Inside Datatrak

Resurrecting a LF radio navigation system

Phil Pemberton
philpem@gmail.com - www.philpem.me.uk - M0OFX

Electromagnetic Field 2022

A few quick words...

There is a lot more I could say
(especially about how the LF nav network works) ...
... but I only have 30 minutes.

I’m mostly going to talk about how I reverse-engineered the hardware and software.

If you want to ask questions or learn more... find me or message me!
(I’ll put my details up again at the end)

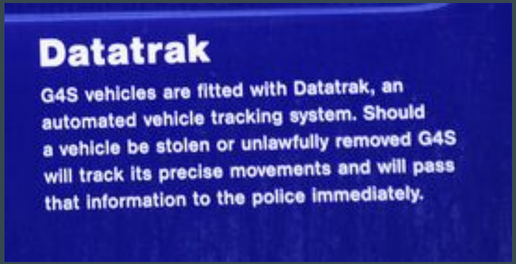
(if you worked on or used Datatrak, I’d love to hear from you!)

/usr/bin/whoami

- `$dayjob = “Embedded software engineer”;`
- Electronics and software reverse engineering hobbyist
  - Created FreeBee (AT&T UNIX PC emulator) and DiscFerret (magnetic disc archiving tools)
  - Reverse-engineered the VideoCrypt ‘token card’ PPV scheme for HackTV
  - Using fault-injection attacks on MCUs to read them out
- Amateur radio enthusiast (73’s de M0OFX)
  - Mainly digital modes
  - Especially old commercial modes like POCSAG, FLEX, Mobitex
- Also: metalworking/machining, prop building, retrocomputing

What ~~is~~ was Datatrak?

What was Datatrak?



Datatrak

G4S vehicles are fitted with Datatrak, an automated vehicle tracking system. Should a vehicle be stolen or unlawfully removed G4S will track its precise movements and will pass that information to the police immediately.

- A land-based longwave radio navigation system
  - Uses transmitters on earth - not satellites
  - A bit like LORAN-C or Decca Navigator System
- Developed by Securicor to track movements of armoured vans and deter theft
- First commercial *Locator* unit released 1988
  - LF navigation with UHF return channel
- Later public release allowed for fleet tracking and stolen vehicle recovery
  - Stolen vehicle tracking (TrakBak), especially high-value vehicles (Ferrari, ...)
- LF+UHF net operated in UK until Oct 2011,
  - Overseas networks operated until around 2014 (notably Argentina)
- Current receivers (Mk5 and later): GPS + GSM, no LF interface

Datatrak Mk.II Locator

- I found it on ebay...
- Seller had more... I bought those too.
- Initial plan: convert for ham use (PSK/LF)

- They were "navigation-only" units
  - Some had UHF transmitter/RF Amp PCBs
  - None had UHF Radio EPROMs (boo)

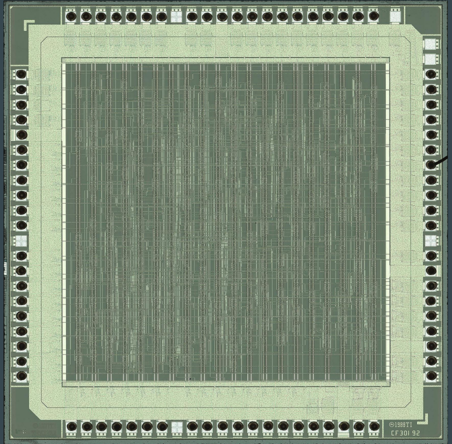
- Found an antenna and a TrakBak (LF+UHF) unit later
  - This had a UHF TX and a Radio EPROM!
  - Also different firmware

Locator Mk.II hardware specs

- 68HC000 main CPU, 10 MHz
    - Either 128 KiB (4x256kbit) or 256 KiB (2x1Mbit) RAM, battery backed
        - 128K: half of the RAM is battery-backed.
        - 256K: all battery-backed
    - Up to 512 KiB EPROM on a plug-in card - V7.1.1.4 firmware is only 256 KiB
- 80C31 co-processor CPU, 12 MHz
    - Controls the radio interface - links the 68000 to the UHF radio network
    - Up to 32 KiB EPROM, fixed 8KiB RAM
- Two RS232 serial ports
    - 15-pin opto-isolated GPIO port
- Custom ASIC

The Datatrak ASIC

- Datatrak P/N “S0089-1”, TI P/N CF30192, “TGCX03 © 1987”
  - TI TGC103 custom gate array
- 68000 glue logic:
  - DTACK, interrupt priority/ack, watchdog, address decode
- LF receiver
  - Local osc: 100kHz fixed and switchable 13.227k or 26.455k (div. 756 or 1512)
    - Mixed in SA602 for 113.227kHz or 126.455kHz IF => low phase noise
  - Phase measure: 20kHz IF vs. 20MHz TCXO (1/1000th cycle resolution)
- I/O:
  - SPI bus (EEPROM), 4x optocoupler inputs
- Decap and die shots (credit: Chris Gerlinsky): https://siliconpr0n.org/archive/doku.php?id=gerlinsky:datatrak:s0089-1



Reverse engineering the Locator

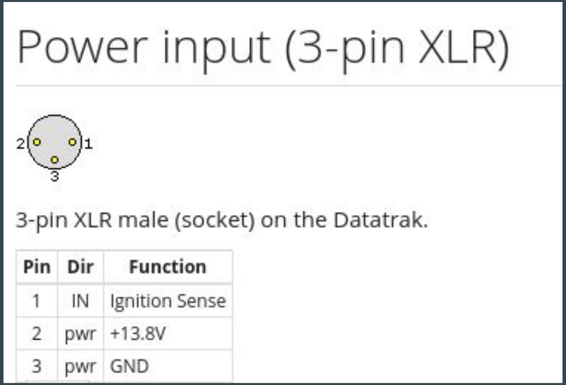
Plan of attack

- Figure out how to power it up (and talk to it)
- Learn more about it
  - Find technical papers
  - Post on forums, ask for help
  - Find people who worked on it
- Reverse engineer the firmware - aim to figure out the hardware
  - Side hustle: build an EPROM emulator and 68000 sniffer pod
- Reverse engineer the hardware (find the main peripherals)
- Reverse engineer the PCB into a schematic
- Write an emulator
- Bonus round: build a signal generator

Does it even work?

- Find the serial port pinout
  - Trace back to the MAX232's RX/TX and GND pins
- Power pinout
  - Find ground on the XLR (multimeter again!)
  - Remaining two pins must be +12V
  - The internal wiring is colour coded:
    - Red for +12V, white for ignition switch
- Power on, watch the serial port.

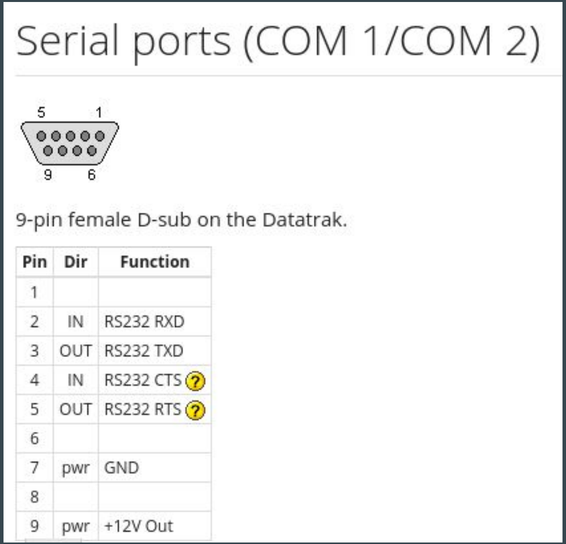
```
*** Interlaced ***
MKII address:000 group:0
V7.1.1.4 27/10/93 15:23:07 bsg (DV4435)
ROM: 178108(0x2b7bc)+16392(0x4008) = 194500(74.19%)
rem: 67644 (128k+63428)
BB-RAM: ok msg:11 chist:11 rcom:11
lp: 520000 190000
R0set patterns: 21 31 42 52 82 43 63 87
```



Power input (3-pin XLR)

3-pin XLR male (socket) on the Datatrak.

Pin	Dir	Function
1	IN	Ignition Sense
2	pwr	+13.8V
3	pwr	GND



Serial ports (COM 1/COM 2)

9-pin female D-sub on the Datatrak.

Pin	Dir	Function
1		
2	IN	RS232 RXD
3	OUT	RS232 TXD
4	IN	RS232 CTS ?
5	OUT	RS232 RTS ?
6		
7	pwr	GND
8		
9	pwr	+12V Out

Huh, there’s a command line...

Ask for help...

```
>
Eh?

> help
help: removed to save space !!
```

Time to do some research.

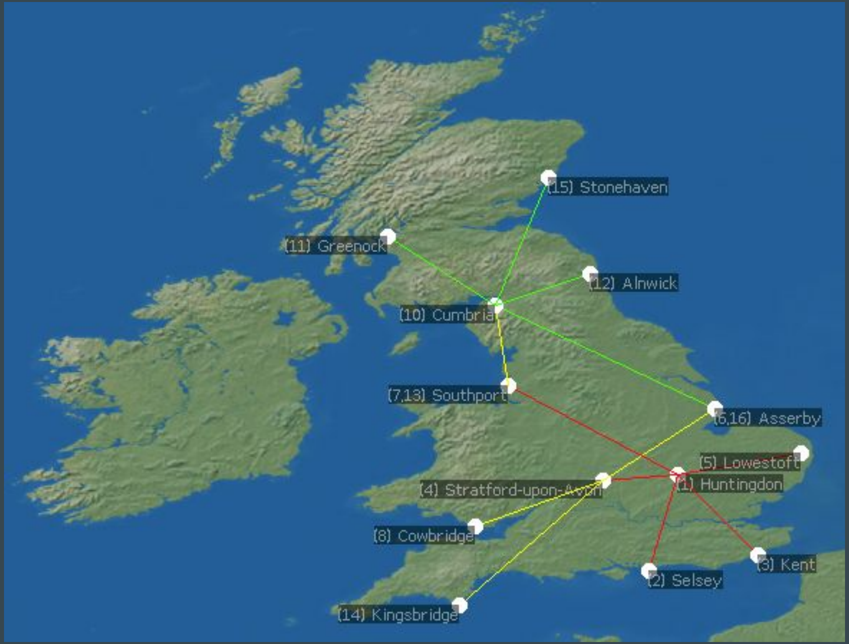
A bit of research

- Patents and journal articles
    - A few patents relevant to the antenna, but not much on the nav system
    - Papers by K. M. Banks, A. G. Scorer and Prof. David Last go into a lot of detail of the LF system]
- Contacted the authors of the papers
    - Got a bit more information which wasn't in the papers
- Post online (forums, mailing lists...)
    - Asked for signal recordings or info on the system
    - Received some old LF recordings from ham SDR RXes, and a few random documents
- Result: I have a good idea what the receiver wants to see on the RF In
- But... nobody had any hardware information (I expected this)

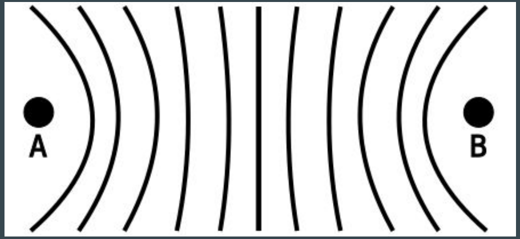
The Datatrak navigation signal and network

The LF navigation network

- 13 transmitters (of a maximum 24)
- Covered 95% of the road area of Great Britain
  - Blackspots: Land's End, Scottish Highlands
- Typical accuracy of 50m or better
- Worst-case accuracy 100m.



Hyperbolic radio navigation in a nutshell

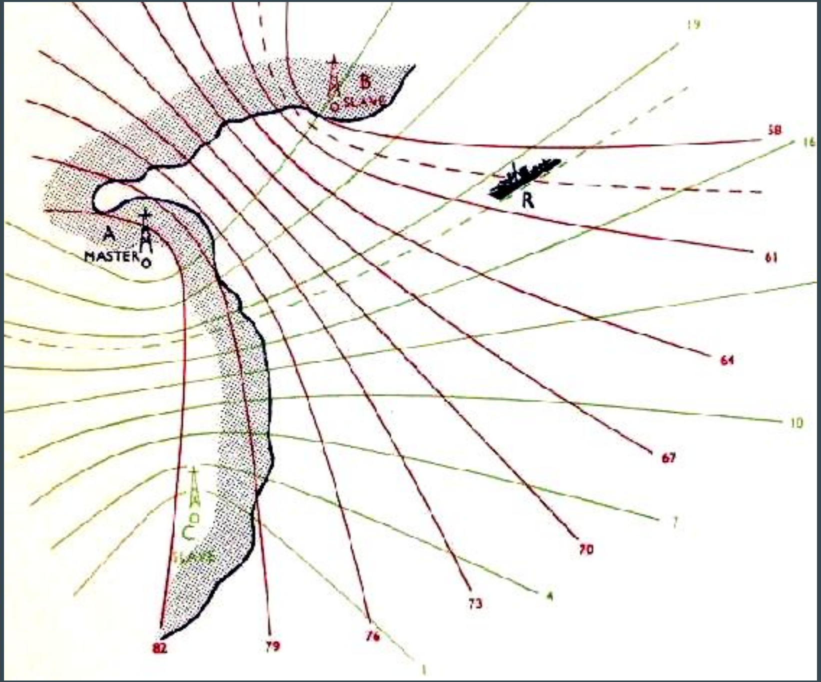


- Primary station “A” transmits a signal
- Secondary station “B” receives A’s signal, transmits its own, phase-synchronised to A’s
- Receiver measures the difference in time of arrival between the two signal
- Repeat or another pair of transmitters to get another set of hyperbolic lines of position
- Intersection of two LOPs gives the position of the receiver
- Intersection of three gives a “cocked hat” triangle containing the receiver location (with some error)

Absolute time TDOA needs accurate clocks in TX+RX (expensive).

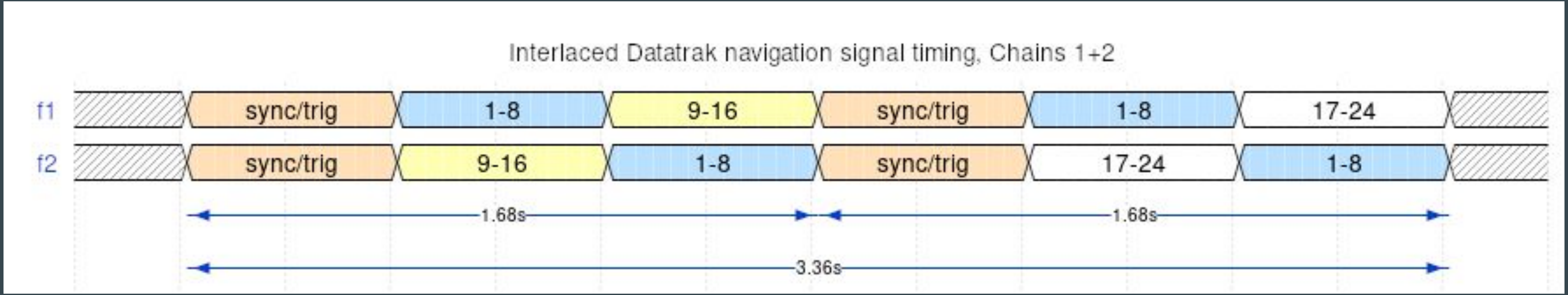
Phase-difference only needs atomic clocks in TX’s.

Problem: the same phase measurement appears every wavelength (one lane) = lane ambiguity



Datatrak signal basics

- One *cycle* = 1.68 seconds
- Maximum 24 location transmitters (in Interlaced mode)
  - 8 slots with updates every 1.68 seconds - sent every Cycle
  - Further 16 with updates every 3.36 seconds
- 64 cycles per *clock unit*
- 65,536 *clock units* before the counters wrap around
- Up to 24 *navigation slots*
  - Slots 1-8 update every cycle, slots 9-24 update every two cycles.



Why Datatrak is special

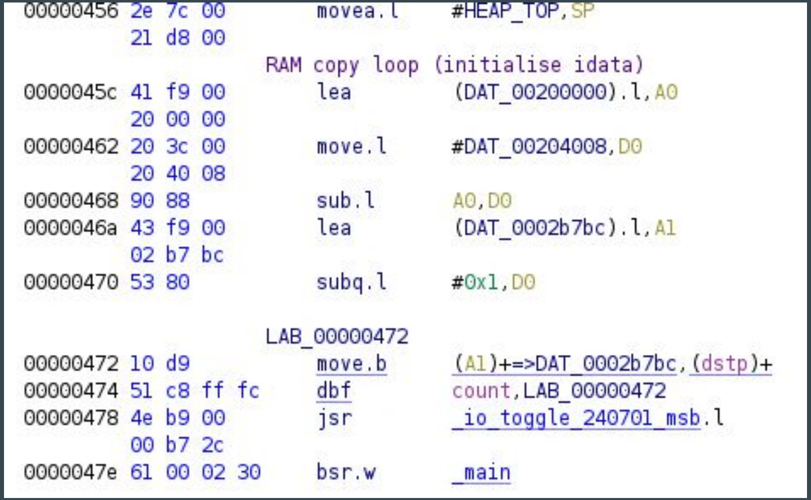
- Only needs two frequencies (F1 and F2, 10% apart) for all the transmitters
- Interleaving increases 8 slots to 24
    - Even more spectrum efficiency, at the cost of having to deal with *slot collisions* (can't RX two slots where both have the same modulo-8 value)
- Automatic lane-ambiguity resolution
    - Each Nav slot contains +40Hz and -40Hz signals
    - Subtract Fn+ from Fn- => equivalent to a nav signal sent at a 80Hz frequency
    - 2000km lane width, but low accuracy
    - Can do the same with F1 and F2 => equivalent to ~200km lane width
- Station and Vehicle data channels on the LF downlink
    - Station: messages between transmitters (e.g. on/off, test). Also carries Almanac.
    - Vehicle: messages to individual Locators or groups of them (e.g. Alarm Reset, Send Location)

Firmware reverse-engineering

Firmware reversing: finding RAM and ROM

- Read out the EPROMs
- Disassemble with Ghidra (and fix some bugs in Ghidra's 68K disassembler)
- But we don't know where the memory is!
- … or do we?
    - 68000 reset vectors are always at address 0:
      ∴ <u>ROM must be at 0x000000</u>
    - 2x 128 KiB EPROMs = 256 KiB:
      ∴ <u>ROM extends to at least 0x040000</u>

    - Reset code (on the right) initialises RAM:
      ∴ <u>RAM must be mapped in at 0x200000</u>
    - RAM is 4x TC55257B or 2x TC551001A
      <u>32 KiB x 4 = 128 KiB</u>
      <u>128 KiB x 2 = 256 KiB</u>



```
00000456 2e 7c 00         movea.l    #HEAP_TOP,SP
         21 d8 00
                       RAM copy loop (initialise idata)
0000045c 41 f9 00         lea        (DAT_00200000).l,A0
         20 00 00
00000462 20 3c 00         move.l     #DAT_00204008,D0
         20 40 08
00000468 90 88            sub.l      A0,D0
0000046a 43 f9 00         lea        (DAT_0002b7bc).l,A1
         02 b7 bc
00000470 53 80            subq.l     #0x1,D0

                       LAB_00000472
00000472 10 d9            move.b     (A1)+=>DAT_0002b7bc,(dstp)+
00000474 51 c8 ff fc      dbf        count,LAB_00000472
00000478 4e b9 00         jsr        _io_toggle_240701_msb.l
         00 b7 2c
0000047e 61 00 02 30      bsr.w      _main
```

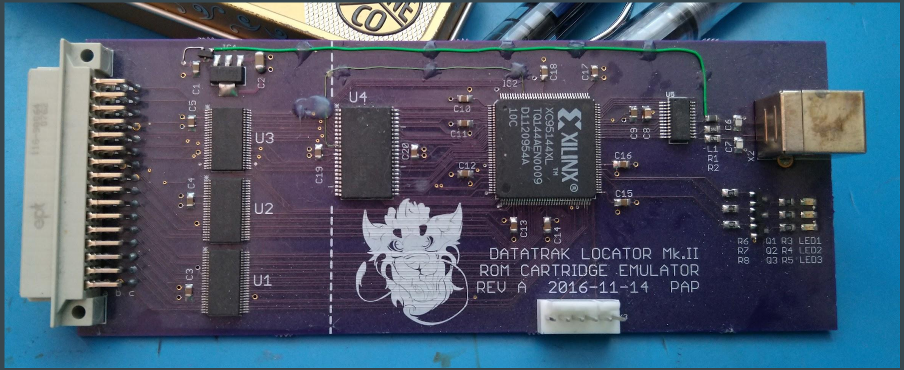
Locator firmware: what I learned

- Text strings from debug message provide *a lot* of information
  - Function names! `printf(__FUNC__ “(): error”) => “msir(): error”`
- Ghidra has problems with jump tables on M68K - issues with switch/case statements
  - Fails to identify the maximum bound of the jump table: listing fills with “`switchX_caseN`” annotations
- Custom RTOS (but a little bit UNIX inspired)
  - Features: threads, mutexes, semaphores, message queues
  - Uses 68K supervisor mode: OS calls done with `TRAP #14` and `TRAP #15`
  - Character device support (console = COM1, brother = COM2, nulld = bit-bucket, ...)
  - Memory allocation is done with `sbrk()`
- Task switches are synchronised to the incoming LF signal
  - Signal is phase modulated - remember the ASIC measures 20kHz IF phase vs. 20MHz TCXO
  - Interrupt for every phase measurement. 1kHz rate, 0.36-degree (“milli cycle”) resolution
- Signal processing and navigation are split across several threads
  - Names are only 3 characters - hard to tell what’s doing what
- Hard to follow hardware code, it’s all bit flips

Observing the hardware

Running my own code: EPROM emulator

- Mk2 Locator has firmware on a plug-in “B7415” ROM card
  - 16-bit data bus, 18-bit address bus (256K x 16-bit = 512 KiB)
  - ROM cartridge has VPP and PGM connections - Locator leaves these disconnected (read-only)
- Used 16-bit-capable (or two 8-bit) EPROM emulators are expensive
- Designed my own!
  - FTDI DT240X USB FIFO, Xilinx XC95144XL CPLD, AS6C4016 256Kx16 SRAM. All runs at 3.3V.
  - 3x level translators to convert 5V Locator logic ⇔ 3.3V emulator logic
- Includes debug “print” port
  - No write pin!
  - Steal 256 bytes from top of ROM
  - Reading one of these bytes makes the CPLD send a byte to the PC
- Also have a PC -> Locator data port
  - Status port (Data Ready bit) and a Data port



Watching the code run: Sniffer Probe

- Tools of the trade:
  - HP 16700A Logic Analyser Mainframe
  - HP 16717A State and Timing Analyser
  - Homebrew 68000 breakout pod (HP E2477AA clone) + 4x HP transition adapters
- Watch the CPU data/address/control bus.
- Inverse assembler allows us to watch instructions being executed

Putting this together: figuring out the hardware

- Find I/O device addresses
  - Trigger on chip select or pin state change event, get address from the bus
  - Trigger on interrupt, get vector number and IPL (priority) from the bus
- What if IRQ pin not accessible? (ASIC)
  - Trigger "processor state: interrupt"
  - See what hardware is being accessed afterwards (inverse assembly)
  - Get the program counter value and look at that code in Ghidra
- Result: mostly complete hardware register map, with a few gaps

Emulation

A Locator emulator: why?

- Prove I understand the hardware
- Can feed in any (emulated) RF signal: no need for signal generator
- Deterministic and repeatable: same input = same output
- Easy to add ‘traps’ in the emulator to monitor behaviour
  - When PC = (value), hex-dump (N bytes from address X)
  - Follow data through the system
    - We know where the Phase IRQ stores data - we can see what reads and processes it
  - Display the task currently being executed (RTOS awareness)
- Generally much easier than configuring the logic analyser, and more targeted

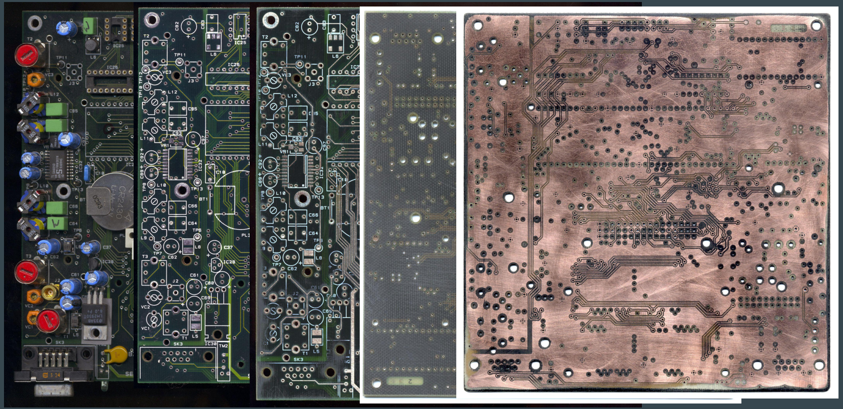
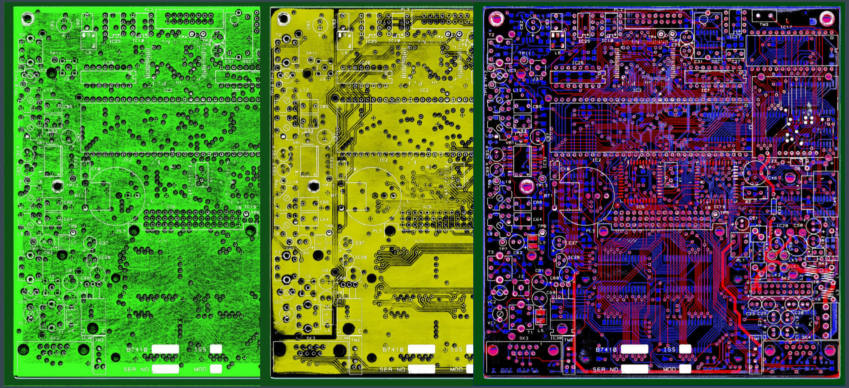
Emulator ins and outs

- Musashi 68000 CPU emulator (as used in FreeBee)
- DIY, quite inaccurate 68681 UART emulation
    - Pretends to be a 68681 with zero transmit time and an infinite buffer
    - No receive (yet)
- Serial ports exposed as two TCP ports (port 10000 and 10001)
    - Netcat listens for connections, emulator connects to localhost
- RF phase data generated internally
- On Github: https://github.com/philpem/datatrak-emulator
- Works well enough to run the firmware and start the RF lock sequence

Reverse engineering the PCB

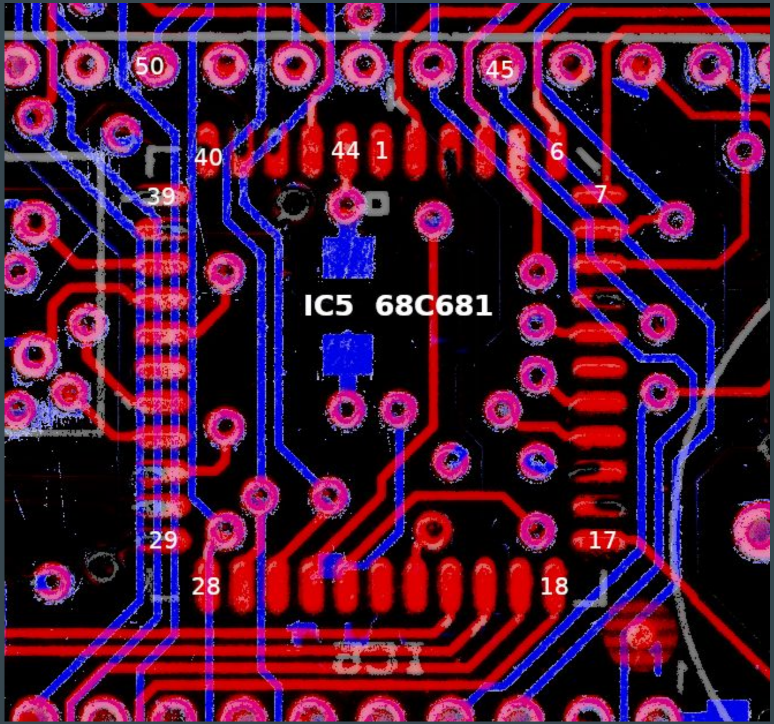
PCB reverse engineering

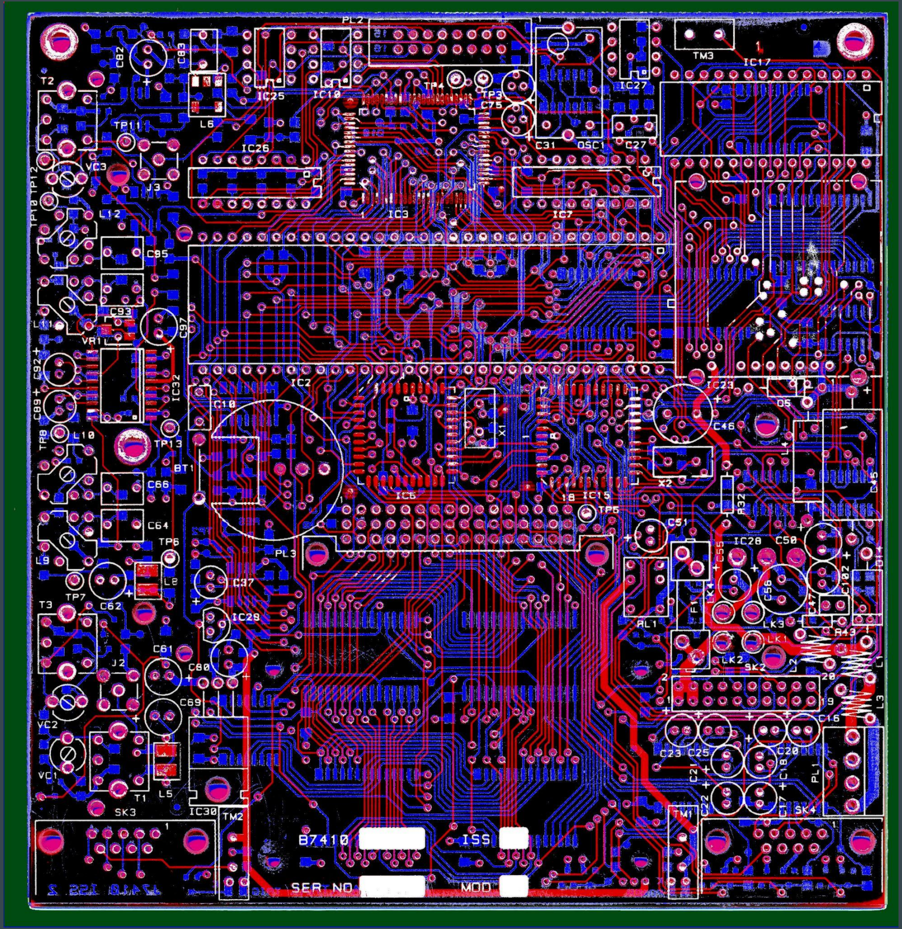
- Desolder components
- Sand 'n' Scan
  - Scan outer layers (silkscreen + solder mask)
  - Sand off silkscreen and solder mask
  - Scan outer copper layers (copper layers 1 + 4)
  - Sand down to the inner layers (takes a long time!)
  - Scan the inner layers (copper layers 2 + 3)
    Use boot polish or paint to increase contrast
- Stack layers in GIMP or Photoshop
  - Scale and rotate to align everything
  - Levels/Curves to improve contrast
  - Change colour of copper layers
  - Make fibreglass substrate (no copper) transparent
- Follow the tracks, create a netlist
- Draw a schematic (or enter it into Kicad)
- Optional: create a PCB
  - Overlay on layer scans to check your work

PCB reverse engineering hints

- Create schematic symbols for ICs
  - Only place components on the schematic as you need them
- Add pin numbers to the track scans (one layer per IC)
- Follow track - enter into Kicad - erase.
  - Use Layer Masks: overpaint the track in black to hide it
  - “Disable Layer Mask” to show all the tracks (to check your work or find your bearings)
  - Leave pads and vias in place: eases pin counting and following tracks on inner layers
- Extra credit: lay out the PCB in Kicad
  - Good as a cross-check: export PNG then overlay on the track scans.
  - Mostly useful if you want to make new PCBs.




B7410
ISS
SER NO
MOD

Current status and next steps

Where am I now?

- Partial schematic diagram (WIP) of the Mk.II PCB
- Most of the hardware registers are known
- 68000 ⇔ 8031 interface method is known, but not 68k-side addresses
- Partially working signal generator
  - Poor lock quality, only works some of the time (not sure why)
- Working emulator (only emulates the 68000)
  - Runs at ~4x real time: good for testing RF signal generation
  - Also has poor lock quality on Trigger: need to look at signal generation code

Where next?

- Decode the phase-modulated data in the recordings
    - Goal: recover a Clock signal and Almanac
    - Download from: https://www.philpem.me.uk/datatrak/recordings
    - Need to find a Gnuradio guru!
- Port TUTOR (aka MACSBUG) monitor to the Locator
    - Goal: run my own code, an assembler, etc. Poke the hardware and see what it does.
- Figure out why the LF signal lock quality is so poor
    - Need to watch data flow through the firmware
- Eventual goal: DIY LF RX+TX, build a Datatrak-like system in the ham LF band
    - 135.7kHz to 137.8kHz = 2.1kHz. Enough for a single-frequency system.
    - Good learning opportunity here

Ask me questions!

- Mastodon: https://digipres.club/@philpem
- Twitter: @philpem
- Web: www.philpem.me.uk
- Email: philpem@gmail.com
- Ham radio: M0OFX

Attempts at signal generation

- This didn’t quite work to plan...
- **Attempt 1**: Marconi 2022E + Arduino
  - Locator wouldn’t lock to the signal (“No trigger” error)
  - 2022E has poor spectral purity (lots of spurs) and inaccurate crystal oscillator
  - Borrowed my Racal 1991 frequency counter’s OCXO - still not good enough to achieve trigger lock
- **Attempt 2**: AD9850 DDS
  - Locked - but poor template match
    - AD9850 only has 11.25-degree phase resolution: Trigger+Clock not very close to the template
- Next plan: signal generator V2
  - AD9837 DDS + TCXO
  - Boards with AD9833 or AD9834 available off-the-shelf: 12-bit phase resolution