Where to get Firmware,

and how to reverse engineering it.

Travis Goodspeed, Dartmouth College, 2022.01.23

What is Firmware?

- Many formats:
  - .BIN -- Flat binary image, lacking a loading address.
  - .DFU -- USB Device Firmware Update
  - .ELF -- Contains everything you'd want. Rather rare.
  - .EXE -- Custom updaters for Windows. Gotta reverse engineer it first!
  - .??? -- Custom file format, lots of variety.

Where to get firmware?

- Download an updater from the manufacturer's website.
- Read the code out of a physical device:
  - Memory chip reader, JTAG.
  - Memory corruption exploit, voltage glitching.
  - Microscopes. (Not joking.)

Firmware Updates

2022.01.17 Upgrade Log | CLICK HERE to download.

APP : V1.1.2 Jan 17 2022,16:31:45

BASE : V1.1.2 Jan 17 2022,15:44:18

Note: This upgrade is important. Please read the following details carefully.

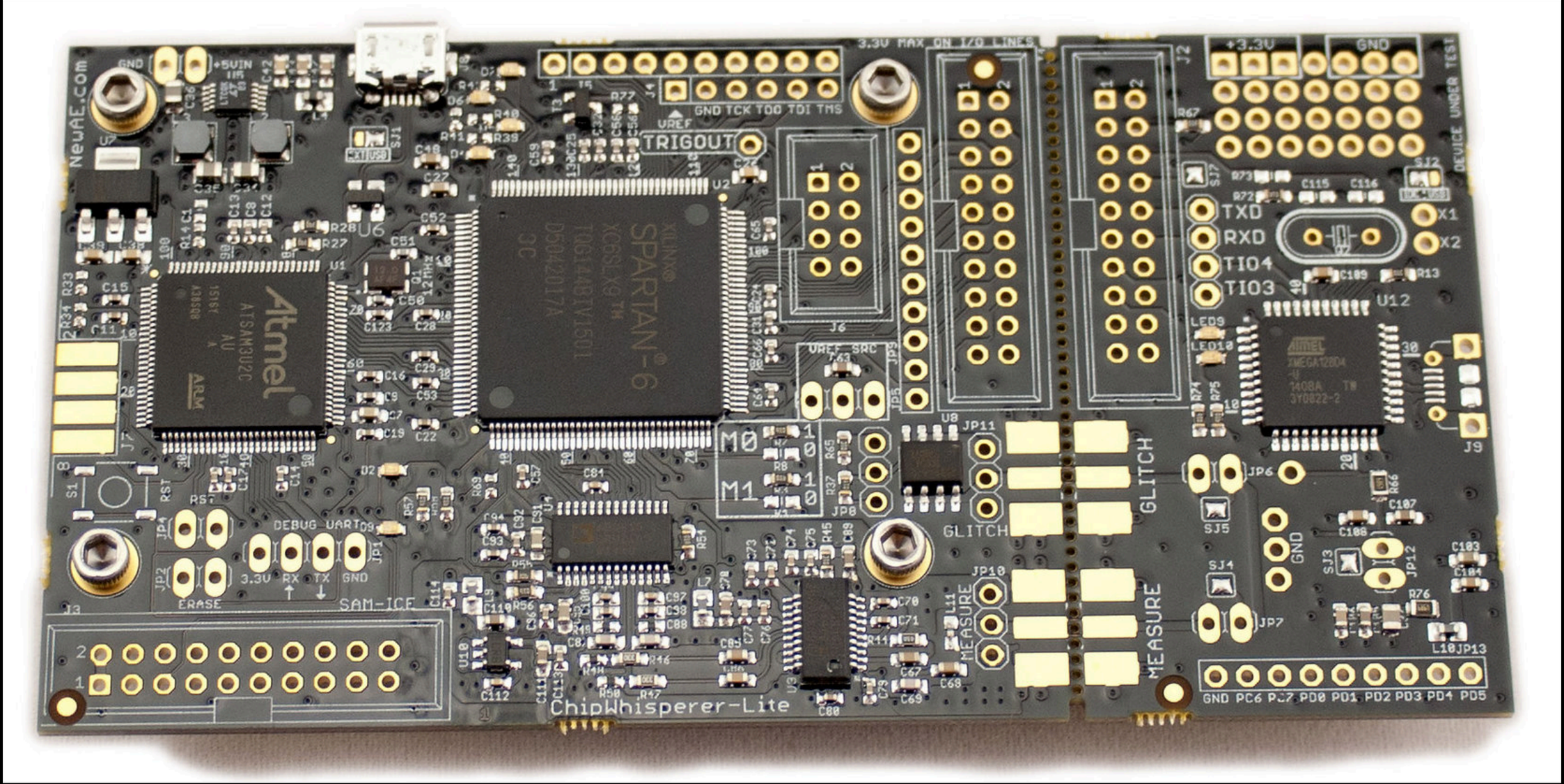
Firmware Updates

- Often encrypted:
  - Tytera MD380 radio uses XOR with a random block.
  - How can we crack that?
- Often wrapped:
  - USB Device Firmware Update protocol in a Windows .EXE.
  - You'll reverse engineer the wrapper first, to get the firmware.

Memory Chip Readers / JTAG

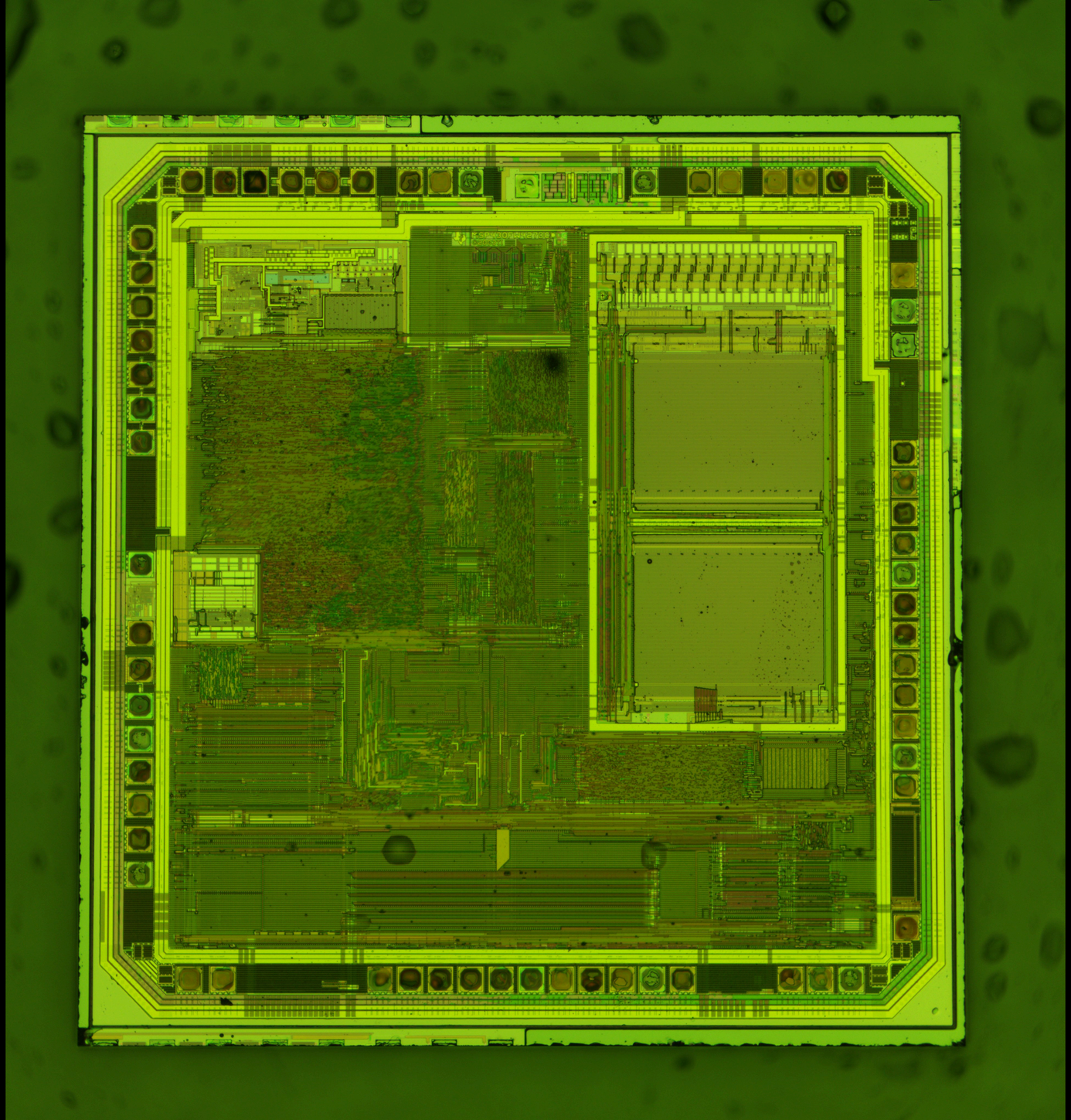
- Unique to reach memory type:
  - Parallel ROM, SPI Flash
  - Compact Flash, Secure Digital
- JTAG
  - Testing protocol for manufacturing.
  - Also does debugging.
  - Sometimes disabled

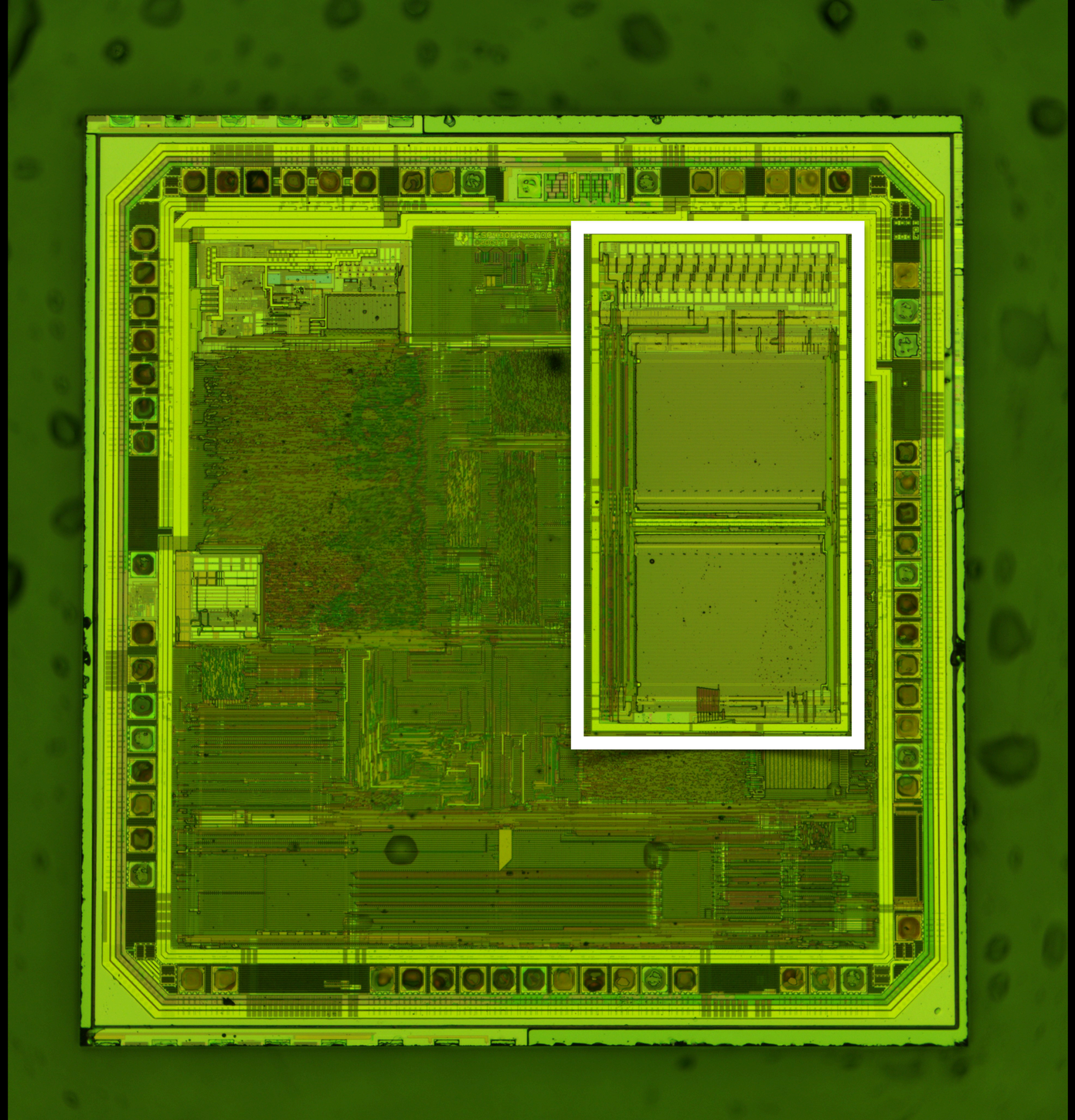
Exploiting the Device to Get the Firmware

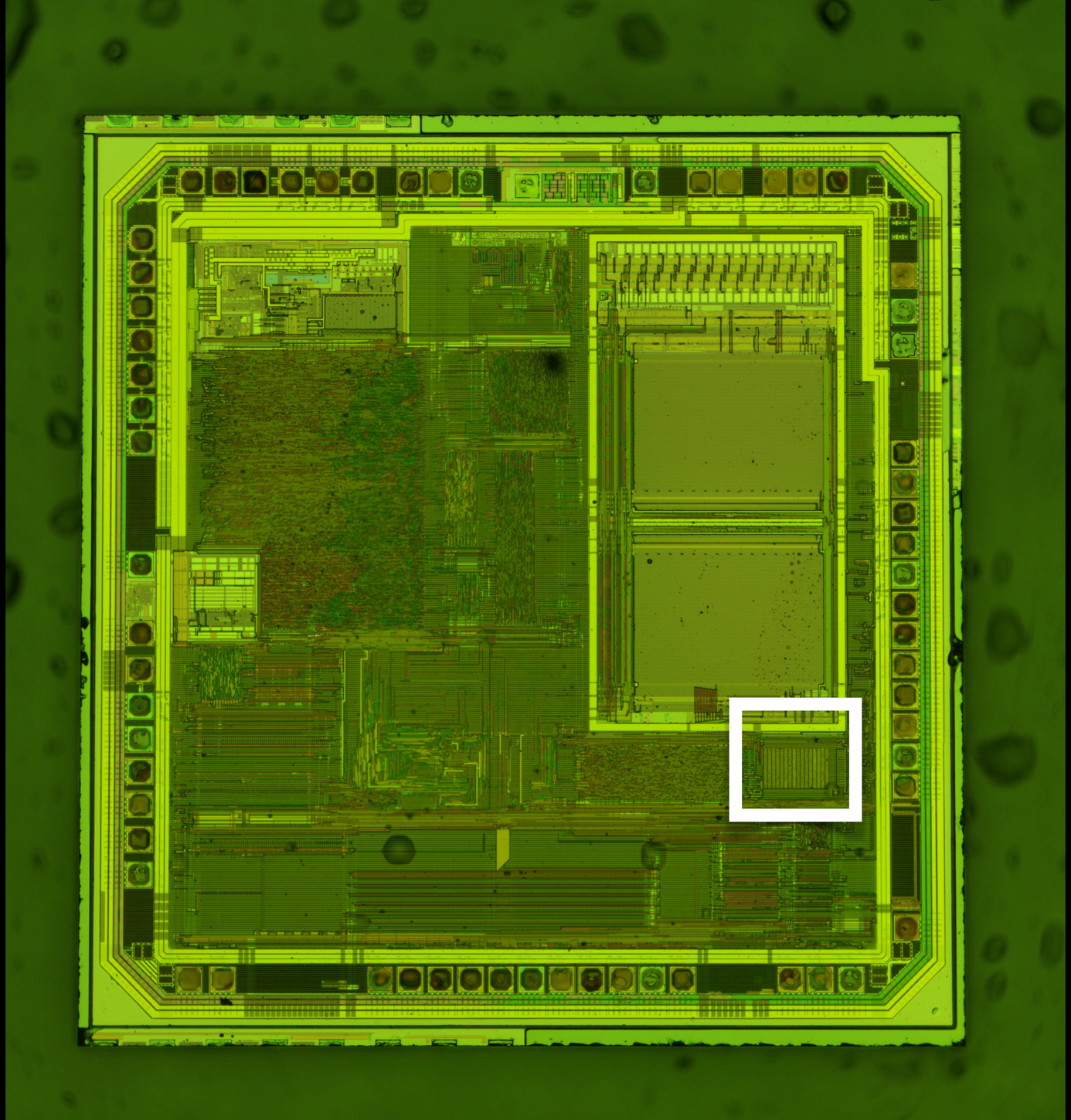


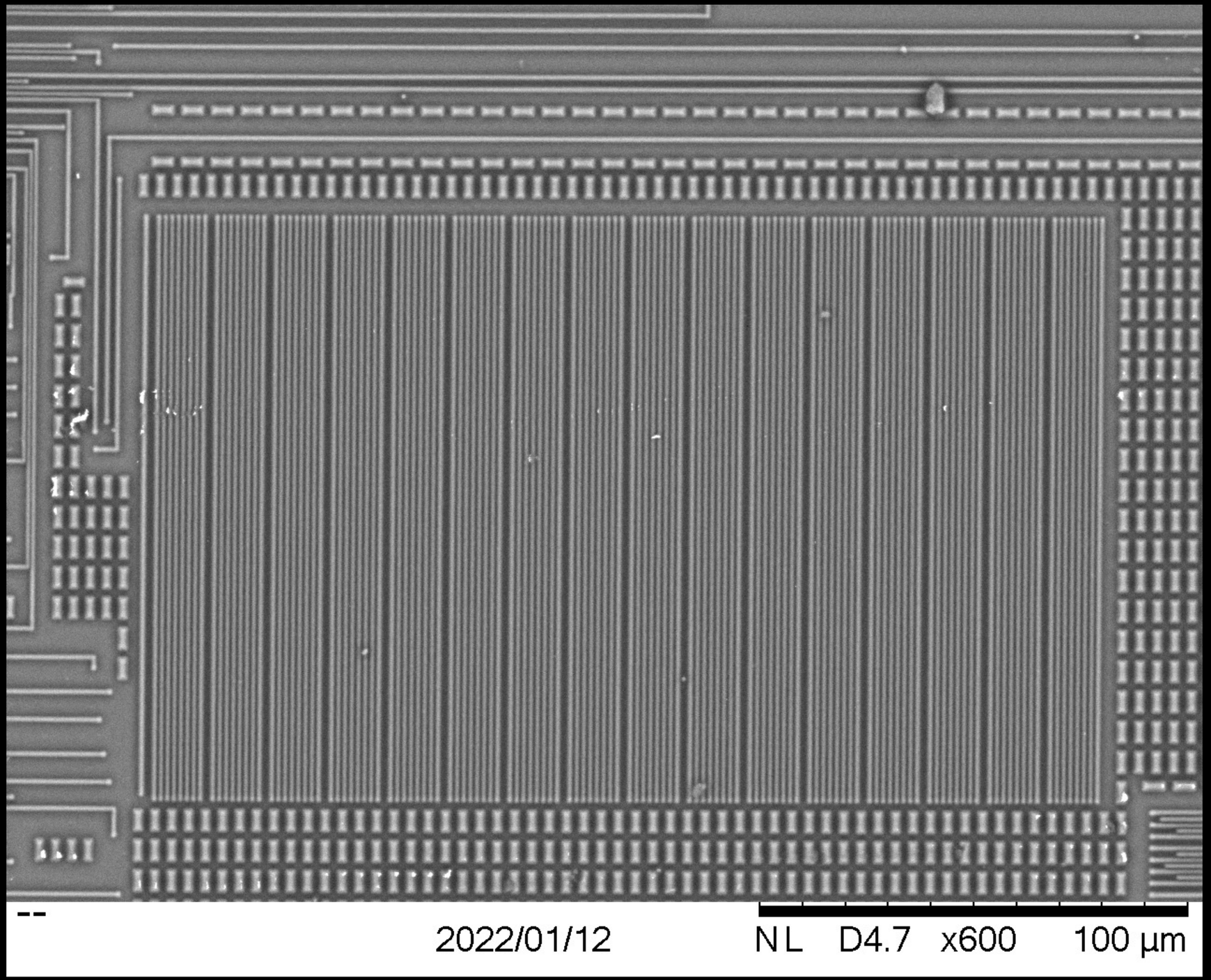
ROM Microscopy

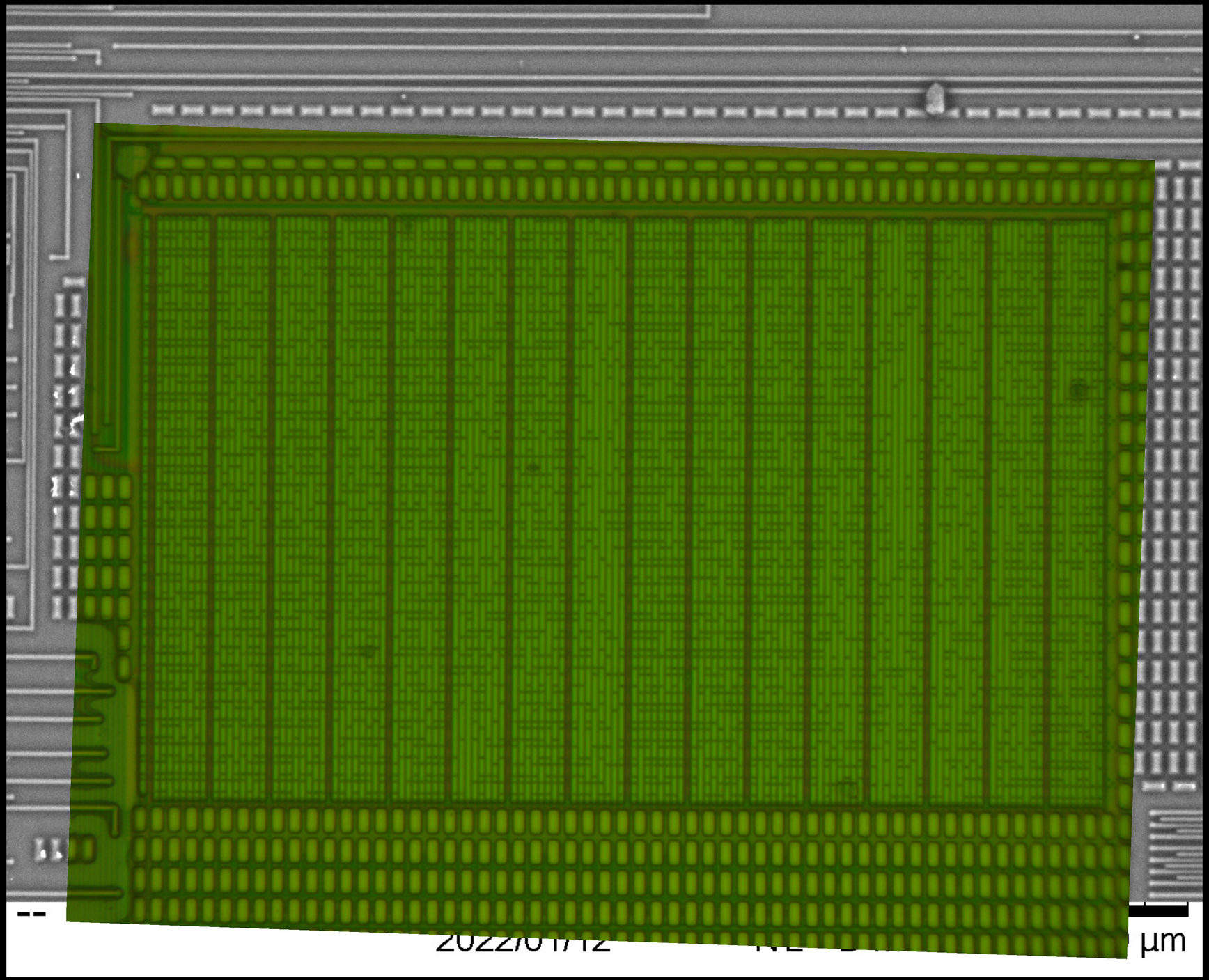
- Mask ROM uses a custom mask layer in chip manufacturing to encode the bits. Different customers can have all other masks the same, and only change the mask that encodes the ROM bits.
- Implant ROM uses a mask with a hole for each bit, then selectively fires a laser at the mask to encode the bits.








--
2022/01/12
NL D4.7 x600 100 µm


--
2022/01/12
µm

Firmware Reverse Engineering

Languages and Loading Addresses

- Much wider variety of architectures than a PC.
  - AARCH32 and Thumb2 in ARM
  - MIPS and MIPS16
  - PowerPC and PPC16/VLE
  - MSP430, AVR, U8/100, Risc V, PIC16, 8051, 6502
- Many chips have two instructions sets: A long and a short one.

Firmware Reverse Engineering

Languages and Loading Addresses

- Where does the code sit?
  - AVR begins at 0x0000, MSP430 *ends* at 0xFFFF.
  - ARM Cortex M3 begins code at 0x0000.0000 or 0x0800.0000.
- Different methods of locating it:
  - Microcontroller datasheet, or SDK's linking scripts.
  - Look for some absolute address, then move the code until they match.
  - Guess until things look right.

Firmware Reverse Engineering

Getting our Bearings

- After loading the code, we don't know any function names.
  - Firmware is statically linked, there are no standard libraries.
  - There are no system calls.
- But we're not helpless:
  - I/O Addresses
  - Interrupt Vector Table

Firmware Reverse Engineering

- Hardware Peripherals have I/O Addresses
  - Standard prefix (0x4000.0000 in ARM)
  - Addressed by a "volatile" pointer.
- These are defined in header files:
  - P1REN is 0x0206
  - P1DIR is 0x0204
  - P1OUT is 0x0202
  - P1IN is 0x0200

```
//! Activate the side button.
void sidebutton_init(){
  //Set mode to I/O, direction to input.
  P1SEL&=~BIT5;
  P1DIR&=~BIT5;
  //Pull up with internal resistor.
  P1REN|=BIT5;
  P1OUT|=BIT5;
}

//! Test the side button.
int sidebutton_pressed(){
  //Easily accessible side button.
  if(!(P1IN&BIT5))
    return 1;

  return 0;
}
```

Firmware Reverse Engineering

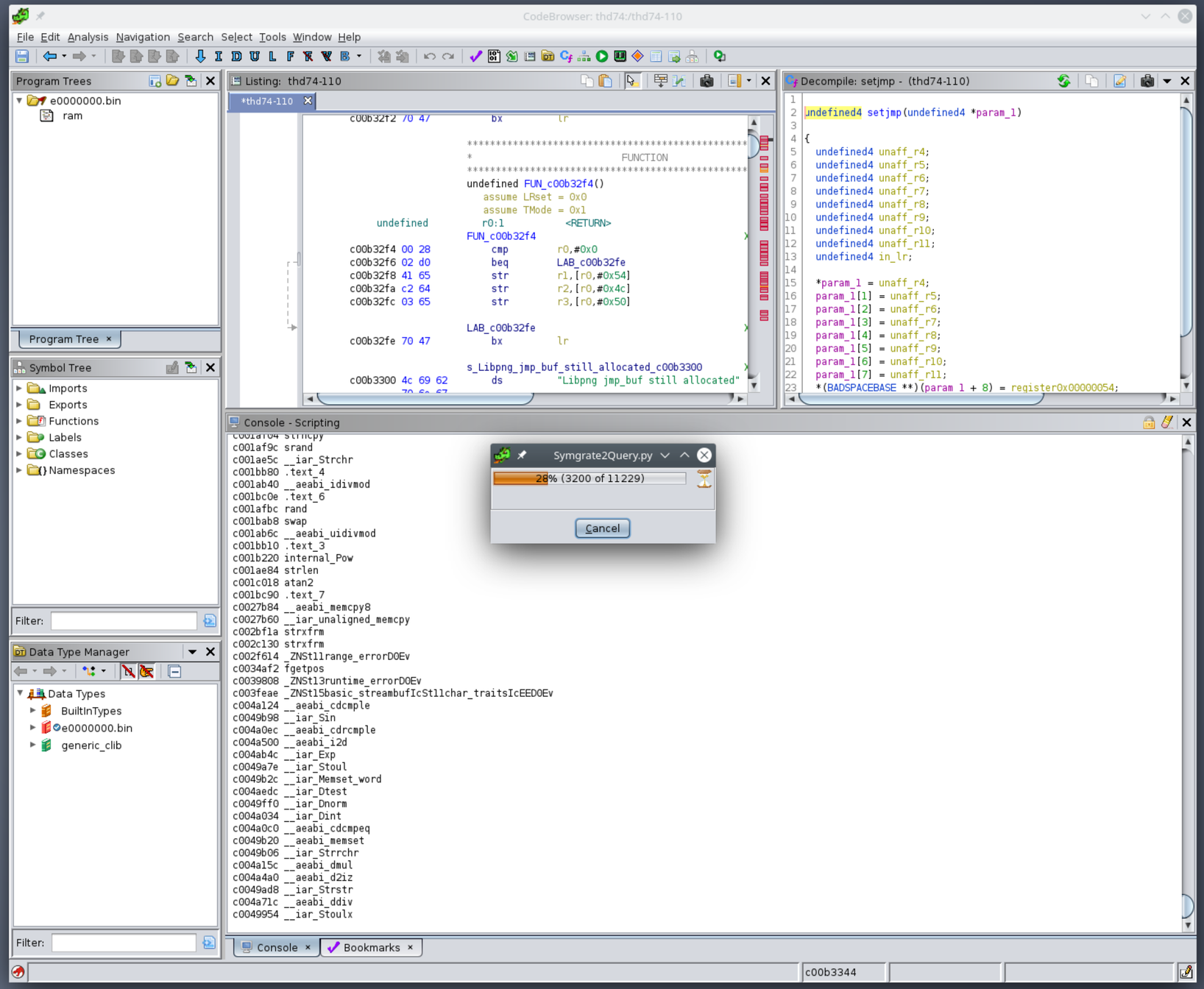
Interrupt Vector Table and Handlers

- Interrupt Vector Table is a list interrupt handlers.
  - "Vector" just means "function pointer."
  - Some architectures have instructions, instead of addresses.
- Continuing our button press example,
  - PORT1_VECTOR is 0xFFE2
  - Instead of polling that pin, the firmware might have an interrupt handler's function pointer at 0xFFE2.

Firmware Reverse Engineering

Recovering Functions from Standard Libraries

- Many functions from come static libraries.
  - Shipped in commercial C compilers and vendor SDKs.
  - Free trials of these compilers are available!
  - License is enforced in the Linker, not the Libraries.
- We can fingerprint these libraries!
  - IDA Pro FLIRT Signatures
  - symgrate.com


CodeBrowser: thd74:/thd74-110
File Edit Analysis Navigation Search Select Tools Window Help
Program Trees
e0000000.bin
ram
Program Tree
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
e0000000.bin
generic_clib
Filter:
Listing: thd74-110
*thd74-110
c00b32f2 70 47 bx lr
FUNCTION
undefined FUN_c00b32f4()
assume LRset = 0x0
assume TMode = 0x1
undefined r0:1 <RETURN>
FUN_c00b32f4
c00b32f4 00 28 cmp r0,#0x0
c00b32f6 02 d0 beq LAB_c00b32fe
c00b32f8 41 65 str r1,[r0,#0x54]
c00b32fa c2 64 str r2,[r0,#0x4c]
c00b32fc 03 65 str r3,[r0,#0x50]
LAB_c00b32fe
c00b32fe 70 47 bx lr
s_Libpng_jmp_buf_still_allocated_c00b3300
c00b3300 4c 69 62 ds "Libpng jmp_buf still allocated"
Decompile: setjmp - (thd74-110)
undefined4 setjmp(undefined4 *param_1)
{
undefined4 unaff_r4;
undefined4 unaff_r5;
undefined4 unaff_r6;
undefined4 unaff_r7;
undefined4 unaff_r8;
undefined4 unaff_r9;
undefined4 unaff_rl0;
undefined4 unaff_rl1;
undefined4 in_lr;
*param_1 = unaff_r4;
param_1[1] = unaff_r5;
param_1[2] = unaff_r6;
param_1[3] = unaff_r7;
param_1[4] = unaff_r8;
param_1[5] = unaff_r9;
param_1[6] = unaff_rl0;
param_1[7] = unaff_rl1;
*(BADSPACEBASE **)(param_1 + 8) = register0x00000054;
Console - Scripting
c001af9c srand
c001ae5c __iar_Strchr
c001bb80 .text_4
c001ab40 __aeabi_idivmod
c001bc0e .text_6
c001afbc rand
c001bab8 swap
c001ab6c __aeabi_uidivmod
c001bb10 .text_3
c001b220 internal_Pow
c001ae84 strlen
c001c018 atan2
c001bc90 .text_7
c0027b84 __aeabi_memcpy8
c0027b60 __iar_unaligned_memcpy
c002bf1a strxfrm
c002c130 strxfrm
c002f614 _ZNSt11range_errorD0Ev
c0034af2 fgetpos
c0039808 _ZNSt13runtime_errorD0Ev
c003feae _ZNSt15basic_streambufIcSt11char_traitsIcEED0Ev
c004a124 __aeabi_cdcmple
c0049b98 __iar_Sin
c004a0ec __aeabi_cdrcmple
c004a500 __aeabi_i2d
c004ab4c __iar_Exp
c0049a7e __iar_Stoul
c0049b2c __iar_Memset_word
c004aedc __iar_Dtest
c0049ff0 __iar_Dnorm
c004a034 __iar_Dint
c004a0c0 __aeabi_cdcmpeq
c0049b20 __aeabi_memset
c0049b06 __iar_Strrchr
c004a15c __aeabi_dmul
c004a4a0 __aeabi_d2iz
c0049ad8 __iar_Strstr
c004a71c __aeabi_ddiv
c0049954 __iar_Stoulx
Symgrate2Query.py
28% (3200 of 11229)
Cancel
Console
Bookmarks
c00b3344

A Tour of the Tower of Babel

So many languages!

- MSP430
  - 16 bit RISC instruction words, optional 16-bit prefix.
  - Von Neumann architecture.
- AVR
  - 16-bit or 32-bit instruction word
  - Harvard architecture.

A Tour of the Tower of Babel

So many languages!

- 8051
  - CISC instruction set, 1 to 3 bytes long.
  - Harvard with multiple memory types:
    - Internal RAM
    - Special Function Registers
    - Program Memory
    - External Data Memory

A Tour of the Tower of Babel

So many languages!

- ARM micros have both AARCH32 and Thumb2 instruction sets:
  - Load/Store Architect
  - Low-bit of PC (IP) choses instruction set.
- AARCH32 (even)
  - 32-bit RISC instruction words
- Thumb2 (odd)
  - 16-bit instruction words, sometimes 32-bit pairs.

A Tour of the Tower of Babel

So many languages!

- Plenty more languages
  - TMS320 -- Found in DSPs, incompatible between revisions.
  - RiscV -- Open source, common in ASICs.
  - U8/100 -- Casio calculators, Oki LCD controllers.
  - Z80, 68000 -- Texas Instruments calculators
  - SATURN -- HP calculators
- Full custom and manufacturer-specific architectures are common in smaller chips.
- ARM/Thumb2 is semi-standard in larger devices.

How to identify the Instruction Set?

- Interrupt Table
  - MSP430 has 16-bit pointers at the *end* of memory.
  - Thumb2 has one even pointer at the beginning, followed by odd pointers.
  - Z80 uses instructions, instead of addresses.
- Instructions
  - Common entry and exit of functions.
  - "binwalk -A" searches for known patters.

How to identify the Instruction Set?

Example Interrupt Table

- Interrupt Table is a list of handler addresses for interrupts.
  - In X86, interrupts are usually system calls.
  - In embedded systems, they are signals from I/O devices.
- "Reset Vector" is the entry point for our program, like main().
- "PORT1 Vector" called when something happens on PORT1.

How to identify the Instruction Set?

Example Interrupt Table

- 30 1A 00 20
- 15 56 00 08
- 29 54 00 08
- 2B 54 00 08
- 2D 54 00 08
- 2F 54 00 08

How to identify the Instruction Set?

Example Interrupt Table

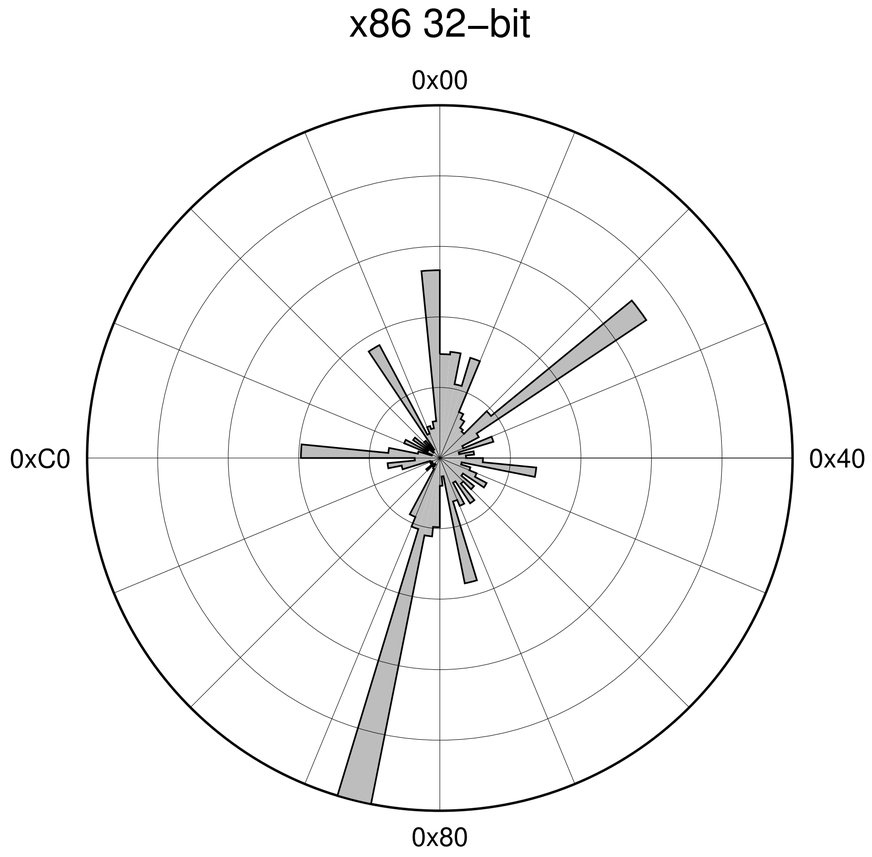
- 30 1A 00 20 = 0x2000.1A30
- 15 56 00 08 = 0x0800.5615
- 29 54 00 08 = 0x0800.5429
- 2B 54 00 08 = 0x0800.542B
- 2D 54 00 08 = 0x0800.542D
- 2F 54 00 08 = 0x0800.542F

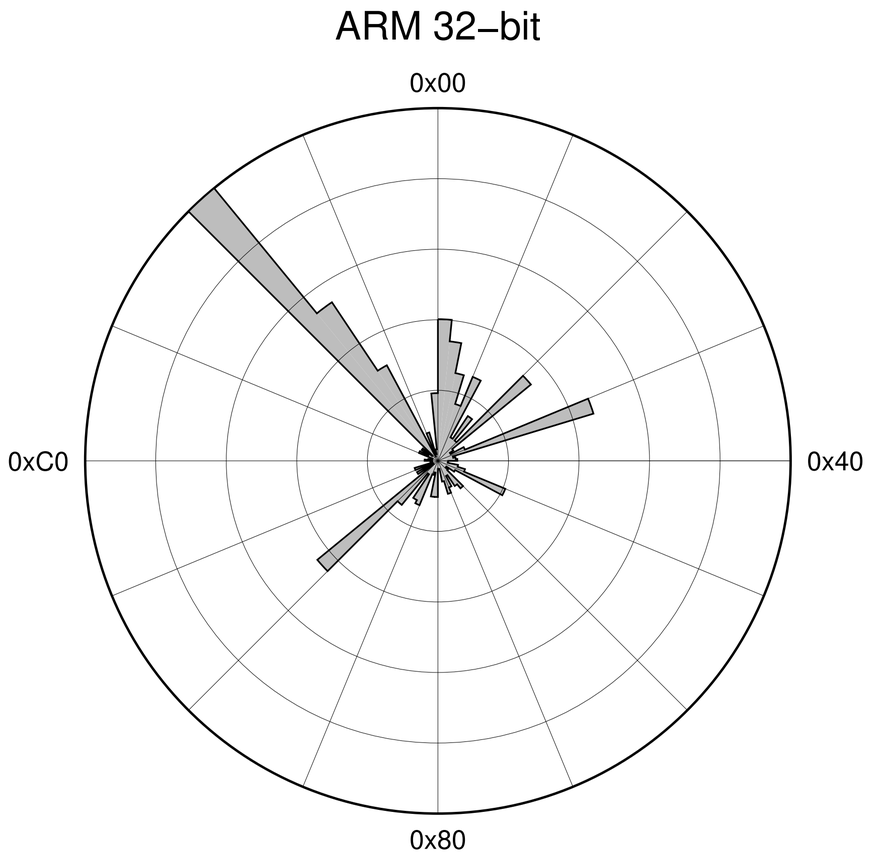
How to identify the Instruction Set?

Example Interrupt Table (STM32F405, Thumb2)

- 30 1A 00 20 = 0x2000.1A30 Top of Stack
- 15 56 00 08 = 0x0800.5615 Reset Vector
- 29 54 00 08 = 0x0800.5429 Non-Maskable Interrupt (NMI)
- 2B 54 00 08 = 0x0800.542B Hard Fault
- 2D 54 00 08 = 0x0800.542D Mem Manage
- 2F 54 00 08 = 0x0800.542F Bus Fault
- Page 372 of 1,751 in the STM32F405 Reference Manual.

Identifying by Common Instructions





Identifying by Common Instructions

- ARM/Thumb2
  - Functions begin with "PUSH {.., LR}" and end with "POP {.., PC}".
  - Literal constants sit between functions.
- MSP430
  - Functions end with "RET." (0x4130)
  - "CALL" (0x12B0) is very common.
  - No gaps between functions.