Reverse Engineering Embedded Hardware

Rolfe Bozier

25-Jun-2014

This work is licensed under a Creative Commons Attribution 4.0 International License.

Agenda

- Before you start…
- Background investigation
- Gaining entry
- Extracting the firmware
- Analysis
- What next?

This work is licensed under a Creative Commons Attribution 4.0 International License.

Legalities, ethics, and morals

- Is it legal?*
  - In some jurisdictions, maybe not. The DMCA in the US is intended to prevent this
  - Did you agree to a licence? If so, you probably want to read it…
  - Do you actually own the product?
  - Are you violating IP laws?
    - Copyright? Maybe if you decide to distribute some derived results
    - Trademarks and design? Once again, only if you try to copy it
    - Patents? This has nothing to do with how you obtain your information
    - Are you violating trade secrets? Sure!

* IANAL

This work is licensed under a Creative Commons Attribution 4.0 International License.

Legalities, ethics, and morals

- Is it ethical?
  - Sure, there is a rich tradition of dismantling technology to see how it works
  - As an individual there can be a lot to learn
  - You can customise a product as you wish
  - Assuming you own the product
- Is it moral?
  - That's up to you
  - What do you intend to do with the information?

This work is licensed under a Creative Commons Attribution 4.0 International License.

Some examples

- The DD-WRT project takes wireless routers (e.g. Linksys WRT45G) and replaces the firmware with their own
- The result is a much more capable and configurable router
- Or an opportunity for you to make it do whatever you want

This work is licensed under a Creative Commons Attribution 4.0 International License.

Some examples

- The Canon Hack Developers Kit (CHDK) and Magic Lantern projects enhance the firmware on various models of Canon digital cameras
- These offer functionality well beyond what Canon envisage (or wish to provide)

This work is licensed under a Creative Commons Attribution 4.0 International License.

Some examples

- My choice of project was the Canon MG5560 printer
  - Contains CiSRA software
  - I don’t know much about the internals
  - Actually, I know next to nothing, so I’m starting from scratch
  - I didn’t rely on the few tidbits that I did know

This work is licensed under a Creative Commons Attribution 4.0 International License.

Know your opponent

- Google around for background information
  - Sometimes the manufacturer will reveal some useful details- look at the product specs
  - Maybe other people have done some investigation
  - Look at similar products
  - For older products you can often find a service manual online
- Look at the product capabilities. The MG5560 has:
  - Wireless networking
  - Scanner
  - Print from LAN and Cloud
  - Web server for local access
  - Web client for printing from Picasa, Flickr etc.
  - Updateable firmware (directly over the net)

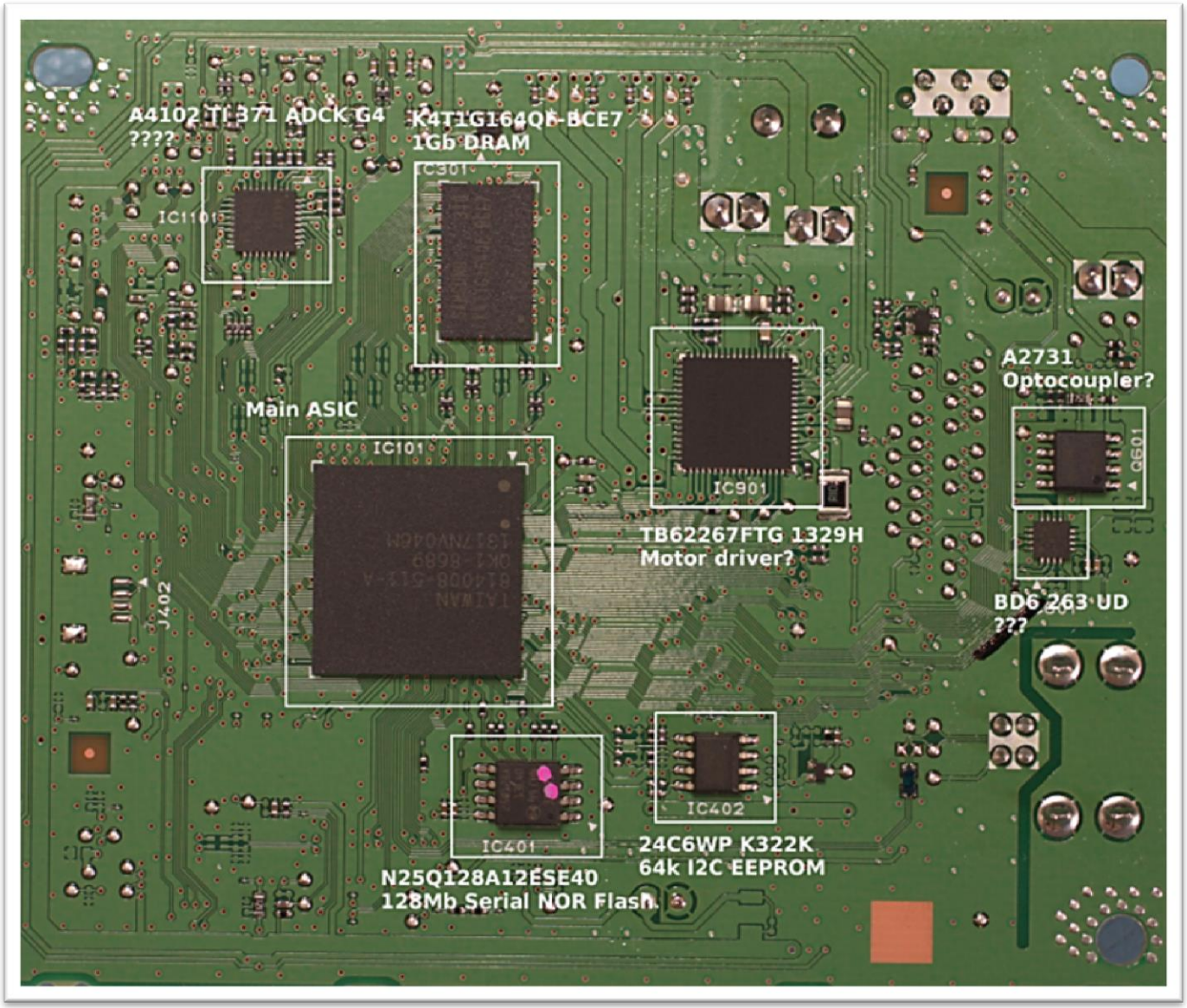
This work is licensed under a Creative Commons Attribution 4.0 International License.

Know your opponent

- Open it up and have a look at the main board[s]
  - Is there anything interesting written on the board?
  - Look up the major part numbers on the internet; you probably won't find some of them
    - Custom ASICs will have no data
    - Sometimes parts are rebranded for a company
    - Sometimes parts are created specifically for a company
    - Smaller parts often have abbreviated part codes
    - Sometimes part numbers are sanded off

This work is licensed under a Creative Commons Attribution 4.0 International License.

MG5560 main board



This work is licensed under a Creative Commons Attribution 4.0 International License.

MG5560 main board

- So… we have:
  - 128 MB SDRAM (working memory)
  - 16 MB Serial Flash (firmware)
  - 4 MB Serial Flash (more firmware?)
  - 8 KB Serial EEPROM (settings)
  - Custom ASIC (???)
  - Probably the motor controller
  - Several other unidentified ICs

This work is licensed under a Creative Commons Attribution 4.0 International License.

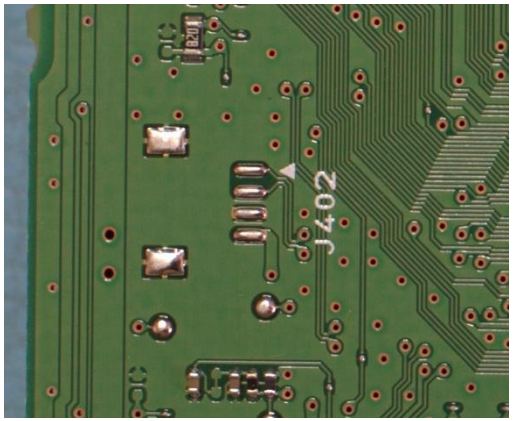
Methods of attack

- There are several different ways we could choose to attack the device:
  - Download someone else's tools (*boooring!*)
  - Try a customised firmware load
    - can be hard in the face of encrypted, signed data
    - the MG5560 gets its firmware directly over the internet
  - Extract data from the device
    - monitor the address/data lines
    - dump any ROM or Flash chips
  - Look for a JTAG or serial port on the board
  - Look for network application weaknesses
  - Look for suspicious network ports
- Companies developing embedded consumer hardware have a responsibility to make them secure
  - … but they usually don't do a very good job
  - It's the usual cost/benefit tradeoff (gamble)

This work is licensed under a Creative Commons Attribution 4.0 International License.

Looking for JTAG / Serial ports

- Generally this is pretty easy – manufacturers often need some form of access to the board for verification
- Let's look at our board



This work is licensed under a Creative Commons Attribution 4.0 International License.

Unpopulated headers

- J401/J402 are unpopulated 4-pin headers (Ground + 3.3V + 2 data lines)
  - These could possibly be serial connections
- There is another unlabeled missing connector, but it is missing associated components so it may be for a different product
- There are other test points around, maybe some combination of these provide a JTAG connection, but it would be a fair amount of work to be sure
- Connecting an oscilloscope up to the data lines and restarting the printer a couple of times revealed signals on one line
  - Looks like 19.2K 8-bit serial traffic
  - The protocol is a bit non-standard though
  - Connect via a USB/Serial bridge to my desktop, start up a terminal program and after much mucking around I get command prompt

This work is licensed under a Creative Commons Attribution 4.0 International License.

J401 serial port

```
DRYOS version 2.3, release #0049+SMP
Copyright (C) 1997-2011 by CANON Inc.
subsystem version : build_13f_130617_01
wlan available
Dry> help
[OTG]
 otg  otg_init  otg_cleanup
[Debug]
 task  sem  event  mutex  cond  mq  timer  itsk  isem  iflg  idtq  imbx  impf impl  icyc  mkobjsize  prio  release
resume  suspend  kill  delete  mkcfg objinfo  meminfo  xd  xm  cmp
[Test]
 echod  gtime  loosesock  netecho  netrace  mctalk  mktest  iotest  chkit4 time  arpsend  chkspi  pingall
setipsec
[Miscellaneous]
 date  dminfo  exit  shutdown  vers
[Network]
 arp  dnsutil  host  ifconfig  netstat  nsupdate  mbufs  ping  route  sockhalt tcputil  pngl  slaup
[deleted]
```

- DryOS is a Canon proprietary embedded OS (no information available)
- based on µITRON (some information can be found on the net)
- DryOS is also used in newer Canon DSLRs
  - so check out the CHDK & Magic Lantern webpages

This work is licensed under a Creative Commons Attribution 4.0 International License.

DryOS

- xd – memory dump, xm – memory modify
  - Playing about with xd, eventually crash it:

```
Dry> xd 45800000 20 b
          0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f  0123456789abcdef
--- Error User Exception ---
[InfoReg]:A0          0x9f946a74        A8          0x9f95cb44
[InfoReg]:A1          0x1fc979e0        A9          0x0000005f
[InfoReg]:A2          0x45800000        A10         0x0000000a
[InfoReg]:A3          0x1fa39d40        A11         0x45800000
[InfoReg]:A4          0x00000001        A12         0x1fd9c708
[InfoReg]:A5          0x00000000        A13         0x00000000
[InfoReg]:A6          0x1fc979e0        A14         0x00000010
[InfoReg]:A7          0x00000002        A15         0x00000000
[InfoReg]:PS          0x00060130        EXCAUSE     0x00000003
[InfoReg]:EPC1        0x1f95cb79        EPC2        0x00000000
[InfoReg]:WINDOWBASE  0x00000007        WINDOWSTART 0x00000080
[InfoReg]:EPC         0xc283c9fe        EPS         0x00000000
[InfoReg]:EXCSAVE1    0x1fa39d40        EXCSAVE2    0x00000000
[InfoReg]:LBEG        0x1fa1f404        LEND        0x1fa1f40e
[InfoReg]:LCOUNT      0x9fa247b5        SAR         0x00000019
[InfoReg]:IBREAKENA   0x00000000        EXCVADDR    0x45800000
[InfoReg]:DDR         0x00000000        DEBUGCAUSE  0x00000000
[InfoReg]:IBREAKA0    0x00000000        IBREAKA1    0x00000000
[InfoReg]:DBREAKA0    0x00000000        DBREAKA1    0x00000000
[InfoReg]:DEPC        0x00000000        EPS2        0x00000000
[InfoReg]:INTERRUPT   0x00000001        INTENABLE   0x00000002
[InfoReg]:CCOUNT      0xe46dccc9        ICOUNT      0x00000000
[InfoReg]:ICOUNTLEVEL 0x00000000
```

Register names – very interesting!

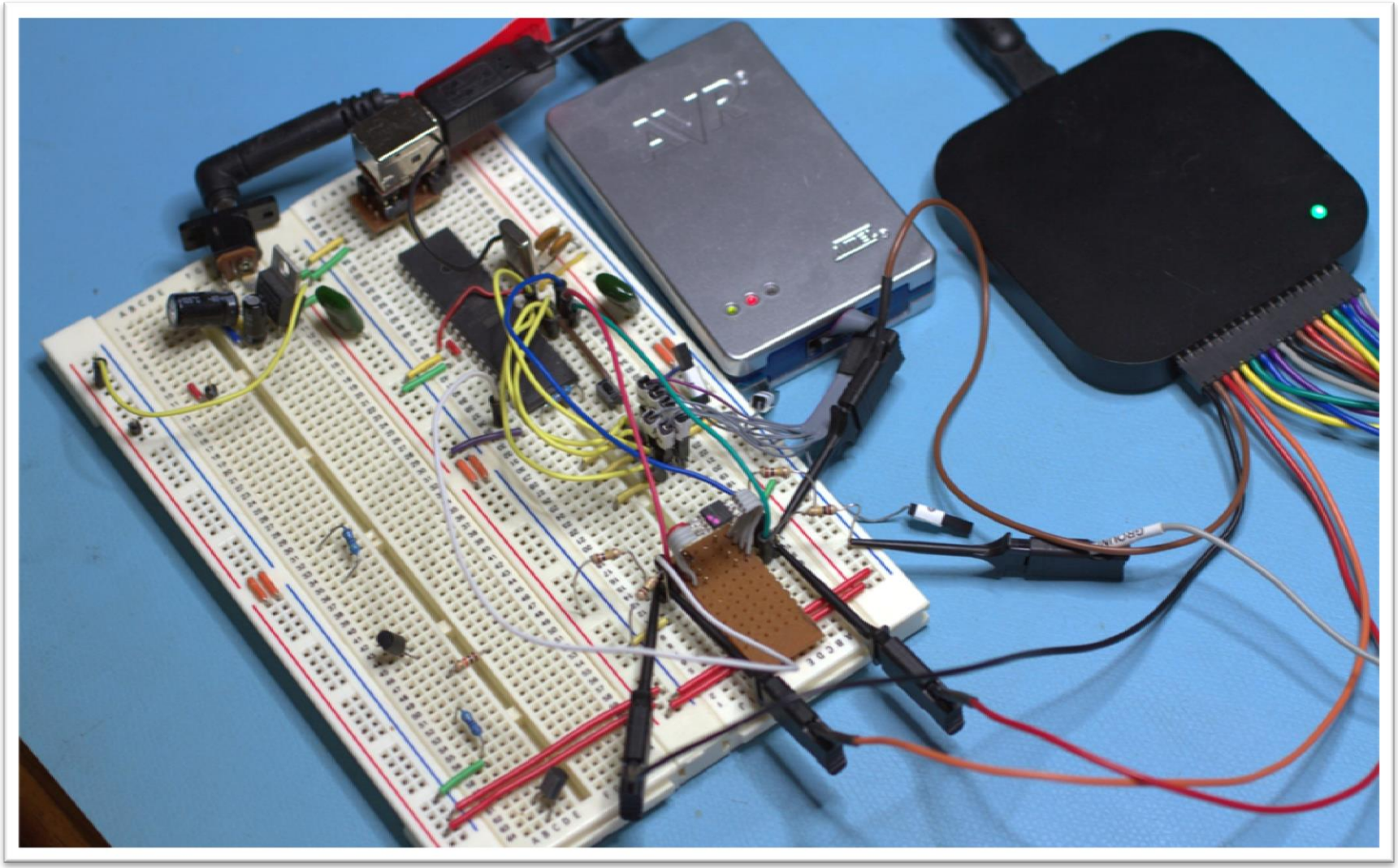
This work is licensed under a Creative Commons Attribution 4.0 International License.

Firmware dumping

- Dumping out memory using xd is interesting, but I want more data
- …what about those Serial Flash chips?
  - Data sheet is readily available, including protocols
  - Desolder from board, connect to some wires
  - Put on breadboard with an AVR microcontroller
  - Write some code to retrieve data from Flash, and dump over a serial connection to my desktop
  - Put the IC back on the board
- Now we can do some serious analysis of the firmware

This work is licensed under a Creative Commons Attribution 4.0 International License.

My setup



This work is licensed under a Creative Commons Attribution 4.0 International License.

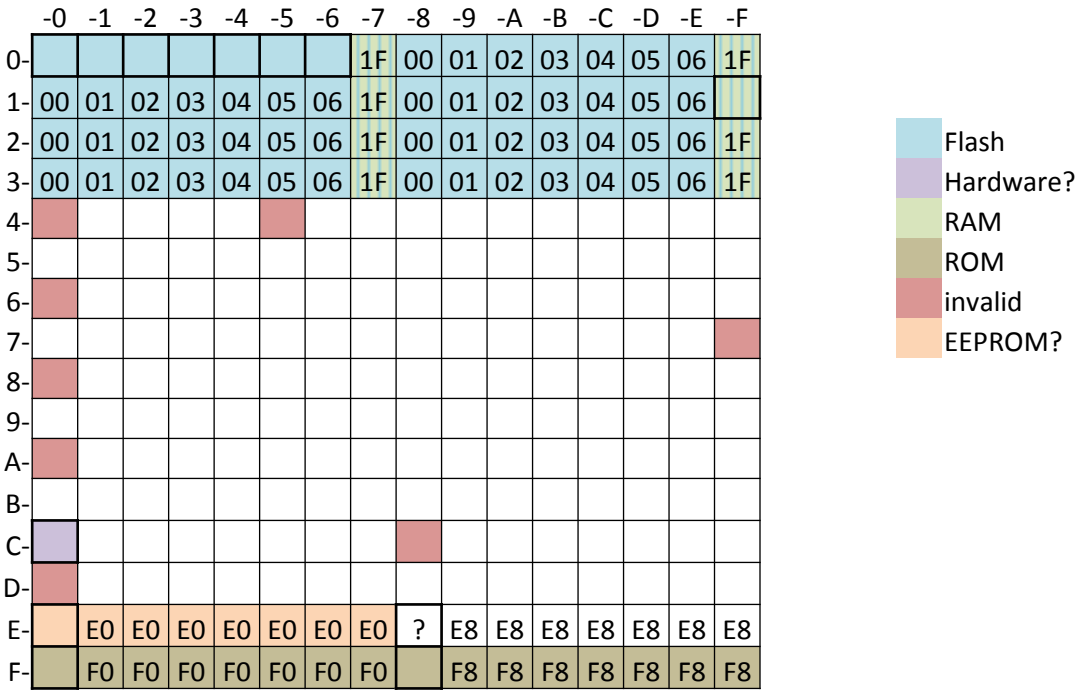
Firmware dumping

- Firmware will contain code and data (binary and text)
- Dump out all the string data – lots of info there
  - Strings from zlib code + compressed data → write a script to decompress the deflated blobs
- What about the executable code?
  - Sometimes you can tell from the hex dump
  - Repeated patterns every 4 bytes suggests 4-byte instructions (RISC, e.g. MIPS, SPARC, ARM)
  - Try googling a few dozen bytes of hex code
  - Try disassembling using different tools
  - onlinedisassembler.com lets you choose from dozens of processors
- I identified ARM code + unknown code
- Remember the register names I found?
  - They provided the clue that the other code was for a “Relax” processor
- Downloaded gcc cross-compiler tools for both architectures
- Code disassembly gives clues about relevant address spaces

This work is licensed under a Creative Commons Attribution 4.0 International License.

Constructing a memory map

- I divided the address space up into 16M chunks and started working out what data (if any) was in each
  - There is some element of trial and error



This work is licensed under a Creative Commons Attribution 4.0 International License.

Constructing a memory map

- As I built up my understanding, I would dump out 16M blocks using xd and save to disk
  - This gave me access to RAM and ROM, as well as the Flash-based data
  - So, this gives me complete access to the Relax side of the dual-CPU device
  - I've written a bunch of scripts that disassemble code and annotate functions that I've worked out
    - (I've identified about 250 functions out of 6600)

This work is licensed under a Creative Commons Attribution 4.0 International License.

Where to next?

- First, I need to unbrick the printer ☺
- It would be nice to get access to the ARM processor
  - Presumably J402 is the way in, but it doesn't respond in the same way
- Maybe introduce some non-trivial changes to the OS
- Ideal goal is to work out a way to get entry without a physical connection

This work is licensed under a Creative Commons Attribution 4.0 International License.