You Should Probably Just Write A Wireshark Dissector

(...And Then, We Should All Dream Bigger)

Joshua Wise
April 2026
MTVRE

@joshua@social.emarhavil.com

Accelerated Tech

Thesis:

- I spend a lot of time sifting through log files of structured communications

- Writing bespoke analyzers is a bad use of time!

Thesis:

Mike Primavera
@primawesome

My neighbor told me bespoke analysis tools keep parsing his event traces so I asked how many tools he has and he said he just goes to the text editor and gets a new script afterwards so I said it sounds like he's just feeding billable hours to grep and sed and then his customer started crying.

7:03 AM · Sep 30, 2019

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job

Thesis:

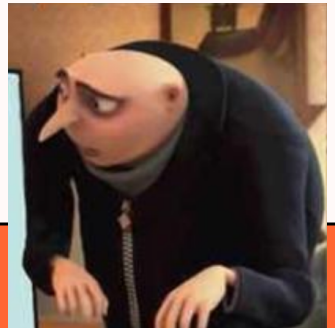
- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
    - Like Wireshark

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
  - Like Wireshark

Thesis:

- I spend a lot of time sifting through log files of structured communications
- Writing bespoke analyzers is a bad use of time!
- Therefore, if you have to write analyzer / parsers more than a few times, consider using a better tool for the job
  - Like Wireshark

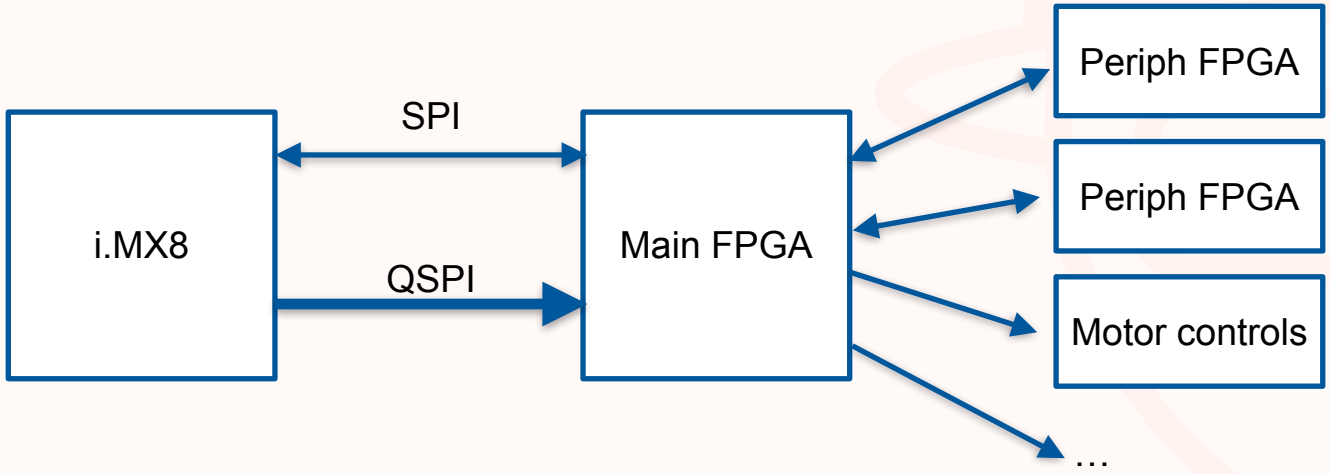


Part 1: You should probably just write a Wireshark dissector

Accelerated Tech

Problem statement

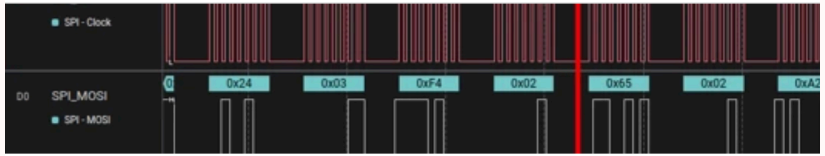
- “SNIAZ” is a connected industrial device with a few operating modes
  - OS runs on Linux SoC, peripherals controlled by an FPGA
- Users submit bug reports to me with logic analyzer traces



Problem statement

Jira

Search



Spaces / [redacted] / [redacted] / [redacted]SCRUM-454

PS [redacted] sensor value doesn't work

Key details

Priority: High

Description

In Prerun mode and in Run mode *ps_[redacted]_open* doesn't work.

Steps for reproducing:

- FPGA FWs from master branch
- [redacted] for any PS
- see response values in offset for Prerun page - 248 bytes, Run page - 80 bytes

Attachments 5

image-202408... 308.png
13 Aug 2024, 01:36 AM

Roller sensor.sal
13 Aug 2024, 01:36 AM

Assignee: JW Joshua Wise

Problem statement

Joshua Wise
August 13, 2024 at 8:33 PM

I decoded the trace that you show there. I see that the trace starts with sequence number 797, for which ps_open=(0, 0, 0); at sequence number 2834, I see that PS2 opens: ps_open=(0, 1, 0)

I checked in my decoder to the `utils` directory of the `sniaz-fpga` repository. To run it:

- [some other steps to get into a JSON trace of packets...]
- decode the trace and look for changes in the `ps_open` field: `joshua@samskara:~/work/sniaz/jira/454$ python3 ../../sniaz-fpga/utils/decode_spi_transactions_from_json_and_look_for_sensor_changes.py spi_trace.json`
- or if you want to fully decode every transaction: `joshua@samskara:~/work/sniaz/jira/454$ python3 ../../sniaz-fpga/utils/decode_spi_transactions_from_json.py spi_trace.json | less`

One obvious solution

```
from data_pages.decode_data_page import *

last_open = None

j = json.load(open(sys.argv[1], "r"))
for txn in j:
    if 'mosi' in txn and 'miso' in txn:
        miso = binascii.unhexlify(txn['miso'])
        mosi = binascii.unhexlify(txn['mosi'])
        if len(miso) == 0:
            continue
        mosi_dec = decode_data_page_request(mosi[6:])
        miso_dec = decode_data_page_response(miso[6:])

        if last_open != miso_dec.ps_open:
            print(txn['t'])
            print(f"> {mosi_dec}")
            print(f"< {miso_dec}")
            print("")
            last_open = miso_dec.ps_open
```

Refining it

- OK, but this produces a wall of text — the contents of every transaction that matches
- Hard to get what you want out of each transaction, visually

```
if type(mosi_pkt) == PreprintModePageToFpga:
    #mosi_txt = f"  cutter_seek_home {mosi_pkt.cutter_seek_home}"
    mosi_pkt.sequence_number = 0
    mosi_pkt.crc16 = 0
    mosi_txt = f"  {str(mosi_pkt)}"
if type(miso_pkt) == PreprintModePageFromFpga:
    #miso_txt = f"  git commit {miso_pkt.main_git_commit:x}, {miso_pkt.main_version_major}.{miso_pkt.main_ver
    miso_pkt.sequence_number = 0
    miso_pkt.crc16 = 0
    miso_txt = f"  {str(miso_pkt)}"
if type(mosi_pkt) == RunModePageToFpga or type(mosi_pkt) == IdleModePageToFpga:
    mosi_txt = f"  cutter_enabled {mosi_pkt.cutter_enabled}, next_cut_line {mosi_pkt.next_cut_line}, " +
               f"lines_to_append {mosi_pkt.feed_number_of_print_lines_to_append}, " +
               f"target_step_value {mosi_pkt.feed_target_step_value}"
if type(miso_pkt) == RunModePageFromFpga or type(miso_pkt) == IdleModePageFromFpga:
    miso_txt = f"  cutter_ready {miso_pkt.cutter_ready}, cut_is_done {miso_pkt.cut_is_done}, " +
               f"last_cut {miso_pkt.last_cut_line}, current_line_number {miso_pkt.current_line_number}, " +
               f"remaining_lines {miso_pkt.remaining_lines}, " +
               f"current_step_value {miso_pkt.feed_current_step_value}"
```

Refining it

- OK, but how do you know which duplicates matter, how many duplicates there are, ...?

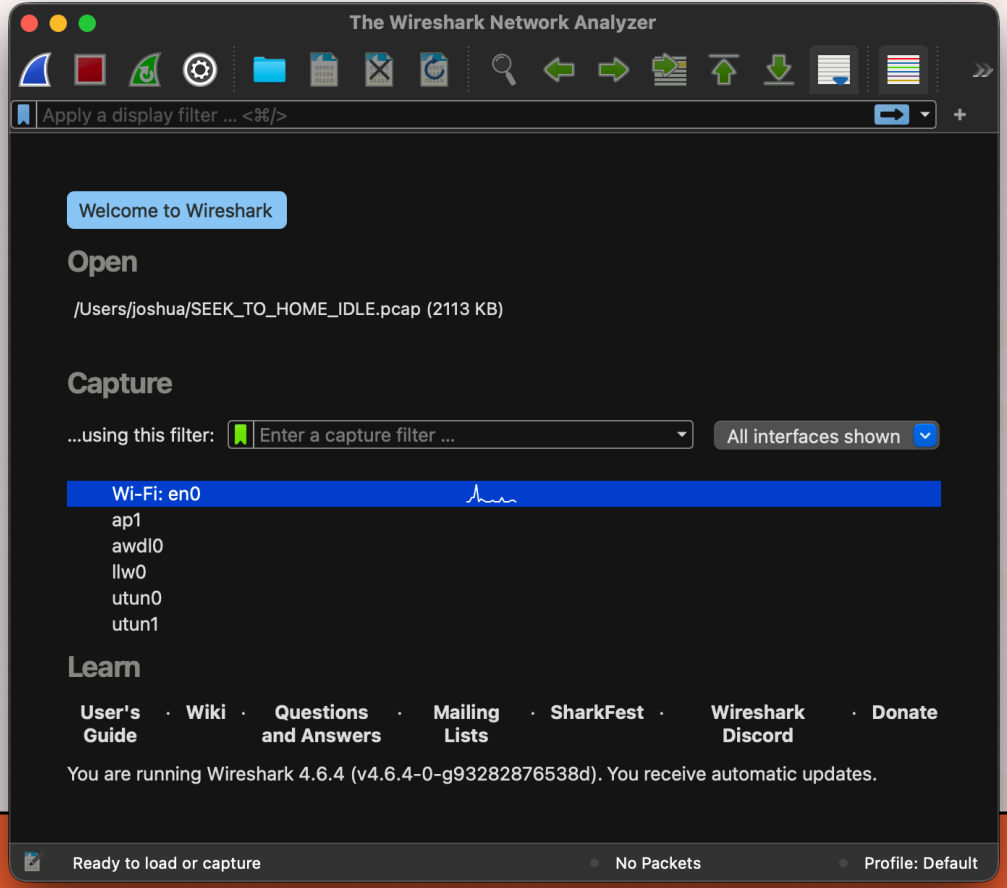
```
if mosi_txt == last_mosi_txt and miso_txt == last_miso_txt:
    n_dups += 1
else:
    if n_dups != 0:
        print(f"  [ {n_dups} duplicates ]")
        n_dups = 0
    print(f"{now * 1e-9:0.5f} {type(mosi_pkt).__name__} {type(miso_pkt).__name__}")
    print(mosi_txt)
    print(miso_txt)

last_mosi_txt, last_miso_txt = mosi_txt, miso_txt
```

Beyond refinement

```
joshua@samskara:~/work/sniaz$ ls */*.py
 main-fpga/saleae_to_vcd.py                          jira-1195/print_spi_transactions_cutter.py
'Logic Capture/print_spi_transactions_fan.py'        jira-1195/print_spi_transactions_from_trace.py
'Logic Capture/print_spi_transactions_ps.py'         jira-1195/vcd_to_trace.py
'Logic Capture/print_spi_transactions.py'            jira-500/print_spi_transactions_fan.py
'Logic Capture/saleae_to_vcd.py'                     jira-760/print_spi_transactions_fan.py
 jira-1100/print_spi_transactions_cutter.py          jira-823/print_spi_transactions_fan.py
```

Isn’t there a tool for packet-structured data?

Wait, but I thought...?



PCAP files are for everyone

https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.html

4. File Header

The File Header has the following format:

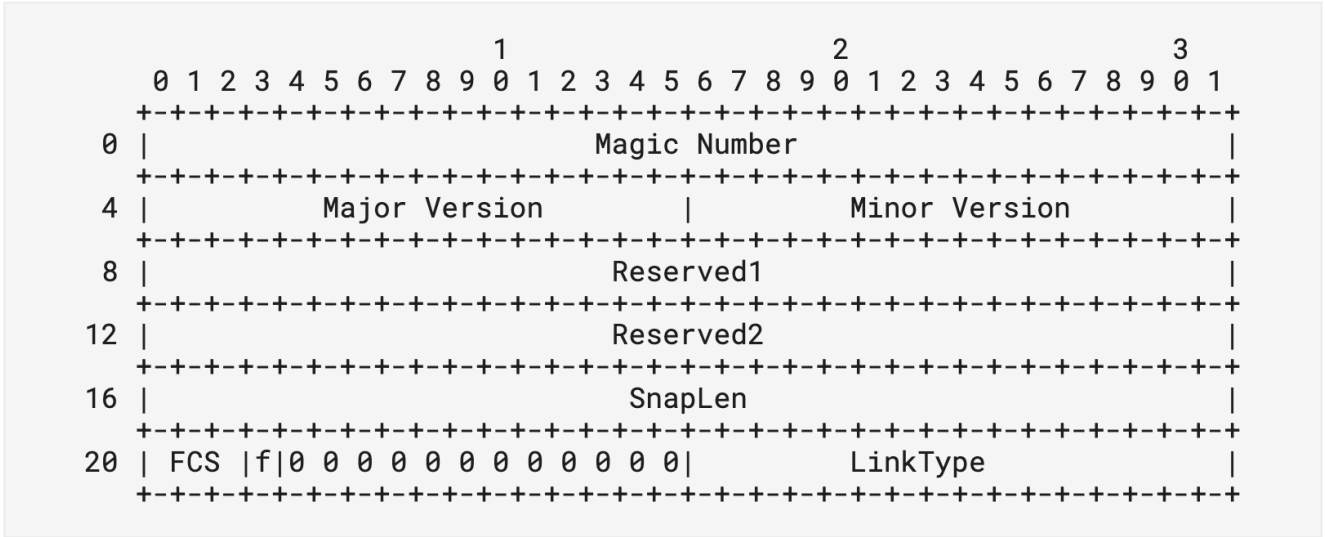


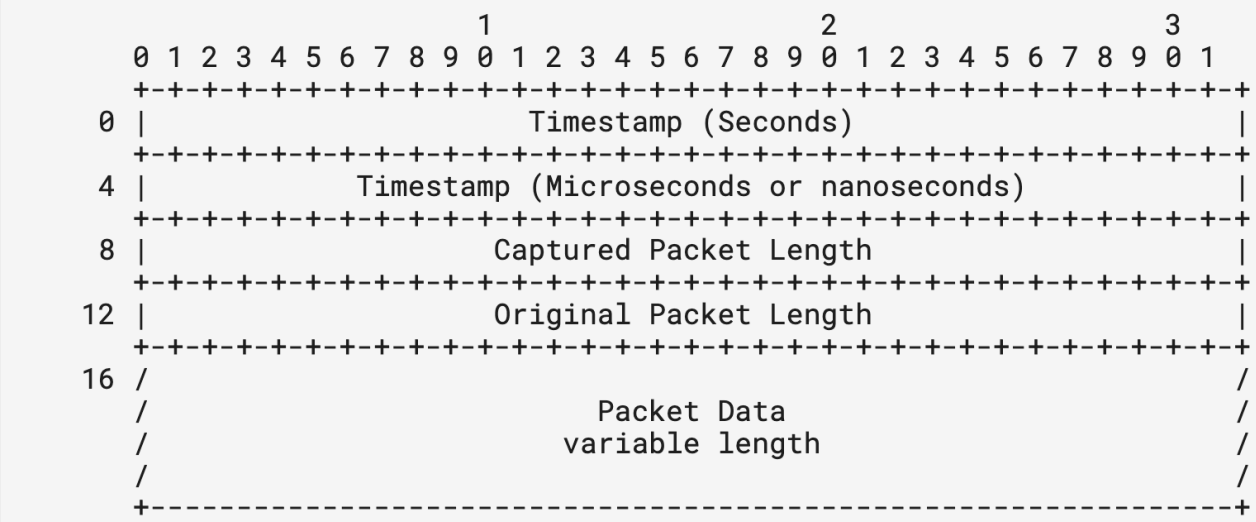
Figure 1: File Header

PCAP files are for everyone

https://www.ietf.org/archive/id/draft-gharris-opsawg-pcap-01.html

5. Packet Record

A Packet Record is the standard container for storing the packets coming from the network.

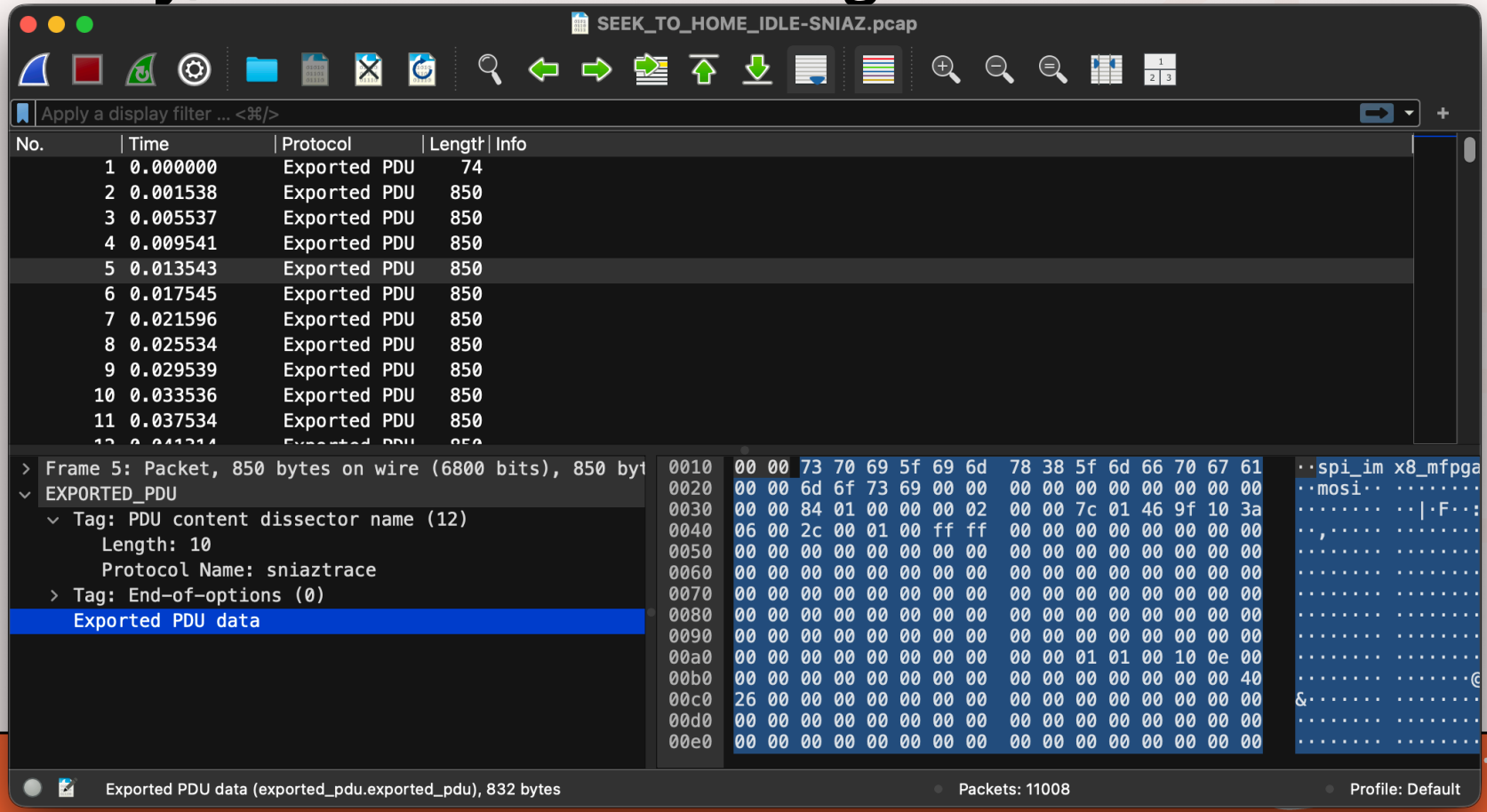


So easy, even an electrical engineer can do it

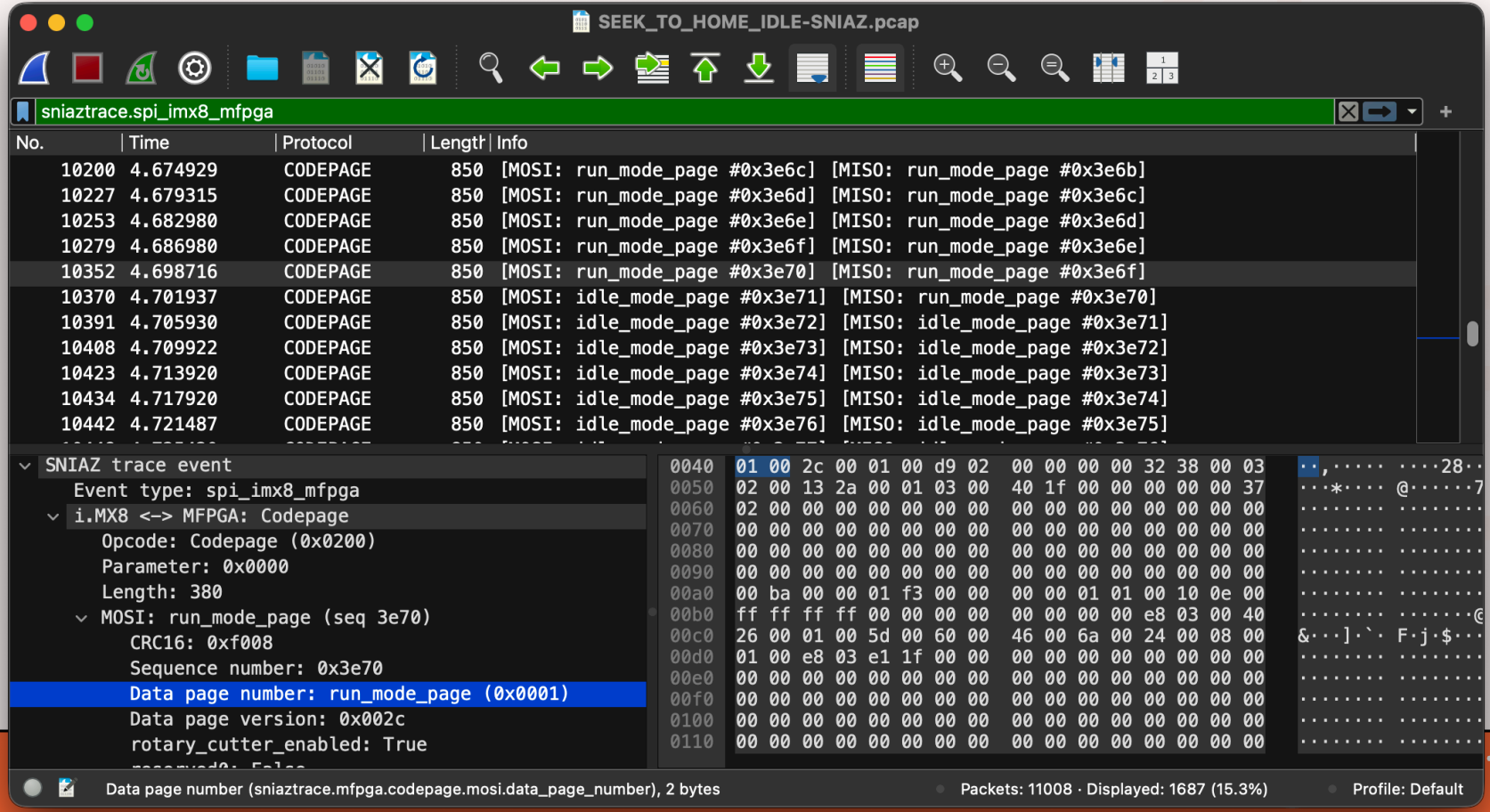
- OK, you probably want to write a “Wireshark upper PDU” packet rather than registering your own pcap datatype with the IETF

```
def write_header(f):
    # LinkType = wireshark upper PDU
    f.write(struct.pack("<LHHLLLL", 0xA1B2C3D4, 2, 4, 0, 0, 4096, 252))

def write_event(f, timestamp, event, dissector_name='sniaztrace'):
    sharkhdr = struct.pack('>HH', 12, len(dissector_name)) + \
               dissector_name.encode() # DISSECTOR_NAME
    sharkhdr += struct.pack('>HH', 0, 0) # END_OF_OPT

    buf = sharkhdr + event

    pcaphdr = struct.pack("<LLLL",
                  int(timestamp), int((timestamp % 1) * 1e6),
                  len(buf), len(buf))

    f.write(pcaphdr + buf)
```

So easy, even an electrical engineer can do it



You Should Write A Wireshark Dissector

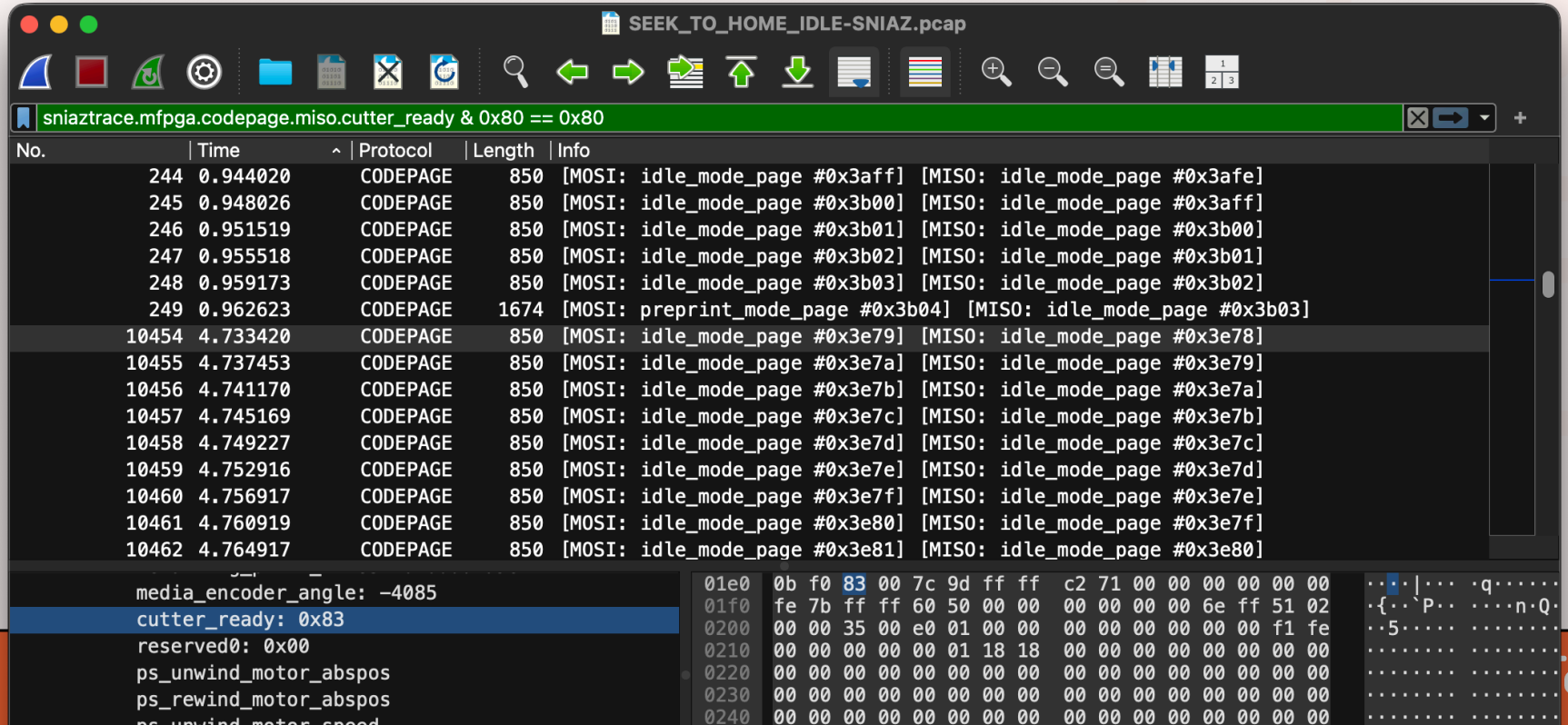


exercise for the reader

- OK writing a pcap file is easy, but your first dissector is not
- Dissectors are Lua scripts
  - That are not really written in very idiomatic Lua
  - OK, also C
- The rest of this talk does NOT discuss how to do it
  - Instead: why to do it, things to consider, ...
  - But I provide a sample
- There is a lot of documentation on it
  - Unfortunately it is not well-organized
- My best suggestion: do it once, prepare to cut and paste

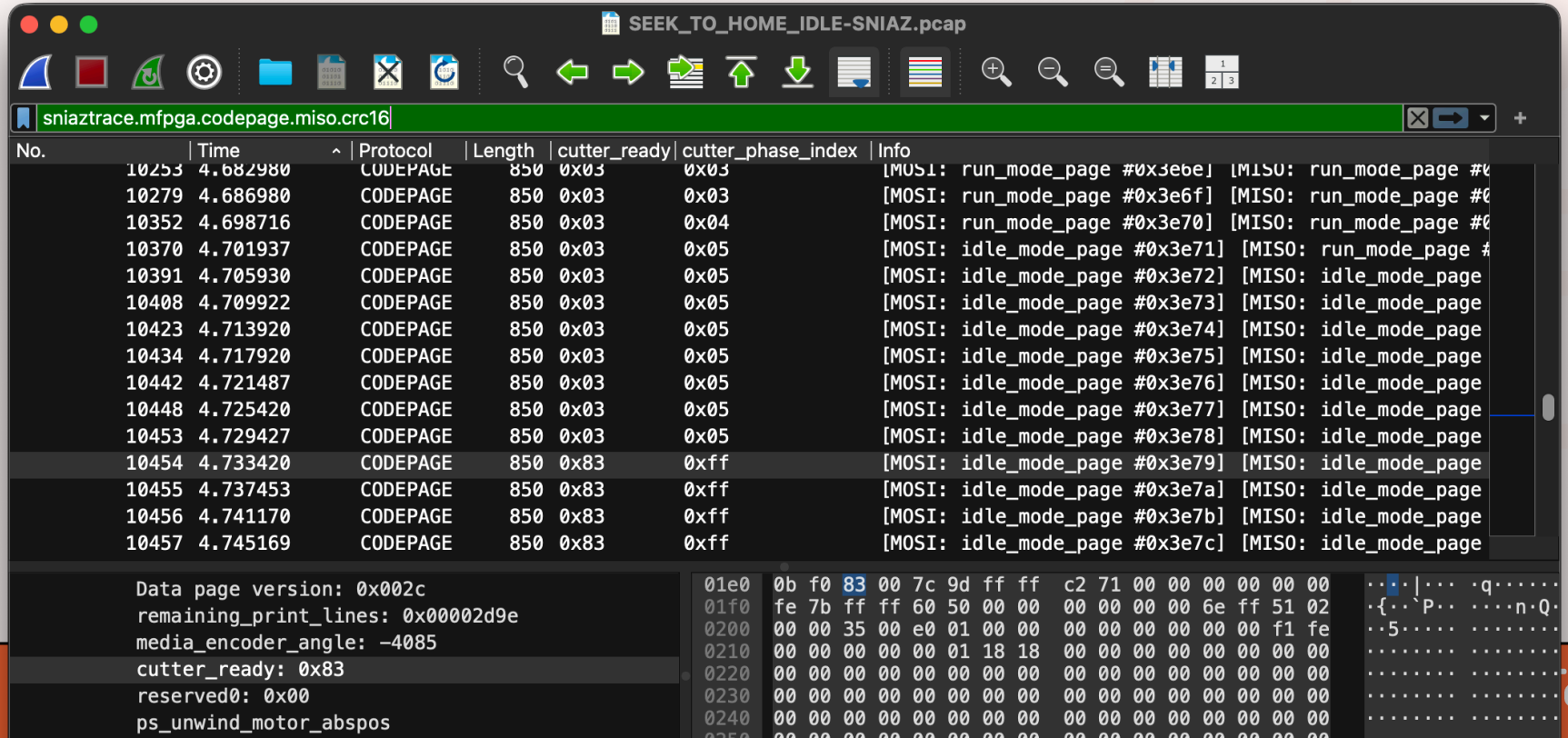
SCRUM-1541: “The cutter did not complete at idle”

- Filter *quickly* to find exactly what time it croaked at



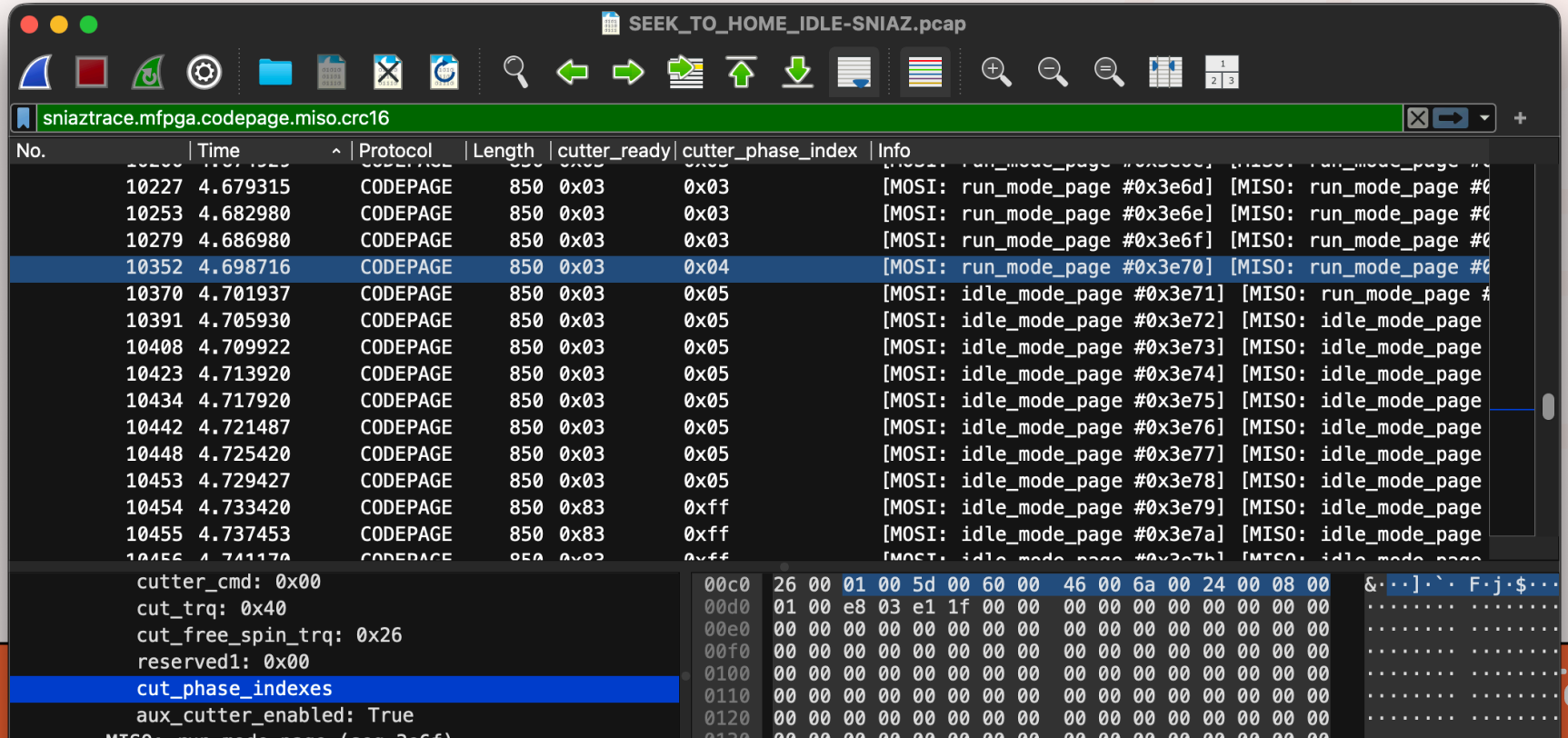
SCRUM-1541: “The cutter did not complete at idle”

- Add columns to see what it was doing at the time



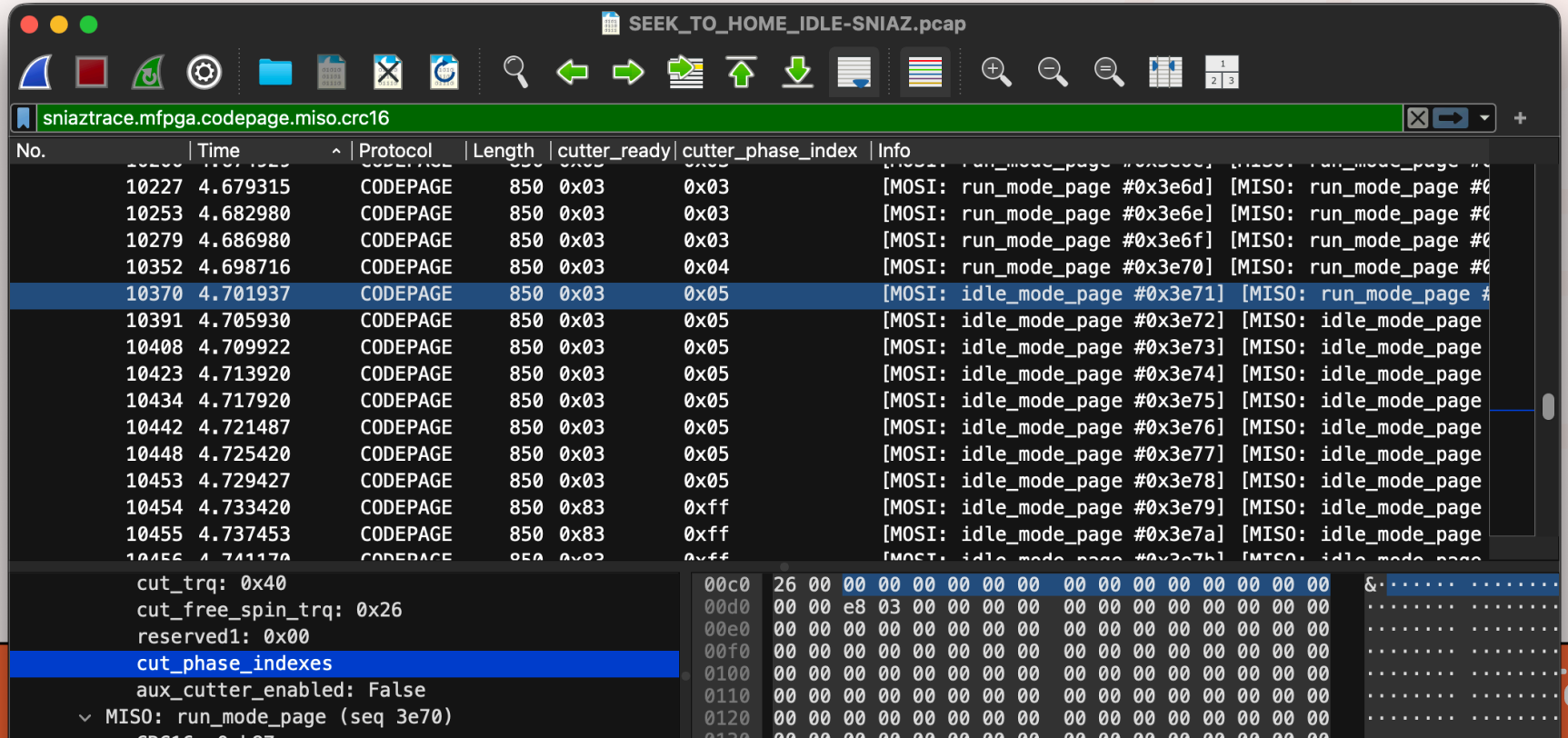
SCRUM-1541: “The cutter did not complete at idle”

- Found preceding transition to idle — compare parameters



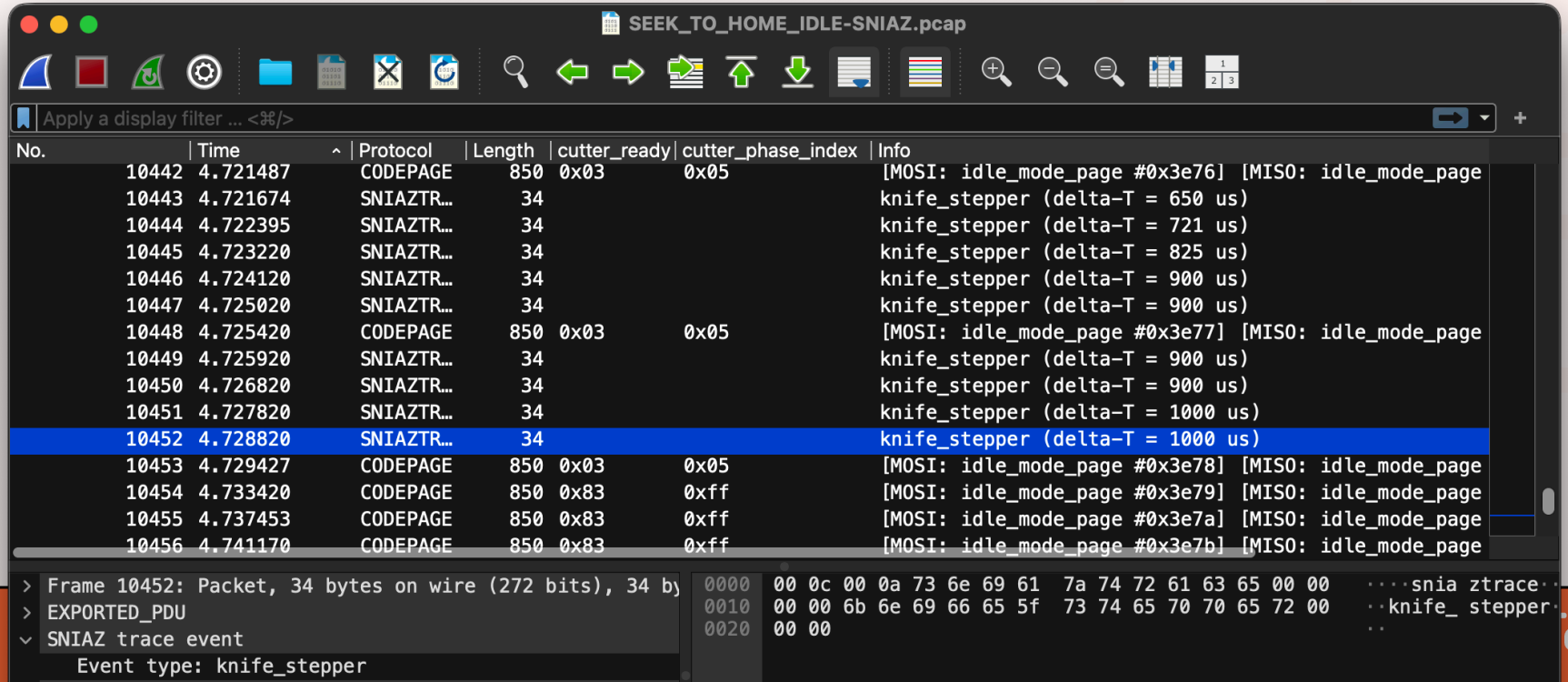
SCRUM-1541: “The cutter did not complete at idle”

- Found preceding transition to idle — compare parameters



You can have heterogeneous events

- Analyze multiple things at once — add other types of events to your pcap (and dissector!) to get better understanding of fault conditions



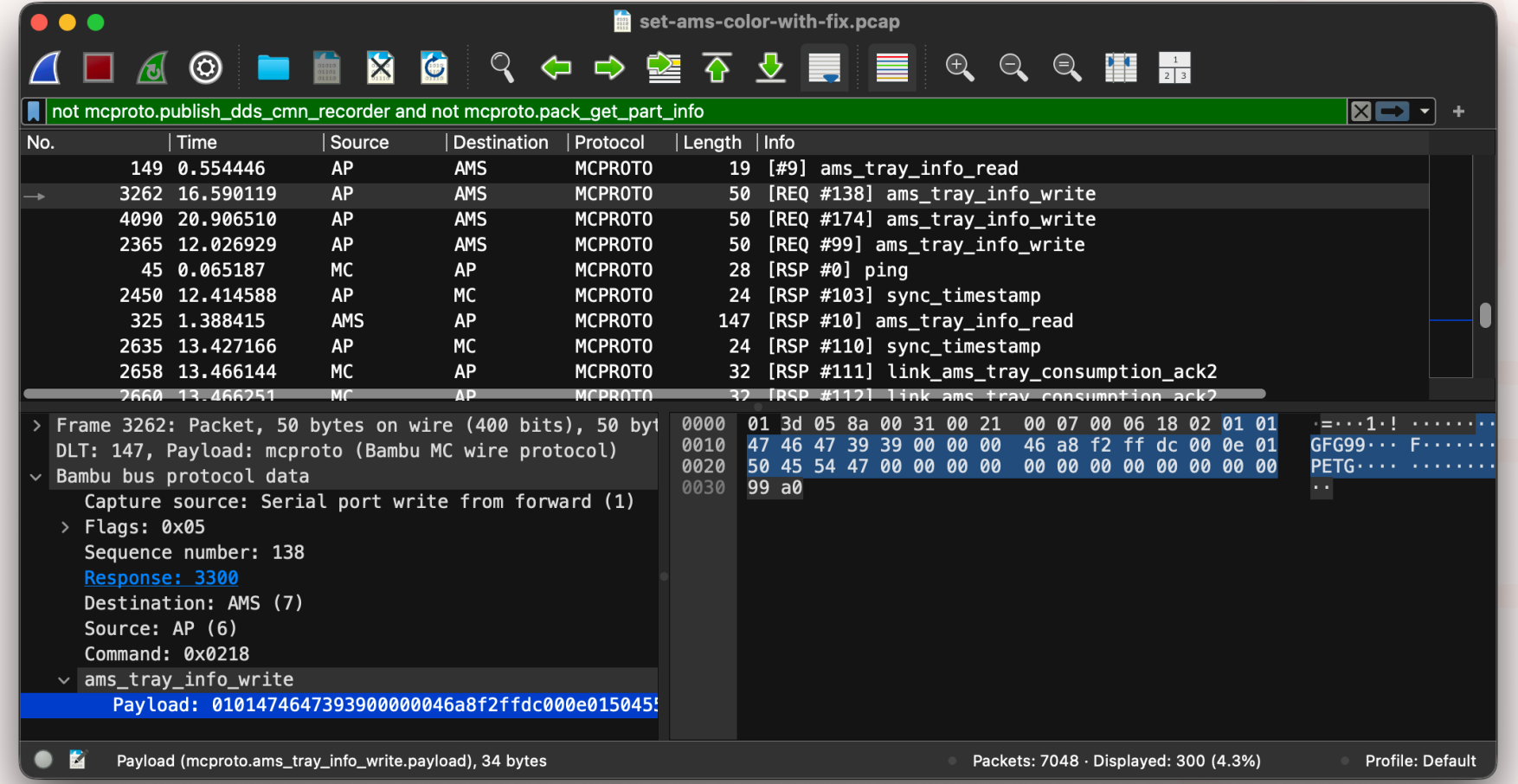
Very quick debug cycle!

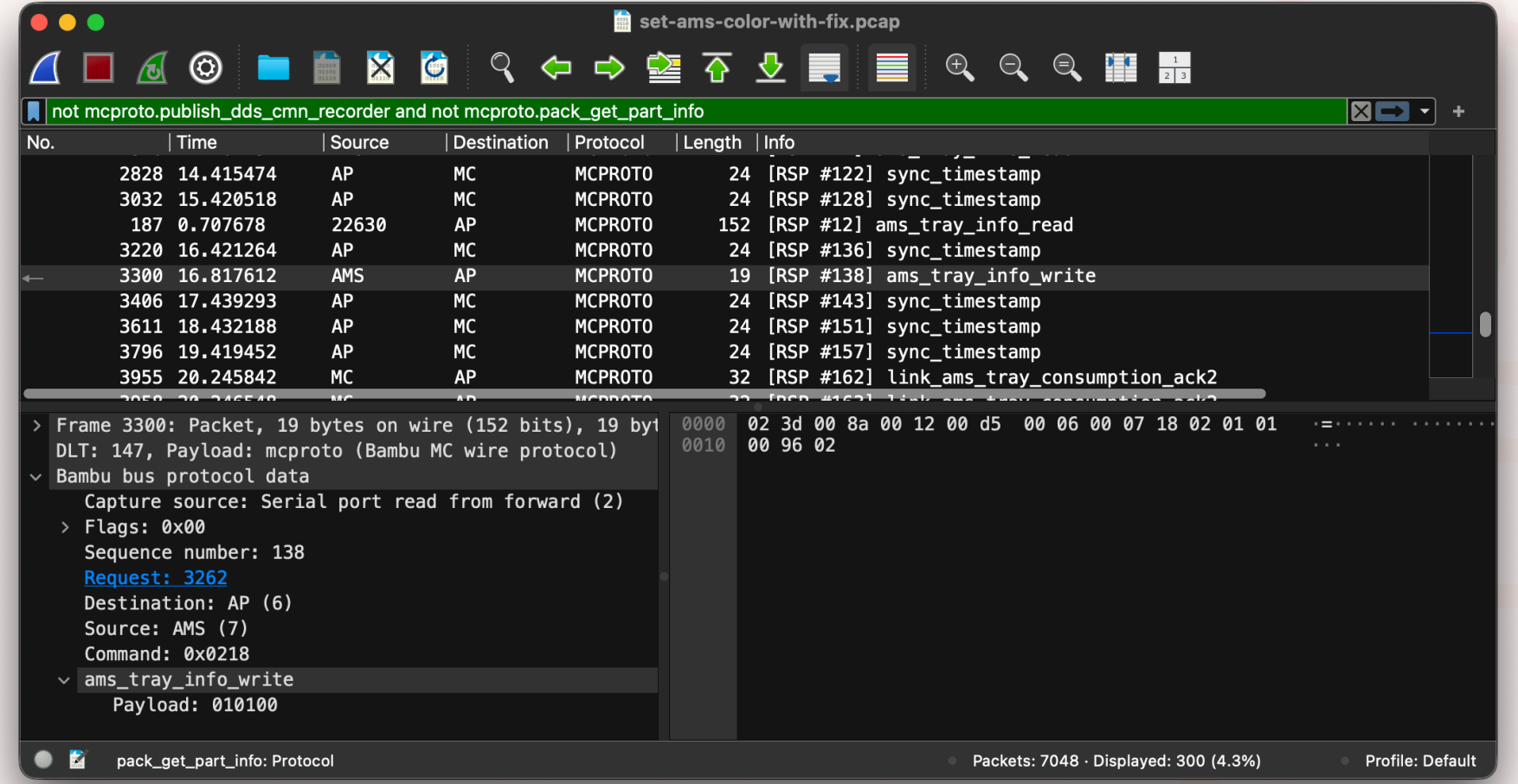
- Previous scripts reparse whole file each time I change anything…
  - Filtering in Wireshark is *fast*
- Being able to link bytes on wire to field indices is very useful to share with SW team
- Set up any fields you want to visualize!

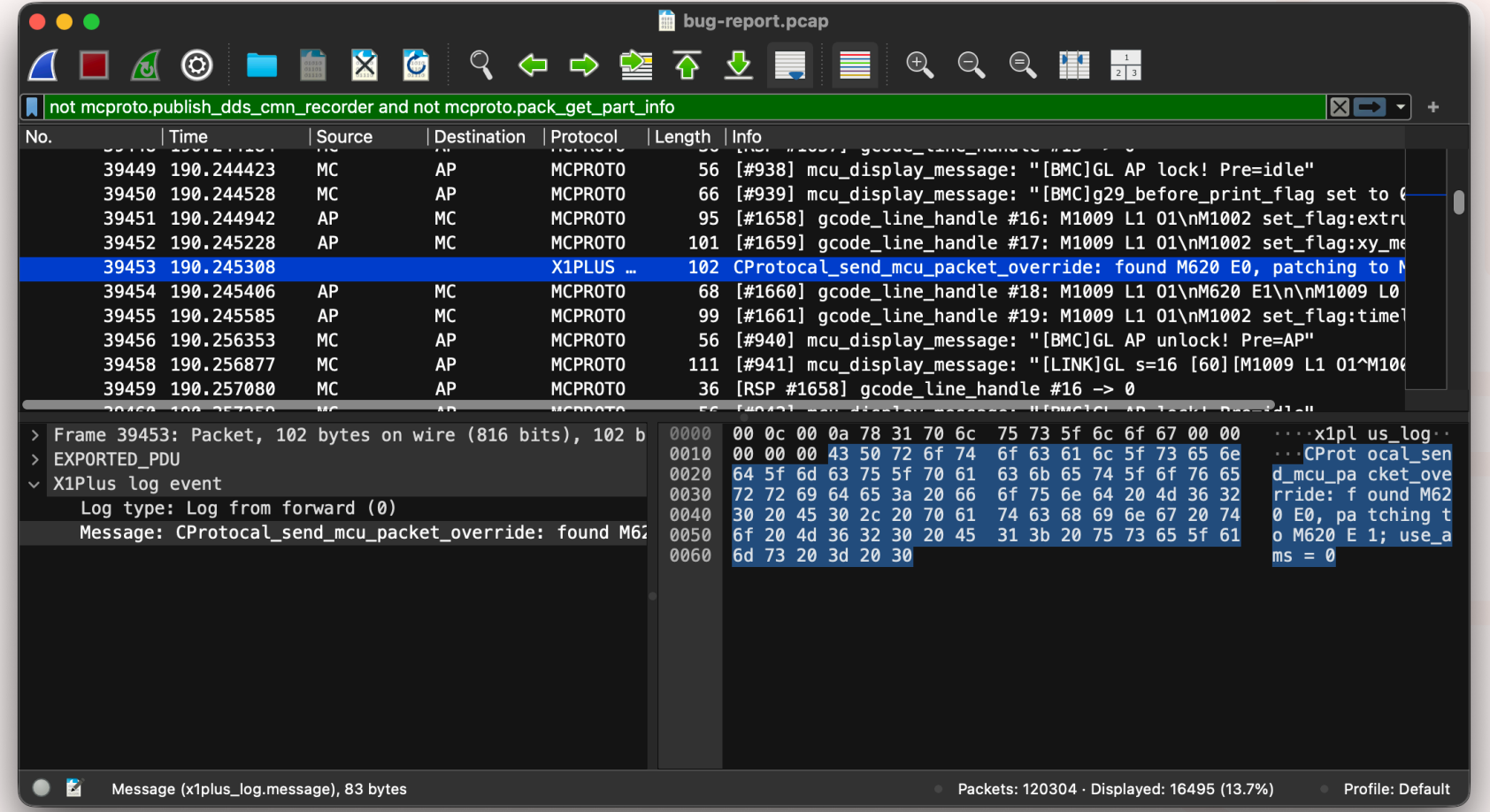
- MSB: **debugging is about exploration; use Wireshark as tool for exploring event-structured data and understanding it better**

The other thing where you want to explore

- PCAP-formatted data is also useful for getting an understanding of *unknown* data on the wire
- Wireshark dissector may become “documentation of record” for proto that you are reverse engineering!
    - Ghidra structure editor only lets you show so much
- Invaluable while visualizing Bambu Lab MC / AP protocol







Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing

```
-- Now for some commands!
function mk_cmd(cmdset, cmdid, name, fields, dis)
        local myname = "mcproto." .. name
        local proto = Proto(myname, name)
        proto.fields = {}
        for _,v in pairs(fields) do table.insert(proto.fields,v) end
        proto.dissector = function (buf, pinfo, root) return dis(proto, fields, buf, pinfo, root) end
        mccommands:set(cmdset * 256 + cmdid, proto)
        return proto
end
```

Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing

```
mk_cmd(2, 9, "mcu_display_message",
        { message = ProtoField.string("mcproto.mcu_display_message.message", "Message") },
        function (proto, fields, buf, pinfo, root)
                local len = buf:len()
                local tree = root:add(proto, buf:range(0, len), "mcu_display_message")
                tree:add(fields.message, buf:range(0, len))
                pinfo.cols.info:append("mcu_display_message: \""..buf:range(0, len):string().."\"")
        end
)
```

Writing dissectors

- Dissector Lua API is arcane, probably useful to have wrappers to hide some of the worst of it and focus on what you are writing
- Just hold your nose and do it


I should write a Wireshark dissector.

Part 2: … and then, we should all dream bigger

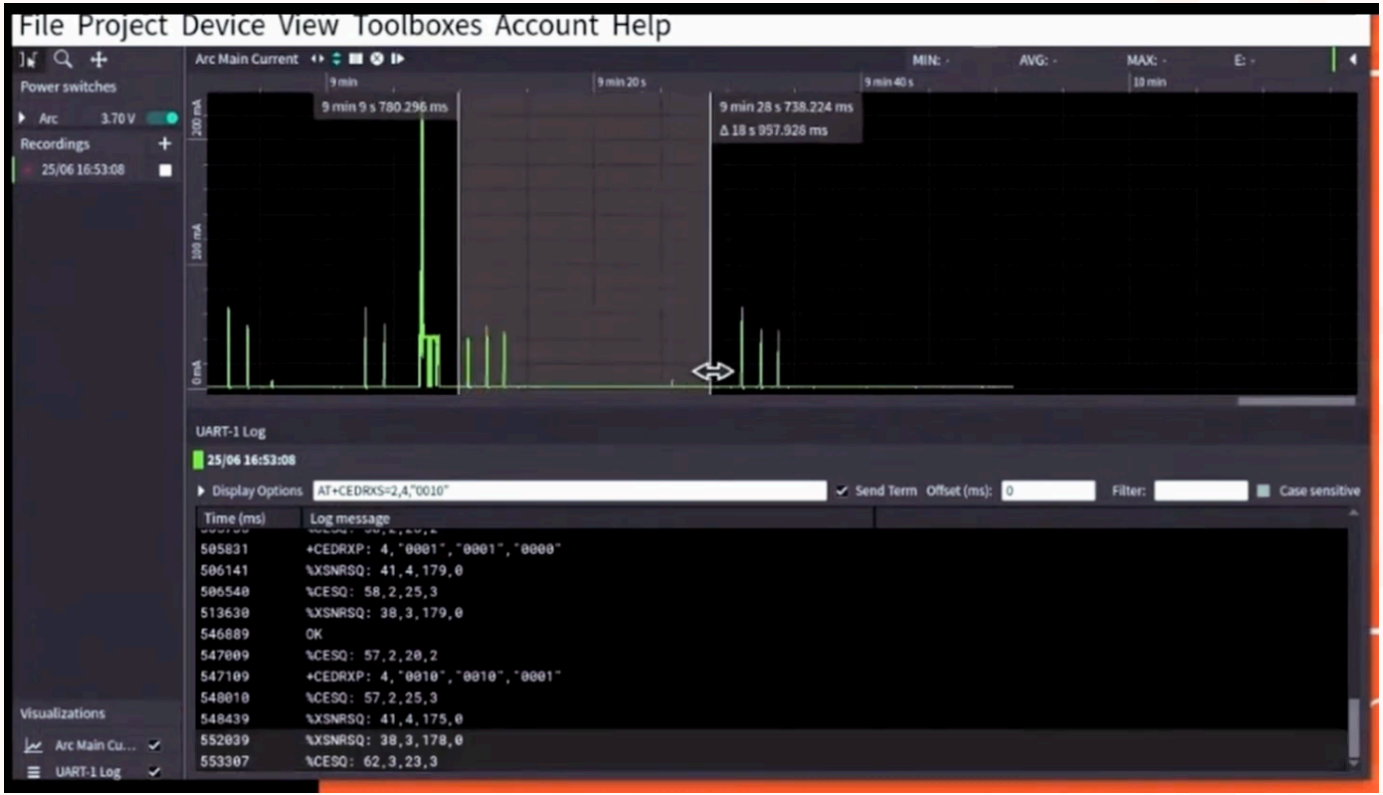
Accelerated Tech

Wireshark is ok, but...

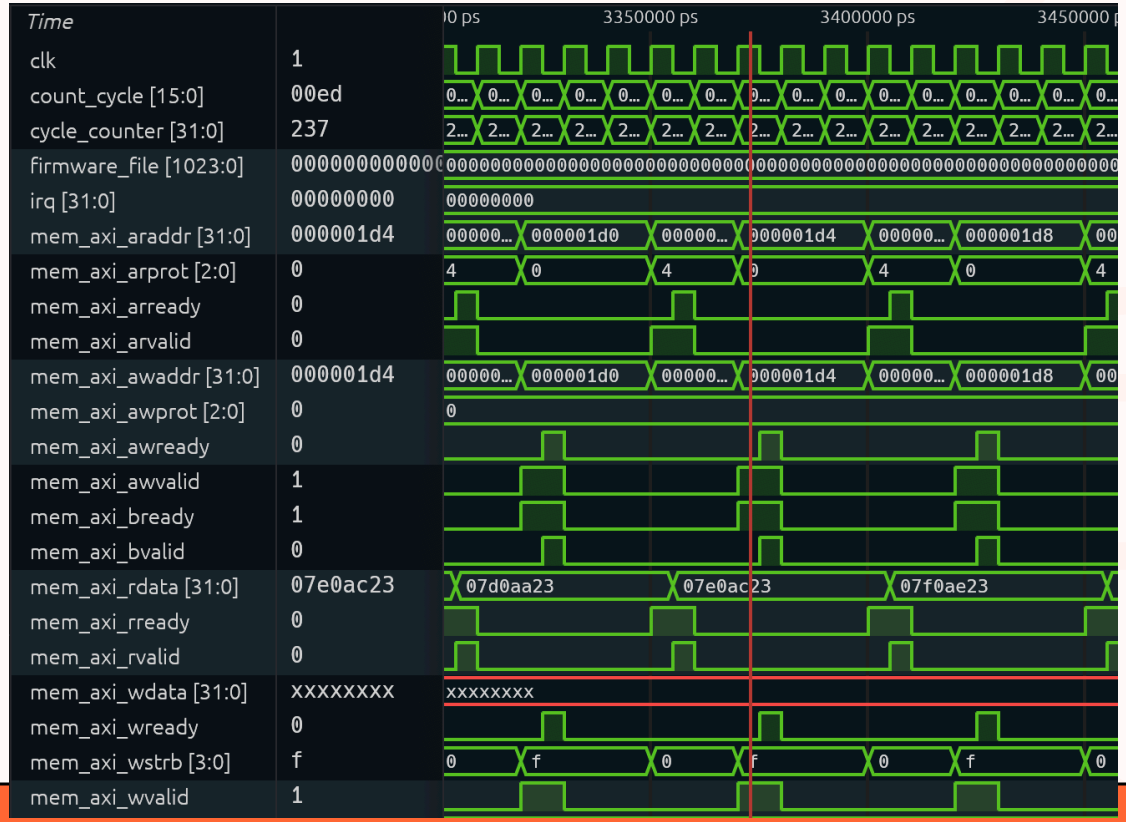
- There are many ways of debugging using time-series data
- Wireshark / pcap tools are only one of them

- Here are a few

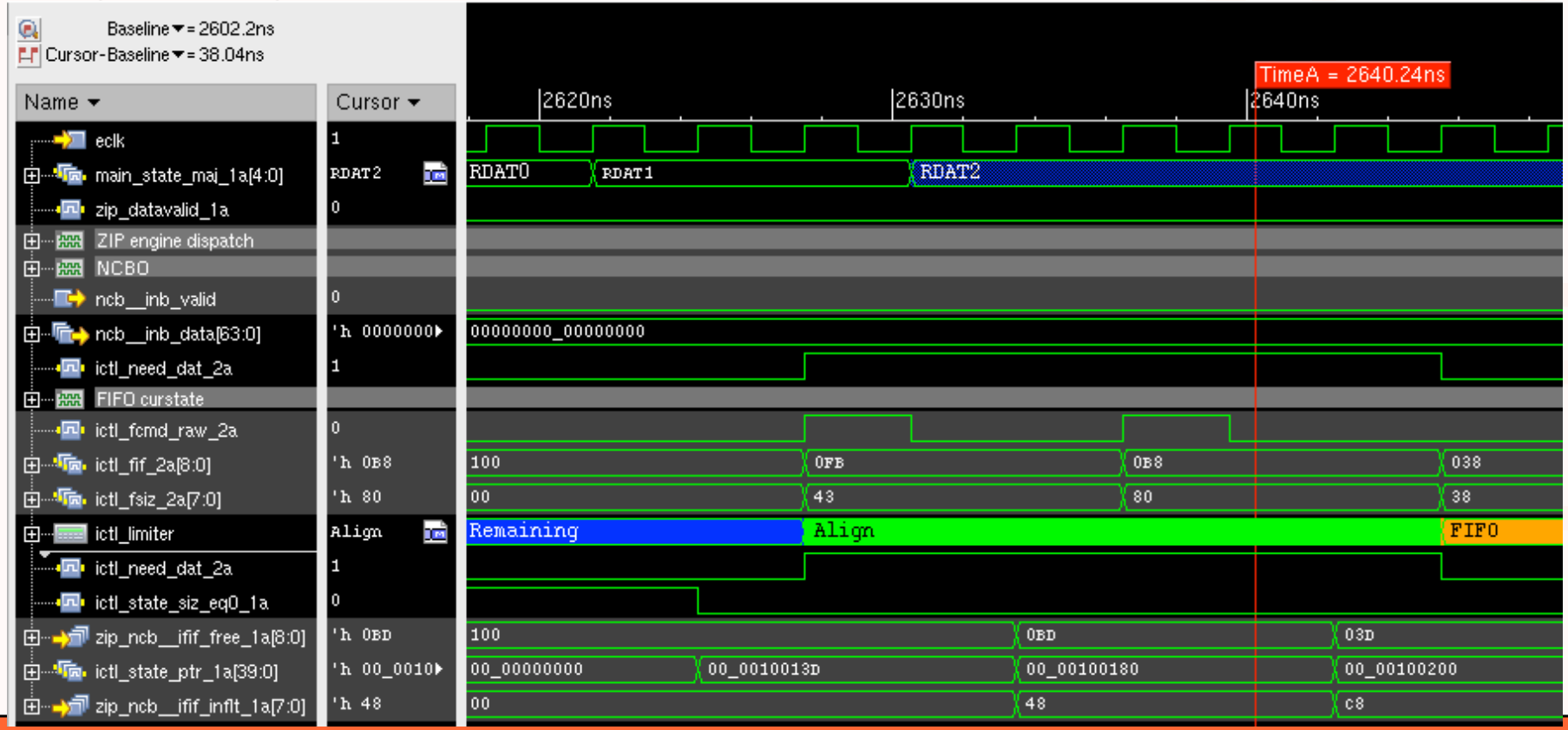
Analog data, synced with UART (Otii Arc)



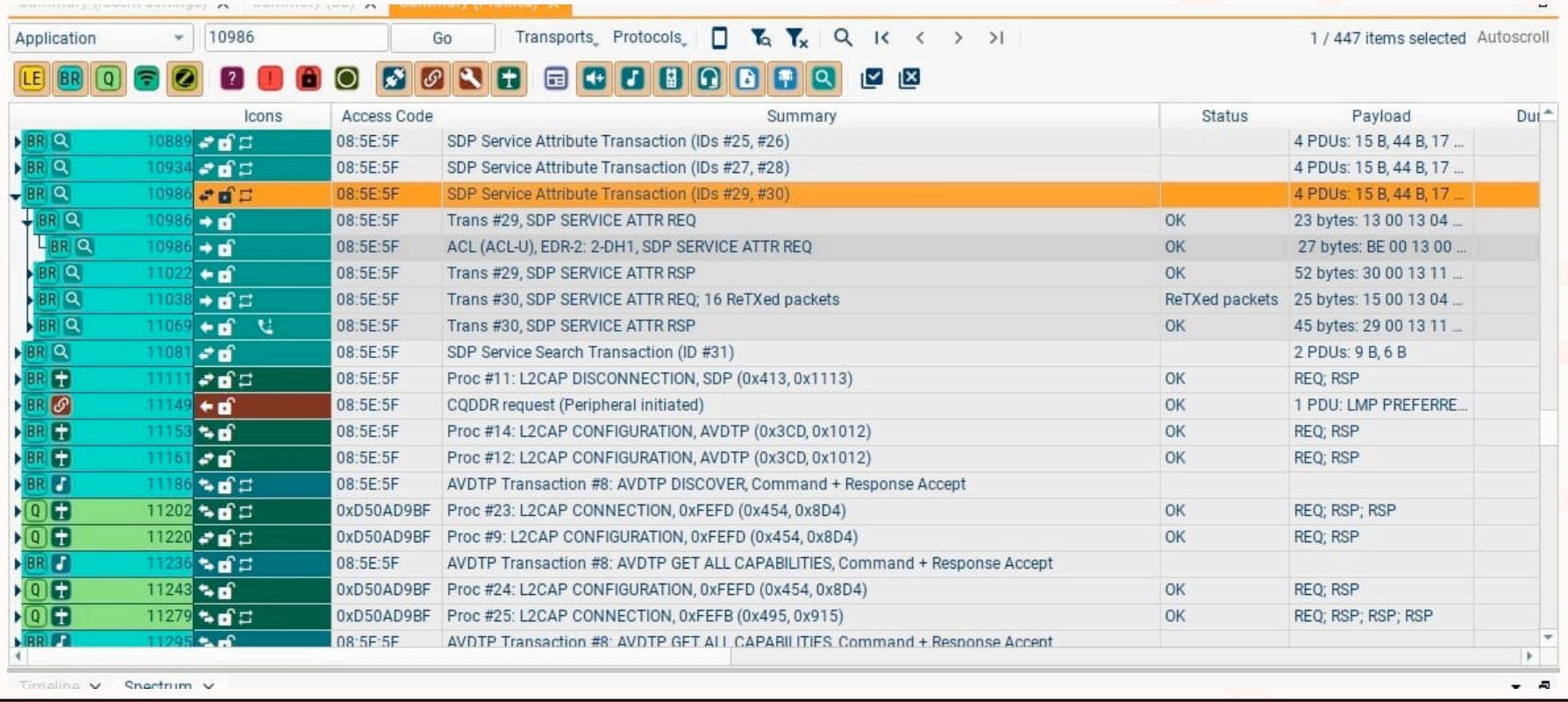
Digital logic waveform viewers (Surfer)



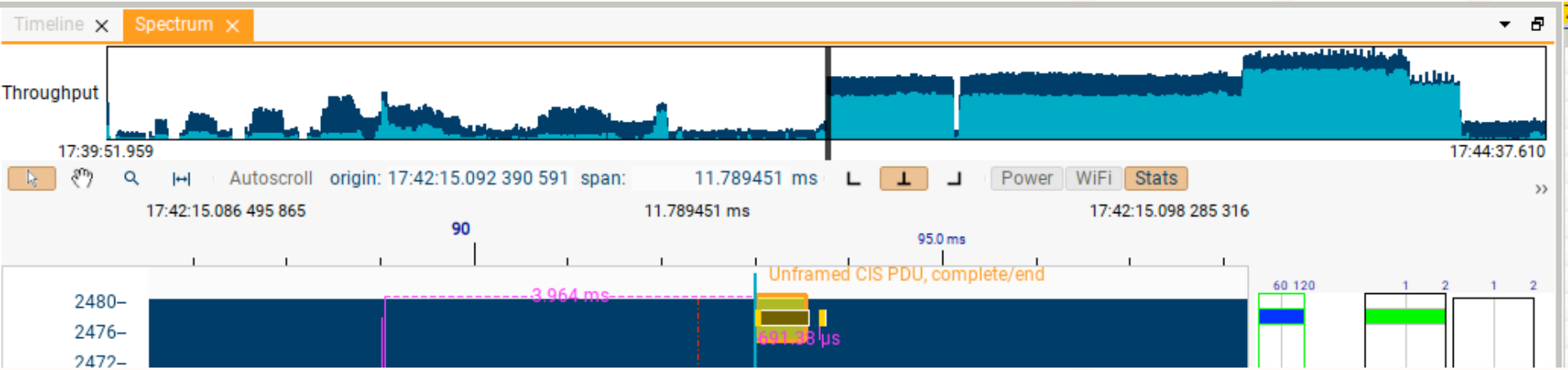
Digital logic waveform viewers (Cadence Simvision, 2009!)



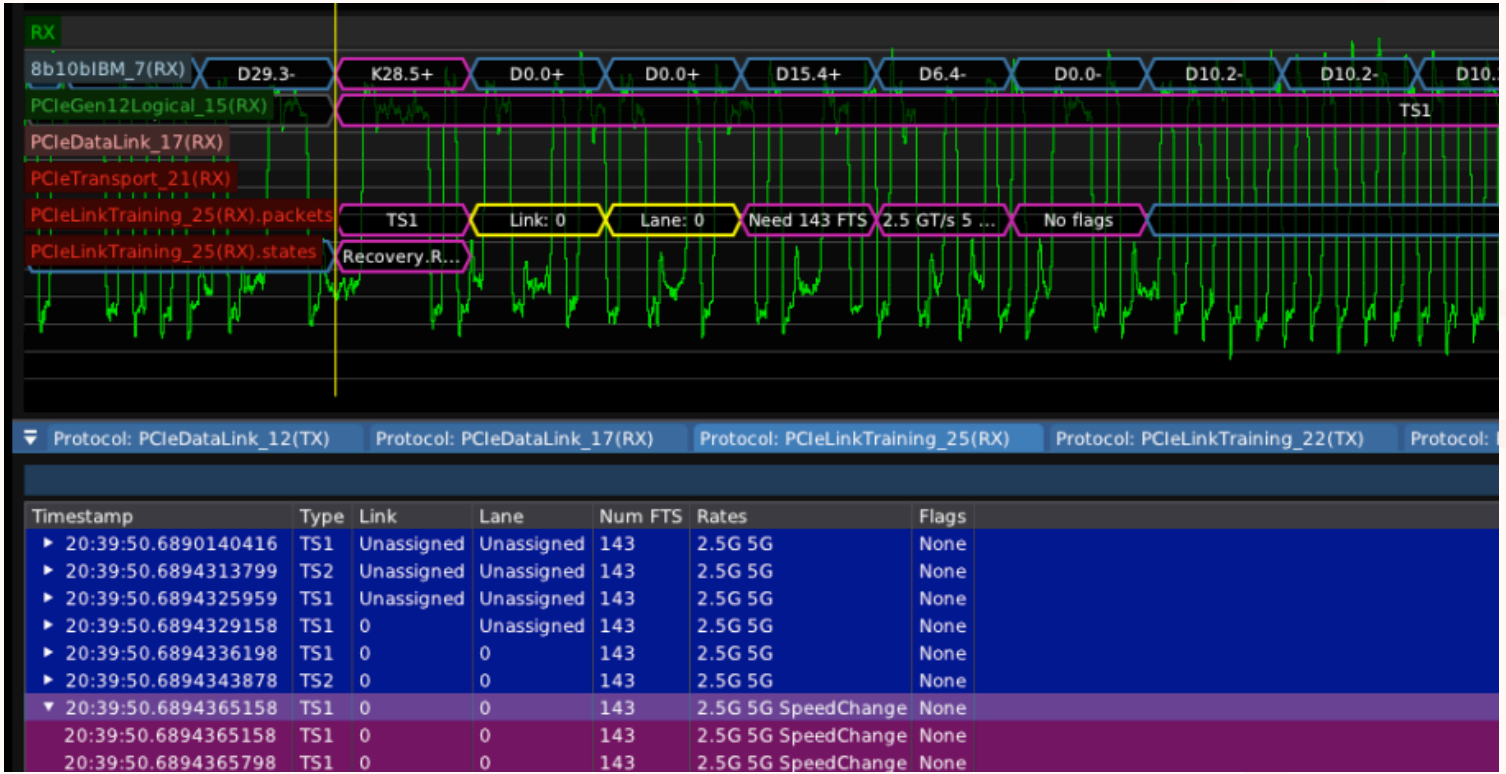
Transaction-level analysis (RFcreations blueSPY)



Spectrum analysis (RFcreations blueSPY)

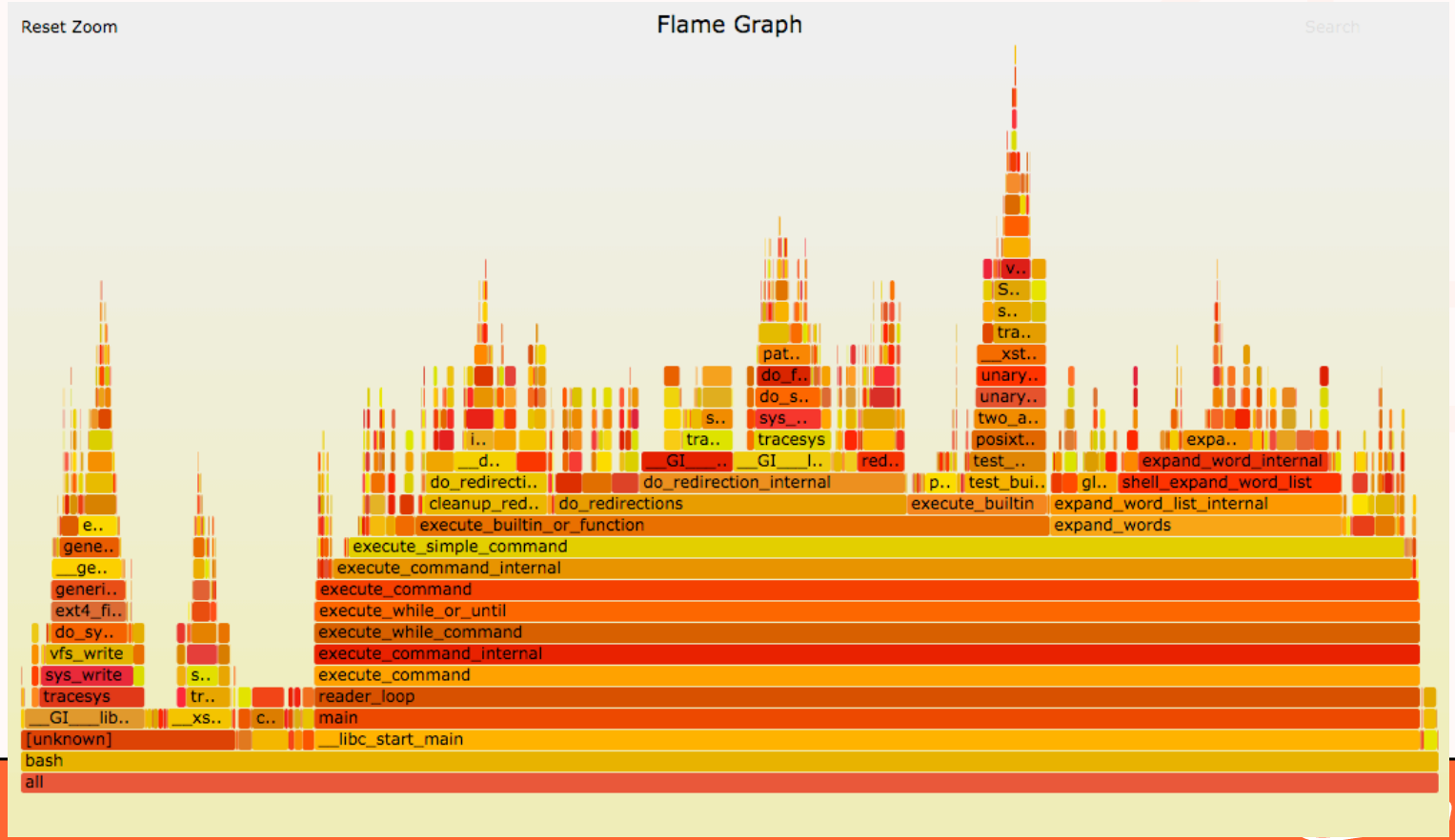


Transaction + analog analysis (ngscopeclient)



Timestamp	Type	Link	Lane	Num FTS	Rates	Flags
▸ 20:39:50.6890140416	TS1	Unassigned	Unassigned	143	2.5G 5G	None
▸ 20:39:50.6894313799	TS2	Unassigned	Unassigned	143	2.5G 5G	None
▸ 20:39:50.6894325959	TS1	Unassigned	Unassigned	143	2.5G 5G	None
▸ 20:39:50.6894329158	TS1	0	Unassigned	143	2.5G 5G	None
▸ 20:39:50.6894336198	TS1	0	0	143	2.5G 5G	None
▸ 20:39:50.6894343878	TS2	0	0	143	2.5G 5G	None
▾ 20:39:50.6894365158	TS1	0	0	143	2.5G 5G SpeedChange	None
20:39:50.6894365158	TS1	0	0	143	2.5G 5G SpeedChange	None
20:39:50.6894365798	TS1	0	0	143	2.5G 5G SpeedChange	None

Program flow: flame charts (Gregg, 2016)



Program values: good old gdb

```
Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$1 = 0
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$2 = 1
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$3 = 2
(gdb) c
Continuing.
```

Program values: good old gdb

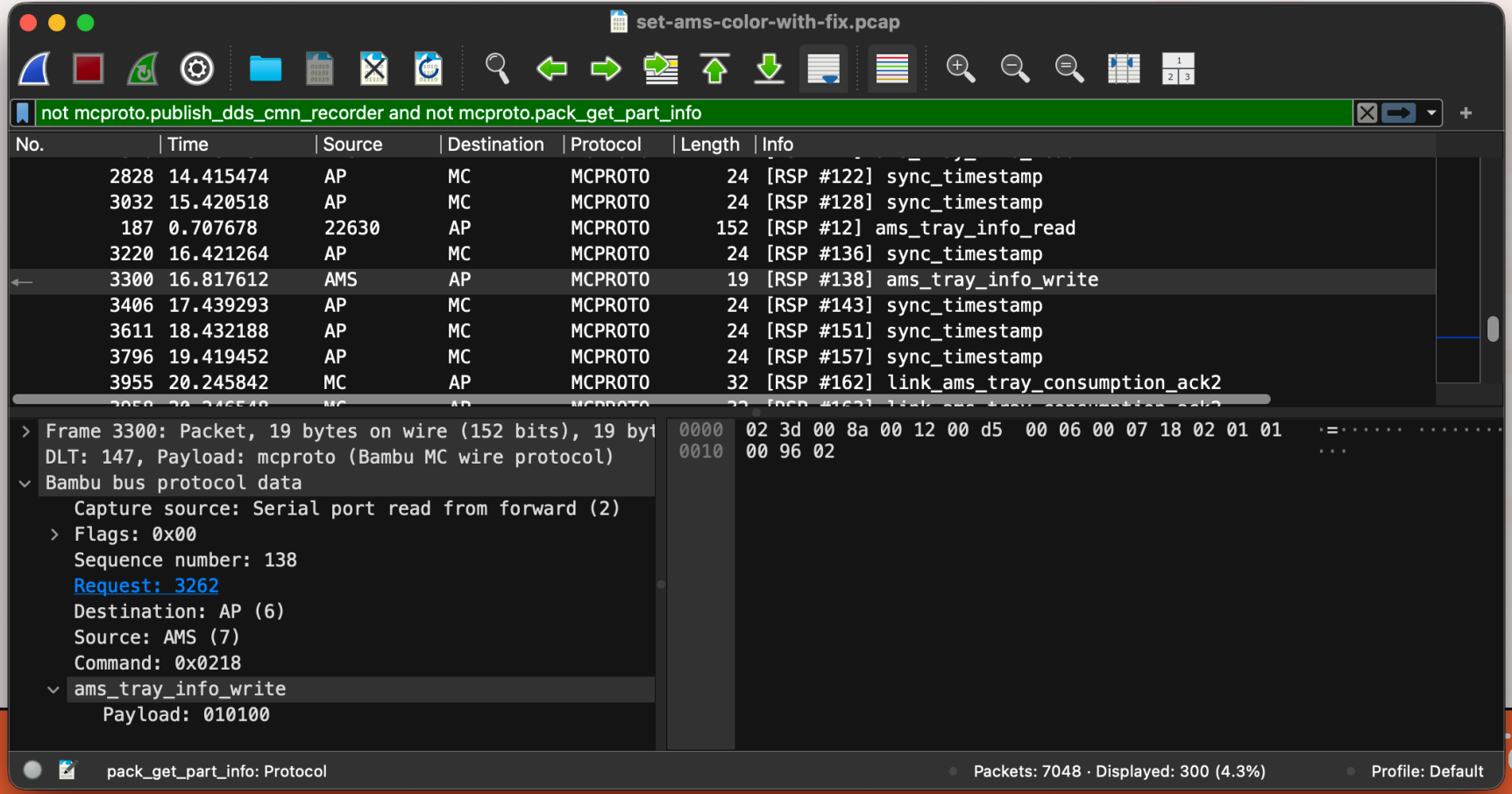
Q: what about more variables?

Q: what about if we ran this in rr, and could step forward and back?

Q: what if all expressions had timestamps associated with them?

```
Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$1 = 0
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$2 = 1
(gdb) c
Continuing.

Breakpoint 1, main () at each-loop.c:3
3               usleep(i);
(gdb) print i
$3 = 2
(gdb) c
Continuing.
```

And, of course, Wireshark



Time series debugging

- All of these are ways to do time series debugging

Time series debugging

- **All of these are ways to do time series debugging**

- None of them are on speaking terms with each other
  - That is a god-damn shame

- I do not know the answer to this

Dream bigger

- It seems clear to me that we are handicapped by tools
- I have often wanted combinations of these
    - scope + ARM swtrace! flame chart plus UART! for the love of everything good, I want memory variables in a waveform viewer!

Dream bigger

- It seems clear to me that we are handicapped by tools
- I have often wanted combinations of these
  - scope + ARM swtrace! flame chart plus UART! for the love of everything good, I want memory variables in a waveform viewer!

- You should build a Wireshark dissector, and experiment with these other modalities too. You deserve better than grep and sed.

- But we all deserve better than what we have. Time for us to get dreaming.

Joshua Wise
April, 2026

Web: https://joshuawise.com/
Work: https://accelerated.tech/
E-mail: joshua@accelerated.tech
Fediverse: @joshua@social.emarhavil.com

Accelerated Tech