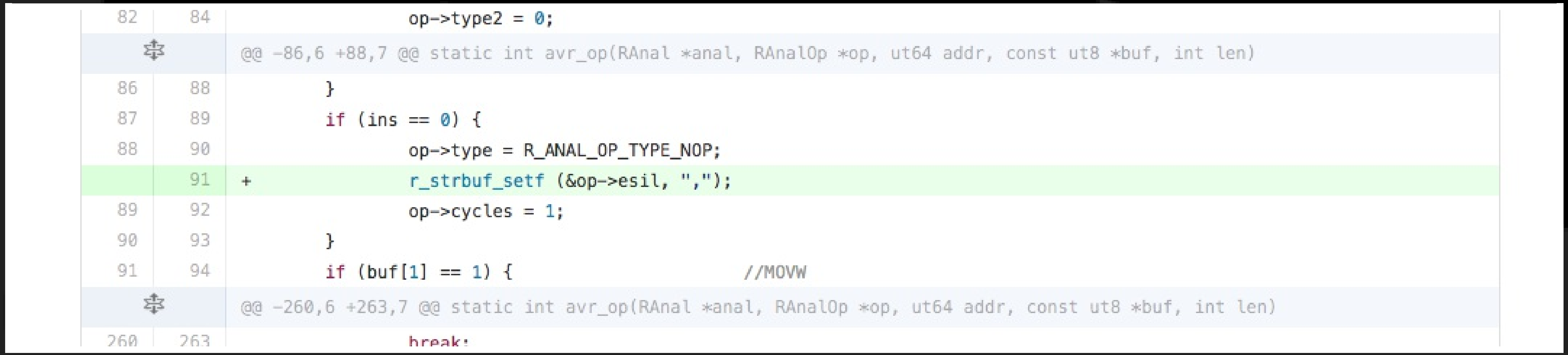
http://radare.today/posts/dockcross/

(and cross-compile radare2 for ARMv5 while we are at it)

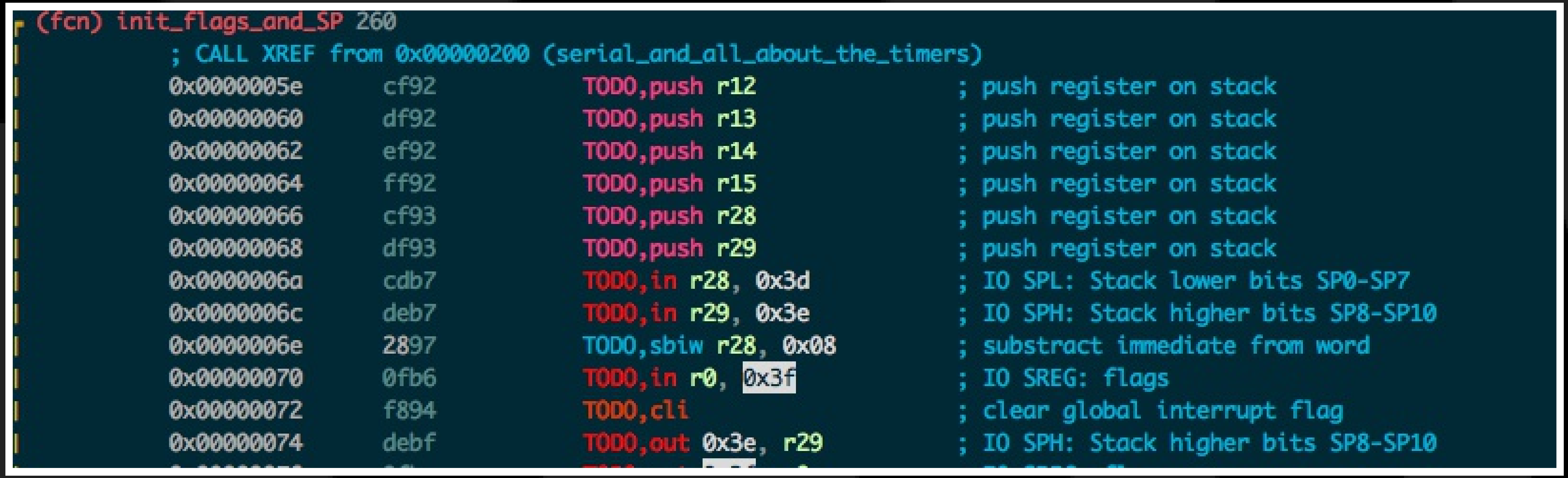
CAPTURE THE R2CON2016 PRIZE!

THIS DRONE CAN BE YOURS!

WITH A LITTLE BIT OF ESIL HACKING



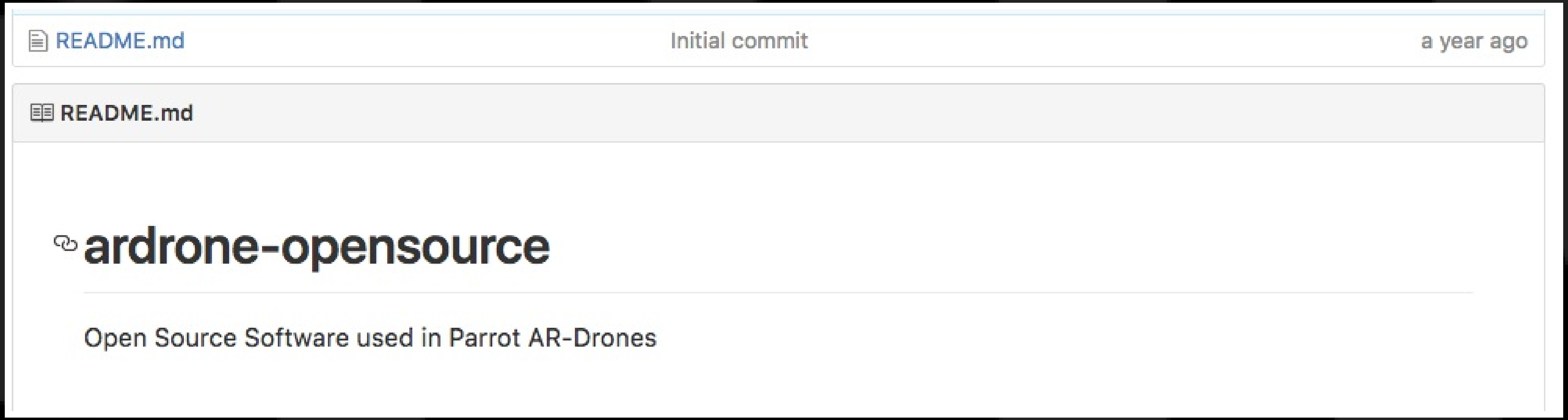
```
                op->type2 = 0;
@@ -86,6 +88,7 @@ static int avr_op(RAnal *anal, RAnalOp *op, ut64 addr, const ut8 *buf, int len)
        }
        if (ins == 0) {
                op->type = R_ANAL_OP_TYPE_NOP;
+               r_strbuf_setf (&op->esil, ",");
                op->cycles = 1;
        }
        if (buf[1] == 1) {                      //MOVW
@@ -260,6 +263,7 @@ static int avr_op(RAnal *anal, RAnalOp *op, ut64 addr, const ut8 *buf, int len)
```



```
┌ (fcn) init_flags_and_SP 260
|          ; CALL XREF from 0x00000200 (serial_and_all_about_the_timers)
|          0x0000005e      cf92           TODO,push r12            ; push register on stack
|          0x00000060      df92           TODO,push r13            ; push register on stack
|          0x00000062      ef92           TODO,push r14            ; push register on stack
|          0x00000064      ff92           TODO,push r15            ; push register on stack
|          0x00000066      cf93           TODO,push r28            ; push register on stack
|          0x00000068      df93           TODO,push r29            ; push register on stack
|          0x0000006a      cdb7           TODO,in r28, 0x3d        ; IO SPL: Stack lower bits SP0-SP7
|          0x0000006c      deb7           TODO,in r29, 0x3e        ; IO SPH: Stack higher bits SP8-SP10
|          0x0000006e      2897           TODO,sbiw r28, 0x08      ; substract immediate from word
|          0x00000070      0fb6           TODO,in r0, 0x3f         ; IO SREG: flags
|          0x00000072      f894           TODO,cli                 ; clear global interrupt flag
|          0x00000074      debf           TODO,out 0x3e, r29       ; IO SPH: Stack higher bits SP8-SP10
```

OR JUST TROLL AROUND

1. Fork https://github.com/Parrot-Developers/ardrone-opensource
2. Pullrequest a real opensource clone of program.elf and flash_blc.hex.
3. Hint: https://github.com/cbarox/ardrone. No AVR OSS part yet though.



README.md Initial commit a year ago

README.md

ardrone-opensource

Open Source Software used in Parrot AR-Drones

FUTURE WORK

1. Determine the meaning of the EEPROM contents.
2. ESIL support for AVR!!!
3. Port F# radare script from the AVR Russians.
4. Share binaries with encrypted project files, integrate with radare-cloud.
5. Fix/try more (working!) simulators like simavr_emscripten and javr.

QUESTIONS?