

SoCsploration

Harvard Architecture Exploitation with Teridian Smart Grid Chips

Nathan Keltner & Josh Thomas

echo \$AGENDA

- WTF is a SoC?
- Harvard is Cool and so can you
- Hardware Reversing
- Security & Teridian chips
- Exploitation!
- Glitching and other musings



./whoami

- Nathan Keltner
 - @natronkeltner
 - Josh Thomas
 - @m0nk_dot
-
- Paid by the epic people @ Accuvant Labs





WTF IS A SOC?



WTF is a SoC

- System on a Chip
- Everything you need to run Fu (except voltage & software)
 - MicroController
 - Memory [{i|x}RAM, FLASH]
 - Peripherals
 - (sometimes) Networking [Wi-Fi, Bluetooth, etc]
- Embedded things / Phones / Cars /
 - (everything – {laptops, servers, etc.})
- Cheap and low power!





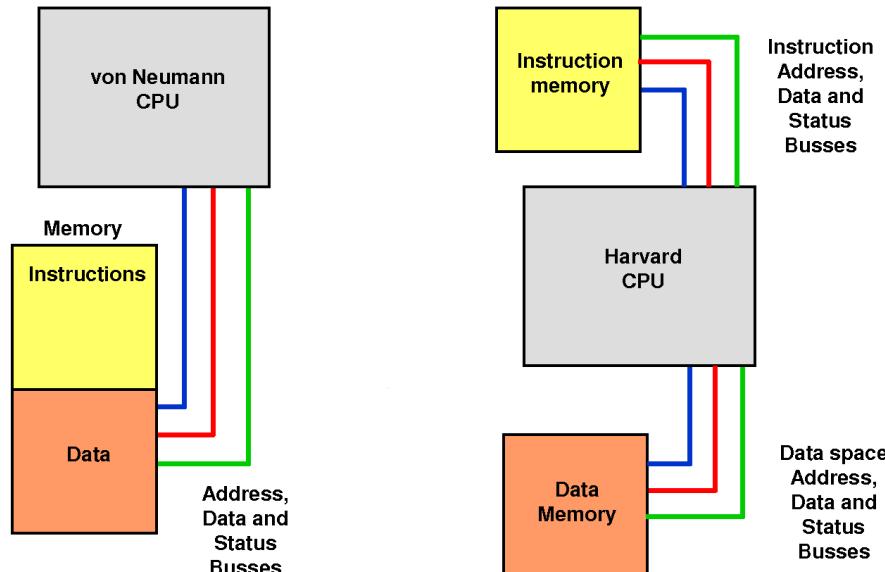
HARVARD IS COOL

and so can you...



FIGHT!

von Neumann and Harvard Architectures



Hardware Computer Organization for the Software Professional
Arnold S. Berger



So, I like x86?

- No NX but disjoint data / code means conceptually hardware enforced NX
- Dual bus allows many instructions to be single cycle (prefetch occurs in parallel)
- Program & Data can have different bit widths
- Crazy registers
- Custom compilers / tools
- Banking, lolPointers, lolStack and more



Tell me more!

- Neighbor Goodspeed preaching in Paris
[http://www.nosuchcon.org/talks/
D1_03_goodspeed_Nifty_Tricks_and_Sage
%20Advice_for_Shellcode_on_EMBEDDED_Systems.pdf](http://www.nosuchcon.org/talks/D1_03_goodspeed_Nifty_Tricks_and_Sage%20Advice_for_Shellcode_on_EMBEDDED_Systems.pdf)



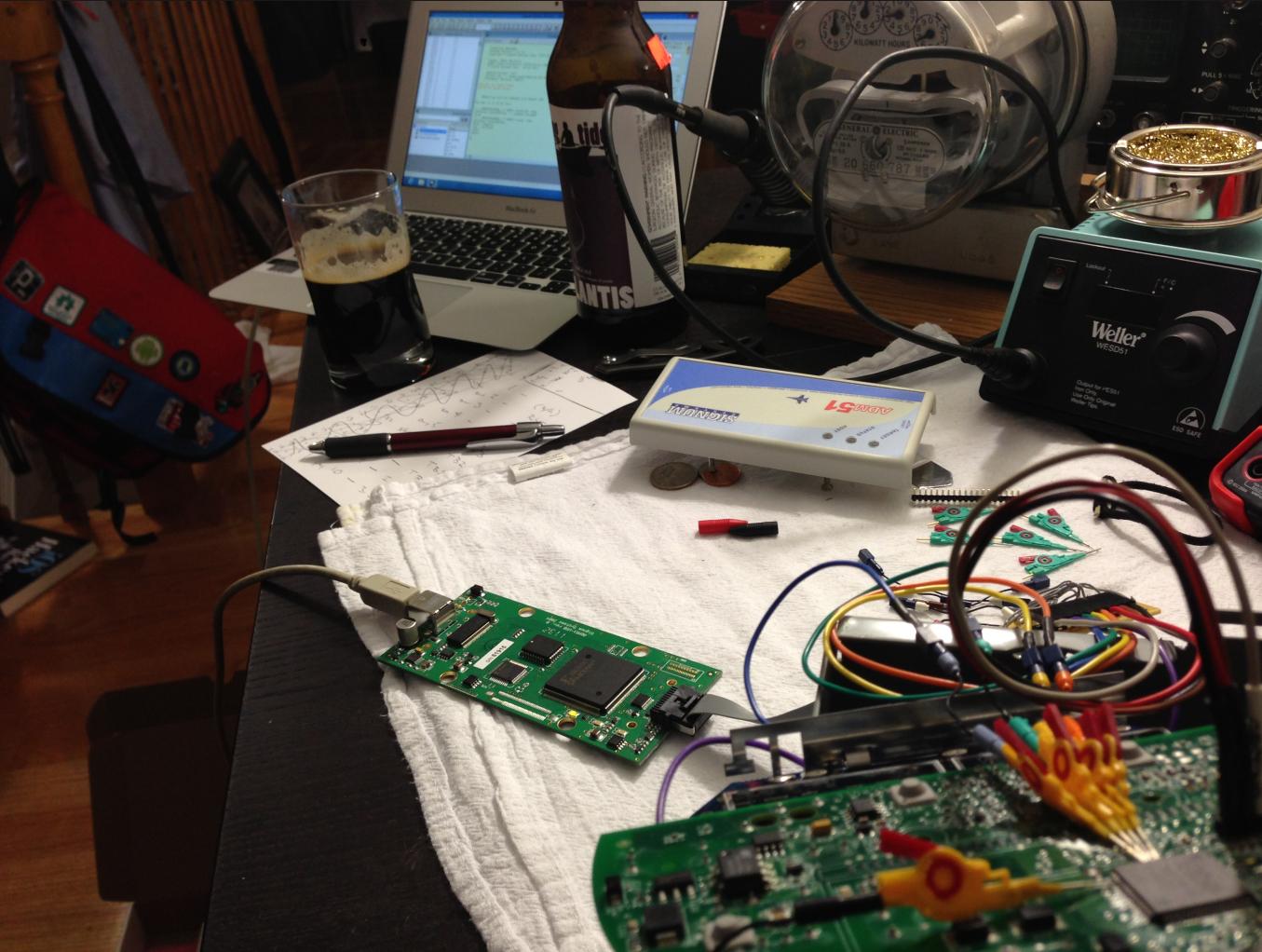
HARDWARE REVERSING

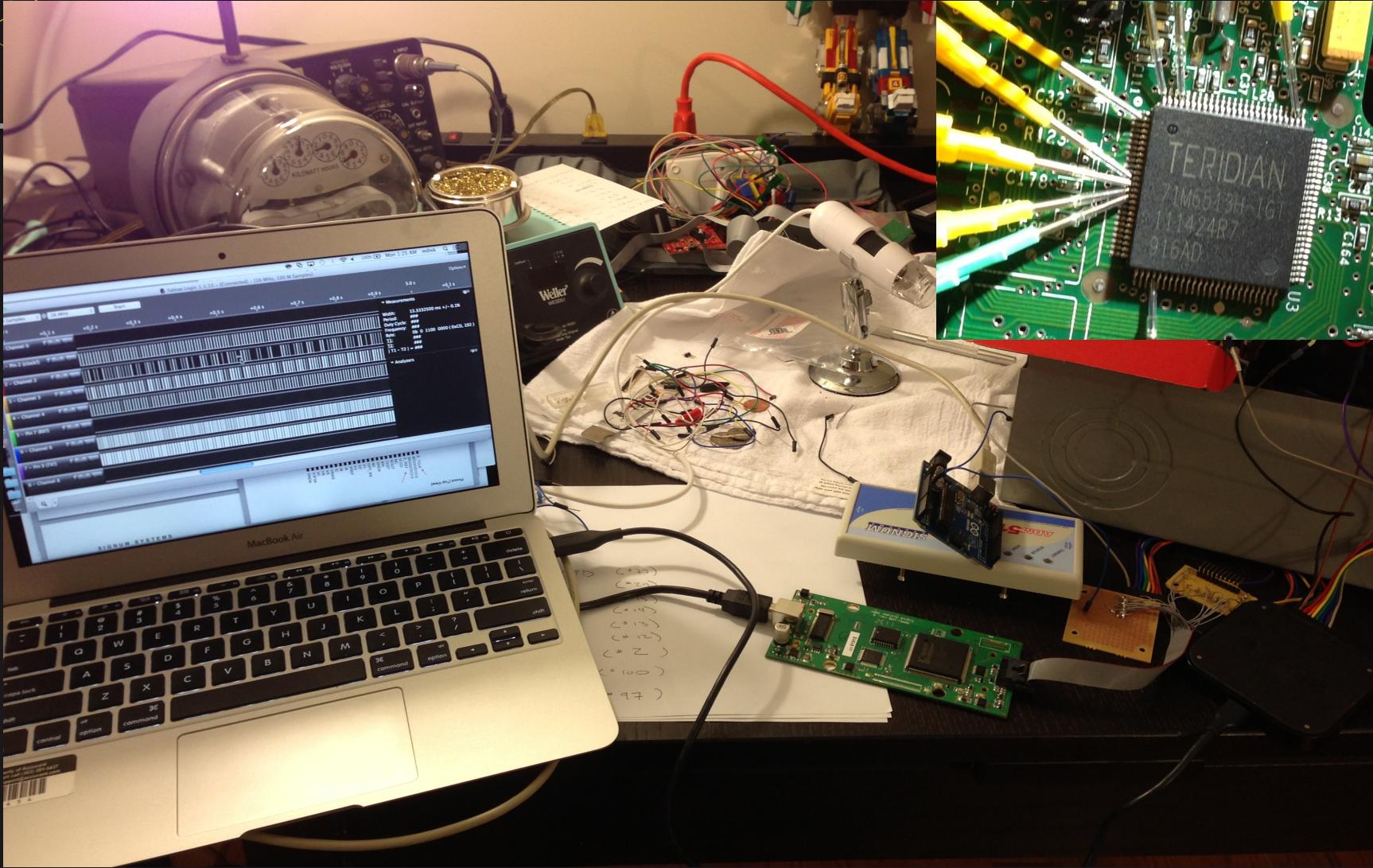


10 second primer

- Rubber soled shoes
 - (voltage.high != self.healthy)
- Logic Analyzer
 - { Saleae, USBee, ...}
- Debugging toys
 - {Goodfet, BusPirate, bootleg Segger tools, etc}
- Ground loops are bad, mmkay? Run your laptop on battery power*
 - *unless you hate your laptop







TERIDIAN

The brain in “smart meter”



Teridian 71M65xx

- Various Teridian chips are in a variety of locations, but the largest deployments seem to be in smart meters
 - Different, but related chips in pin pads, distribution automation, home automation
 - Teridian Semiconductor bought by Maxim Integrated a few years ago. Kept the brand name.
- Always metrology, but sometimes perform network support, too



Teridian 71M65xx

- When they aren't hooked up to a radio, attack surface is mostly local/physical access
- Active interfaces are an internal serial connection and external optical port
 - Occasionally EEPROM, infrequently a SPI slave port
 - (Find #winning here)
- When they talk directly to a radio, suddenly attack surface is much more interesting

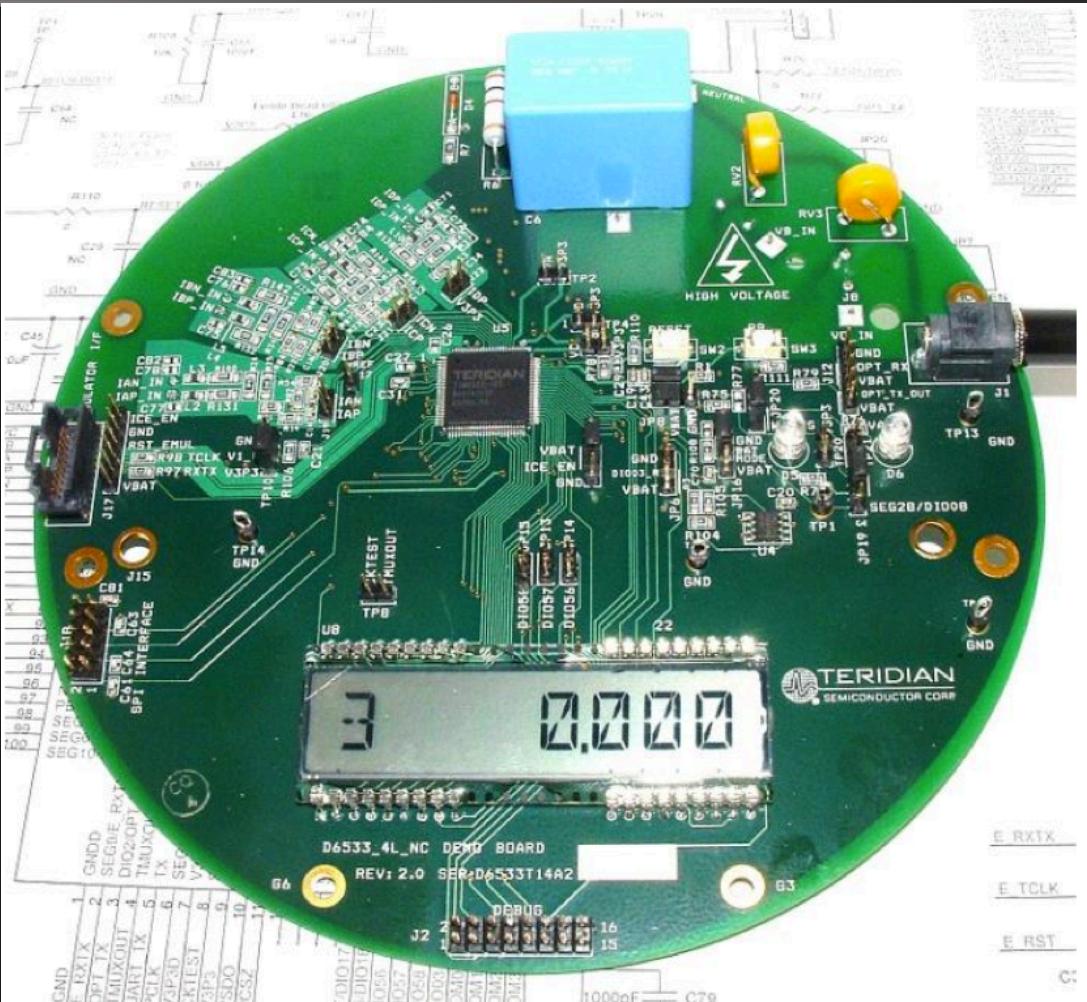


Teridian 71M65xx

- For each processor class, Maxim sells demo boards for ~\$250 that wrap a fake meter around a specific chip model
- These are extremely useful for getting moving quickly



Teridian 71M6533-DB

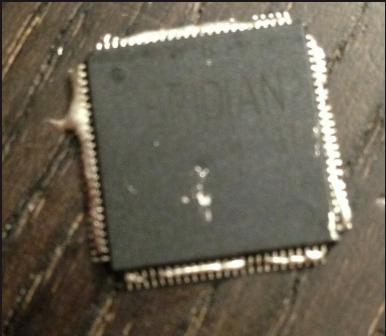
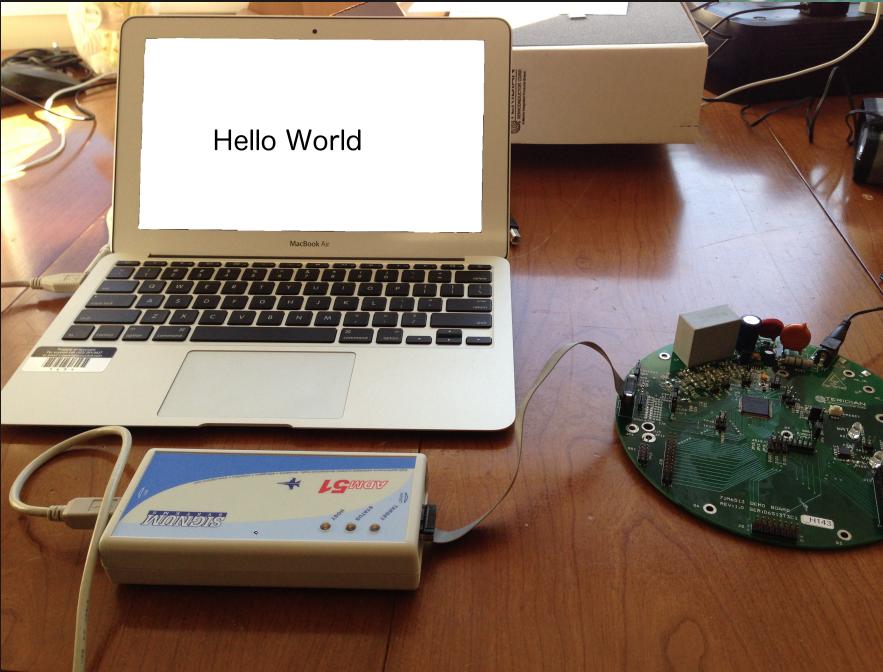


Teridian 71M6533-DB

- For a variety of reasons, it's painful to move code from a meter on to a demo board, but it's an option. If it works, your life just got easier.
- If you get too frustrated, sometimes it's worth it to just desolder the chip and pop it on the demo board.



ProTip: AllTheThings.move()



Teridian 71M6533-DB

- For tracing around the board, we use a hot air rework station to clear everything off. This allows for a high res image on a flatbed scanner.
 - And you'll get the chip off to drop onto a breakout board or a demo board.
- Meters are a dime a dozen. But these demo boards are painfully hard to get in stock, so we didn't destroy one to have pretty pictures...



ProTip: AllTheThings.KillWithFire()



Modifying a production PCB

- You'll have to make modifications to the PCB to enable debugging
- Just read the datasheet, it tells you what needs to be done. Tl;dr is usually to unground certain pins, disable the hardware interrupt by tying another pin to 3.3v, etc.



Order the right one(s)!

- Last, some chips provide code tracing functionality, but not all do. We bought the wrong one for this preso because our target proc was out of stock (\$%&\$!)
- If yours doesn't, try buying the closest chip that does
- You'll <3 tracing



Teridian 71M6533



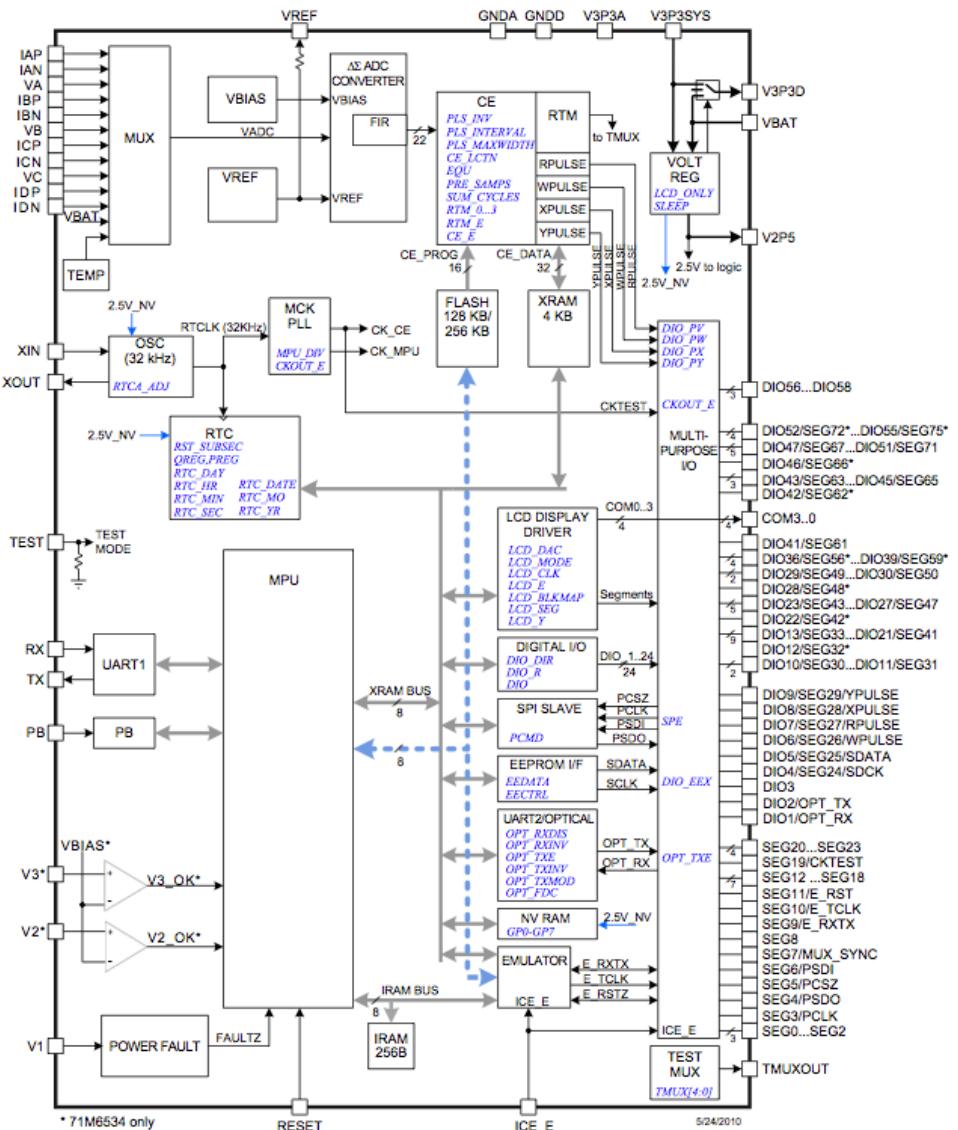
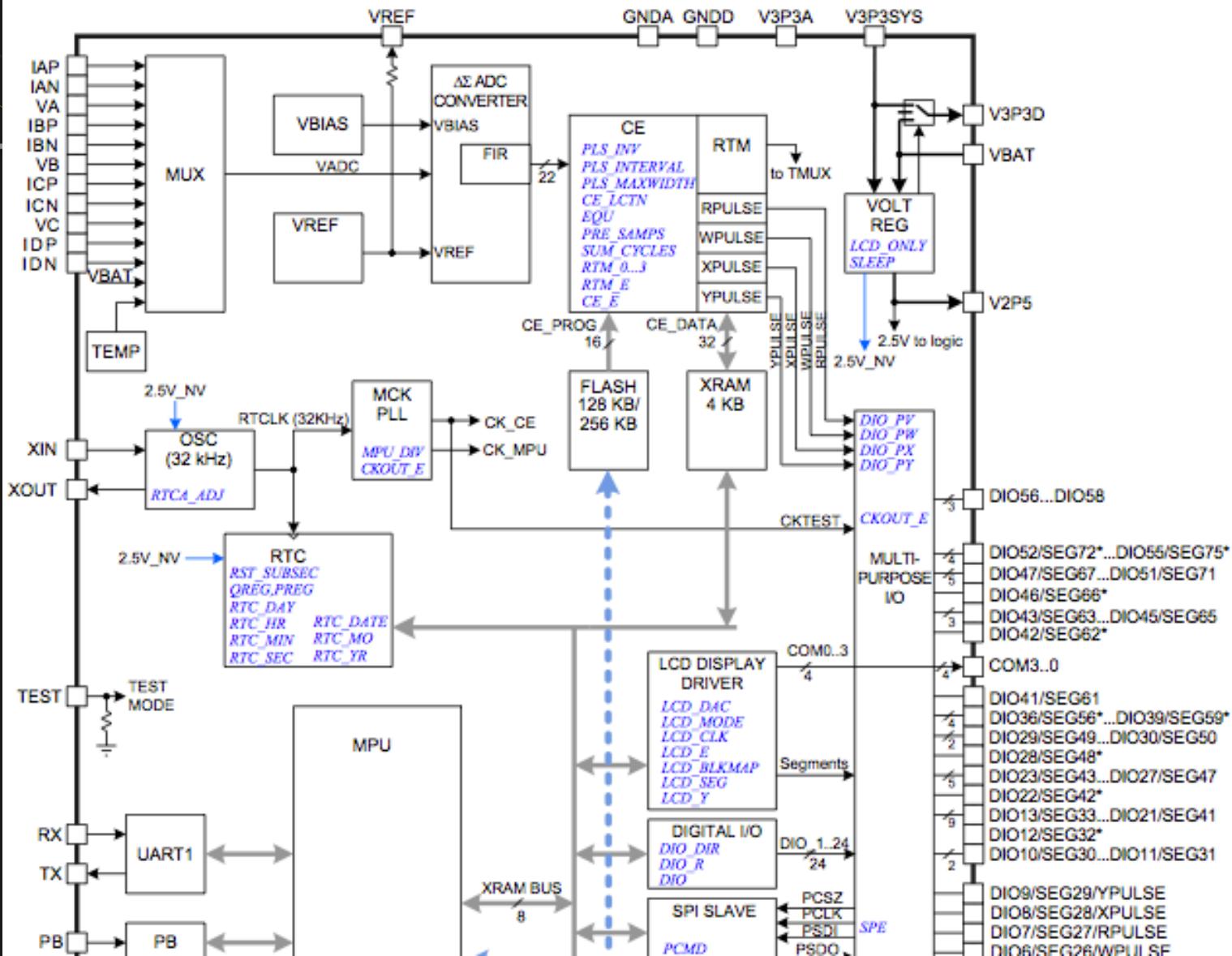
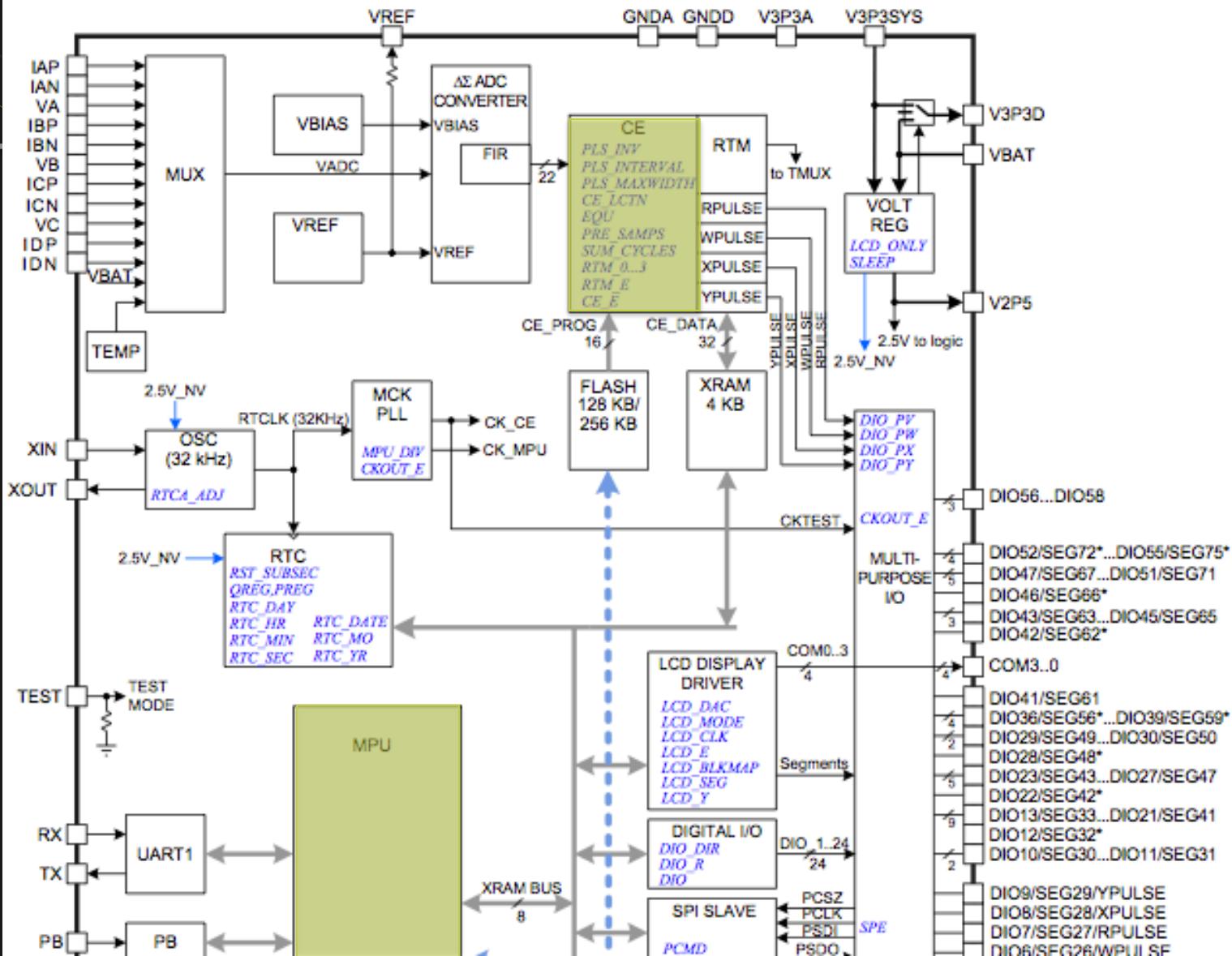


Figure 1: IC Functional Block Diagram







MPU & CE

- The MPU is an 8051-derivative, the 80515. Standard 8051 instruction set, just with fancy parallelization to speed it up
- Byte aligned instructions
- The Computation Engine is used to perform accurate mathematical operations on power measurement data



Teridian 71M6533-DB

- For our purposes, we don't care about the CE, other than that it'll step on your toes during exploitation because it shares memory with the MPU
- The 8051 core, memory, and the ICE components do everything we care about



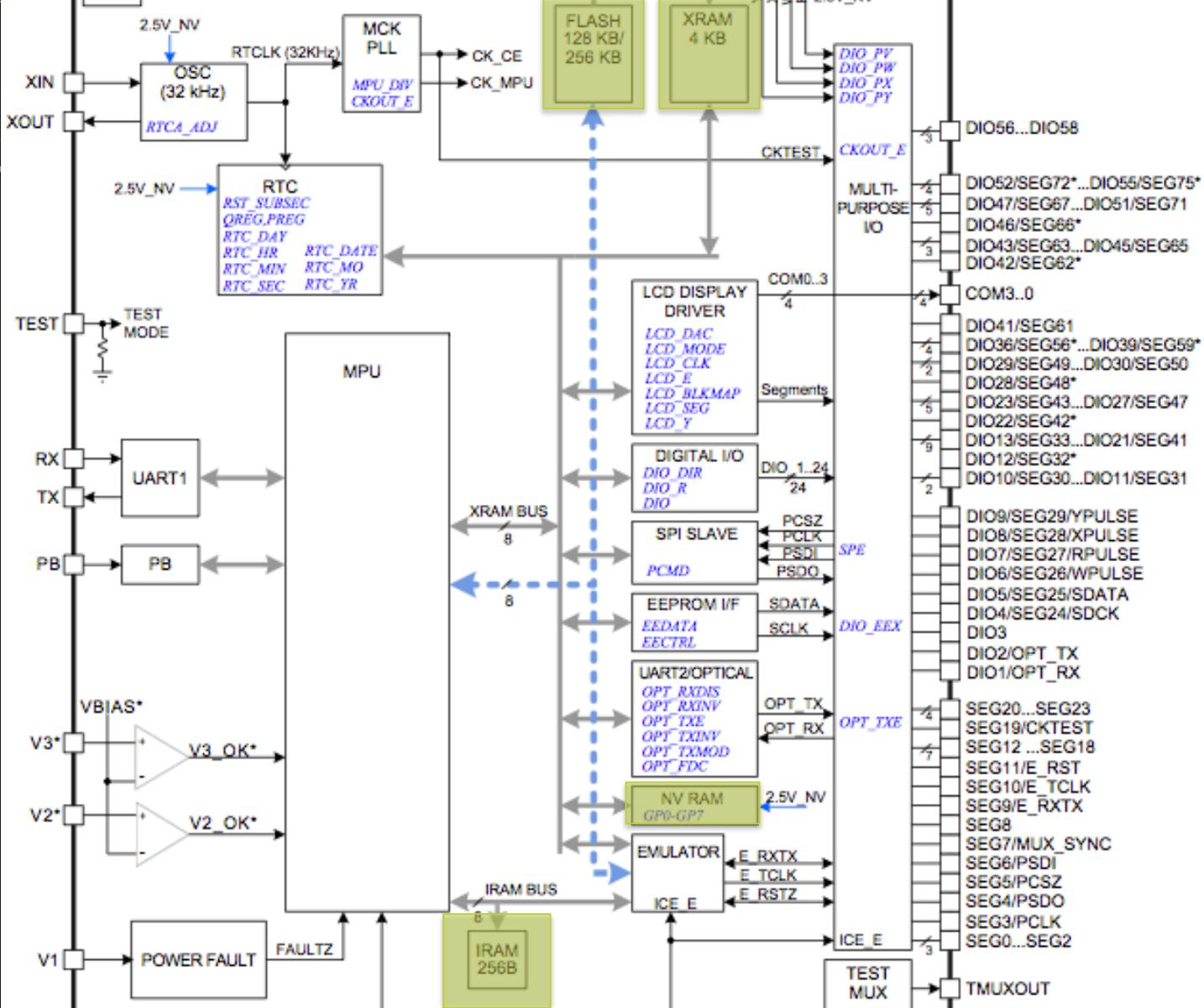


Table 7: Memory Map

Address (hex)	Memory Technology	Memory Type	Name	Typical Usage	Memory Size (bytes)
00000-1FFFF	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	128 KB
00000-3FFFF [†]	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	256 KB [†]
on 1K boundary	Flash Memory	Non-volatile	Program memory	CE program	8 KB max.
0000-0FFF	Static RAM	Volatile	External RAM (XRAM)	Shared by CE and MPU	4 KB
2000-20BF, 20C8-20FF	Static RAM	Volatile	Configuration RAM (I/O RAM)	Hardware control	256
20C0-20C7	Static RAM	Non-volatile (battery)	Configuration RAM (I/O RAM)	Battery-buffered memory	8
0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256



Table 7: Memory Map

Address (hex)	Memory Technology	Memory Type	Name	Typical Usage	Memory Size (bytes)
00000-1FFFF	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	128 KB
00000-3FFFF [†]	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	256 KB [†]
on 1K boundary	Flash Memory	Non-volatile	Program memory	CE program	8 KB max.
0000-0FFF	Static RAM	Volatile	External RAM (XRAM)	Shared by CE and MPU	4 KB
2000-20BF, 20C8-20FF	Static RAM	Volatile	Configuration RAM (I/O RAM)	Hardware control	256
20C0-20C7	Static RAM	Non-volatile (battery)	Configuration RAM (I/O RAM)	Battery-buffered memory	8
0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256



Flash Memory

00000-3FFF [†]	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	256 KB [†]
-------------------------	--------------	--------------	----------------	-----------------------------------	---------------------

- Where code and certain static data is stored
- Can't normally be modified
- Addresses only 64KB at a time
- Uses bank switching to address the rest



Flash is weird.

- You can't turn on individual bits in flash; you can only turn them off (sound familiar?)
- A mass erase of flash turns all bits 'on' (e.g. mem is filled with 0xFF); you write arbitrary bytes by selectively turning off bits
- You can only erase pages of flash, not bytes



So you wanted to change that?

- To make arbitrary changes to a portion of code memory, you have to:
 - Copy the entire page into XRAM
 - Make your modifications
 - Mass erase the page of flash
 - Turn off interrupts
 - Flip a bit that swaps out how the MOVX instruction works
 - Copy from XRAM back to flash a byte at a time



So you wanted to change that?

- Oh, and make sure the CE doesn't step on you and block your access to flash. You probably aren't monitoring the special flag that tells you that instruction just failed.
- And don't forget to check if a special bit is set that can block access to first X pages of flash (designed to block accidental erasure of bootloader code)



Did we say flash is weird?

- But, you *can* unset bits
- So in specific situations you might be happy to unset a few to change a particular value
- (See Travis' presentation from NoSuchCon in May)



Table 7: Memory Map

Address (hex)	Memory Technology	Memory Type	Name	Typical Usage	Memory Size (bytes)
00000-1FFFF	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	128 KB
00000-3FFFF [†]	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	256 KB [†]
on 1K boundary	Flash Memory	Non-volatile	Program memory	CE program	8 KB max.
0000-0FFF	Static RAM	Volatile	External RAM (XRAM)	Shared by CE and MPU	4 KB
2000-20BF, 20C8-20FF	Static RAM	Volatile	Configuration RAM (I/O RAM)	Hardware control	256
20C0-20C7	Static RAM	Non-volatile (battery)	Configuration RAM (I/O RAM)	Battery-buffered memory	8
0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256



IRAM

0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256
-----------	------------	----------	--------------	--------------------	-----

- Internal RAM is *small*
- SFRs, registers, some very small data structures

Table 8: Internal Data Memory Map

Address Range		Direct Addressing	Indirect Addressing
0x80	0xFF	Special Function Registers (SFRs)	RAM
0x30	0x7F		Byte addressable area
0x20	0x2F		Bit addressable area
0x00	0x1F		Register banks R0...R7



IRAM

0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256
-----------	------------	----------	--------------	--------------------	-----

- Dependent on compiler (and compiler settings), but the stack's usually here
- But because it's so small, variables aren't (at least with Keil)

Table 8: Internal Data Memory Map

Address Range		Direct Addressing	Indirect Addressing
0x80	0xFF	Special Function Registers (SFRs)	RAM
0x30	0x7F		Byte addressable area
0x20	0x2F		Bit addressable area
0x00	0x1F		Register banks R0...R7



Table 7: Memory Map

Address (hex)	Memory Technology	Memory Type	Name	Typical Usage	Memory Size (bytes)
00000-1FFFF	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	128 KB
00000-3FFFF [†]	Flash Memory	Non-volatile	Program memory	MPU Program and non-volatile data	256 KB [†]
on 1K boundary	Flash Memory	Non-volatile	Program memory	CE program	8 KB max.
0000-0FFF	Static RAM	Volatile	External RAM (XRAM)	Shared by CE and MPU	4 KB
2000-20BF, 20C8-20FF	Static RAM	Volatile	Configuration RAM (I/O RAM)	Hardware control	256
20C0-20C7	Static RAM	Non-volatile (battery)	Configuration RAM (I/O RAM)	Battery-buffered memory	8
0000-00FF	Static RAM	Volatile	Internal RAM	Part of 80515 Core	256



External RAM

- Variables go here, usually
- Because this is what your operations interact with, typical vulns will give some sort of control of XRAM
- But that sucks, because registers aren't here, and pointers are rare



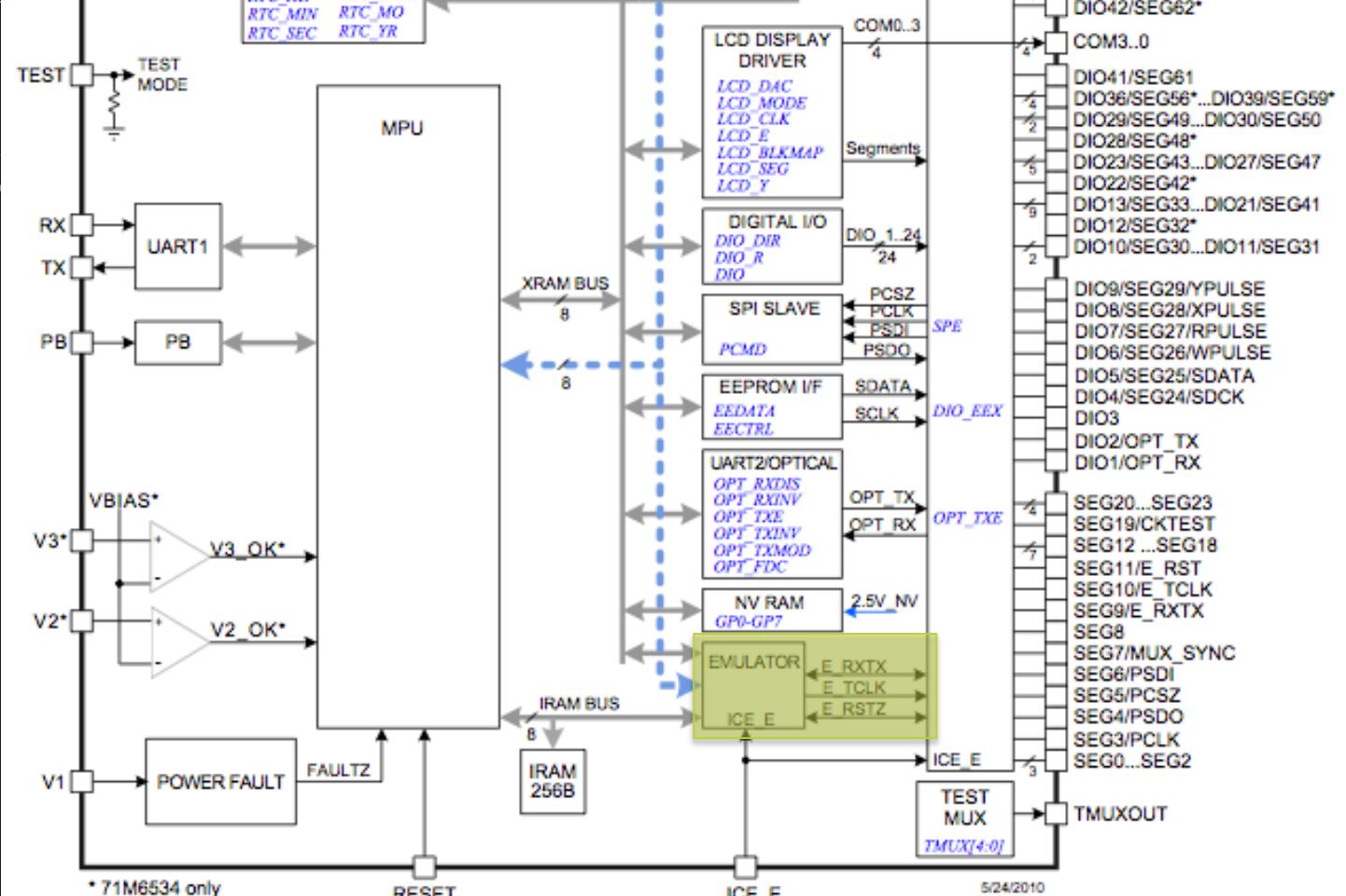


Figure 1: IC Functional Block Diagram



ICE, baby

- Teridian uses a proprietary, undocumented debugging interface
- Requires you to buy a \$1200 device to debug



The \$1200 FPGA



ICE, baby

- Doesn't turn on until the “preboot” phase of execution is complete.
 - Either the first 30 or 60 instructions, depending on chip
- Turned off by setting a disable bit during the preboot phase
- On by default, unless you explicitly turn it off



ICE, baby

- The PinPad SoC's security enable bit is more like a JTAG fuse
- Set it once, it can never be unset (or so the docs say, we've not looked yet)
- The smart meter SoCs chose not to implement this as a permanent switch, so it's enabled by default at every RSET



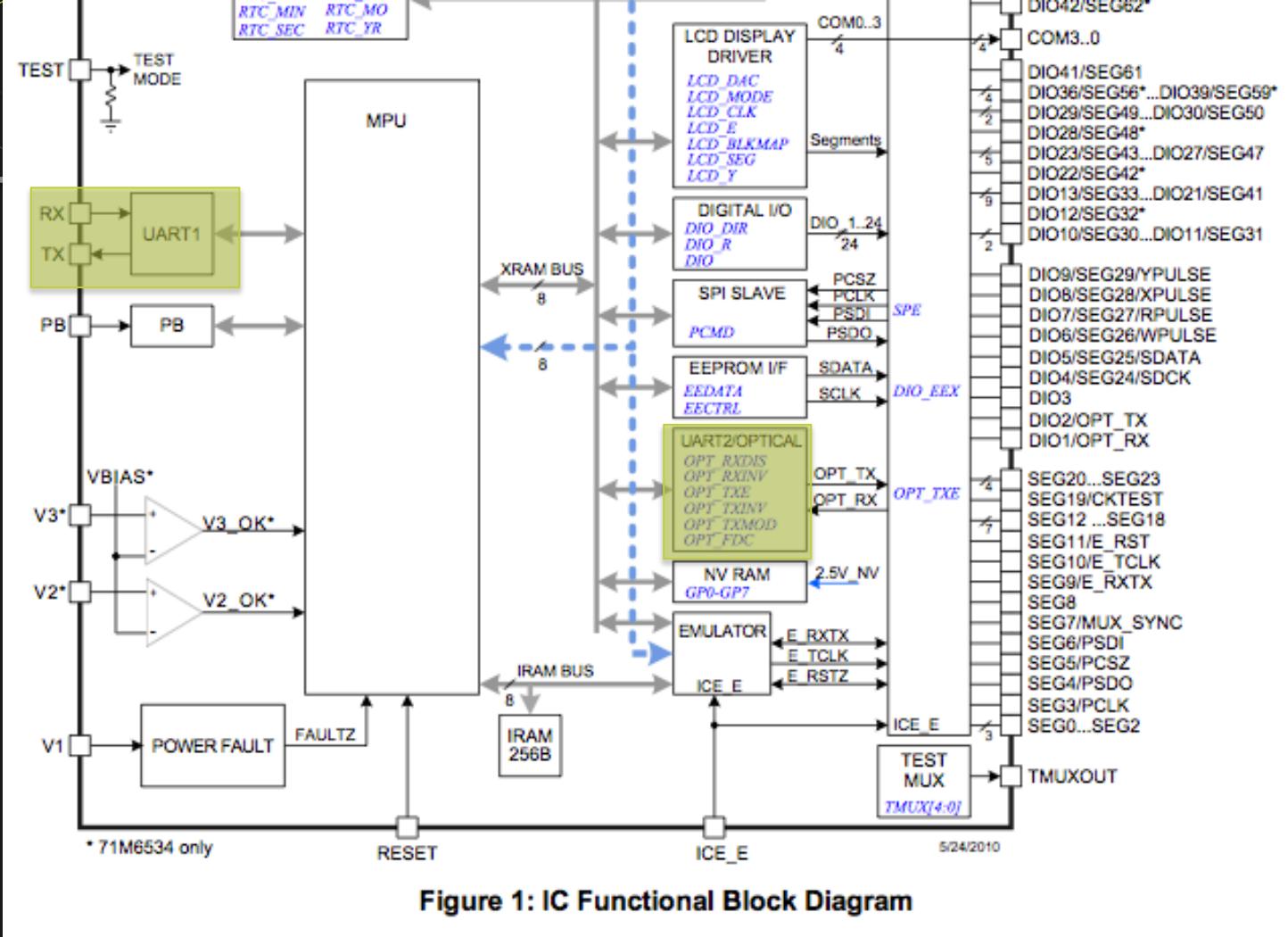


Figure 1: IC Functional Block Diagram



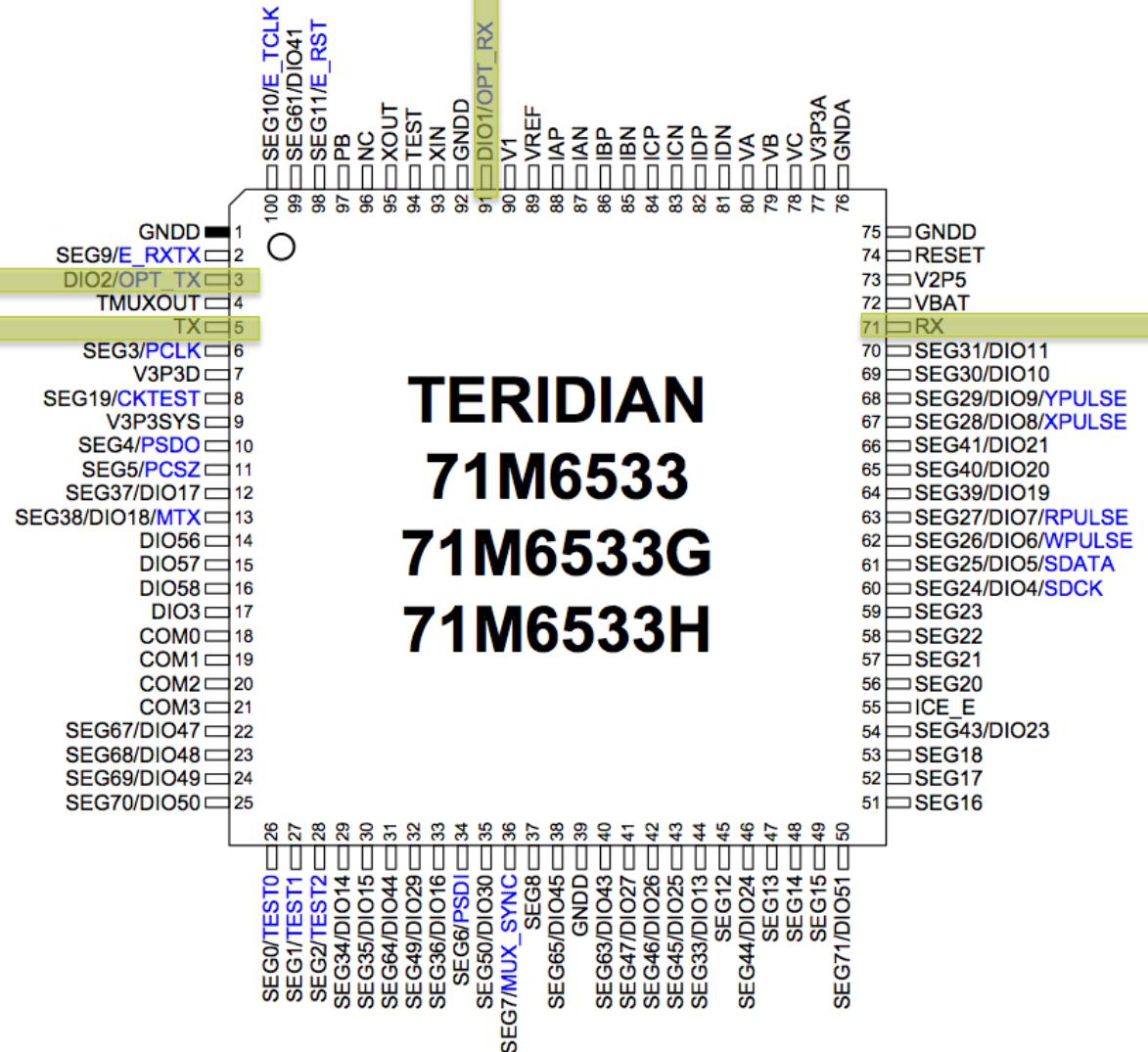


Figure 53: Pinout for 71M6533/71M6533G/71M6533H LQFP-100 Package



Interfaces to the world

- Here be gremlins
- Any ‘remote’ exploitation will occur over these interfaces
- Usually used to talk to a daughter card handling data transmission, or sometimes a radio chip itself



Interfaces to the world

- Drives the optical port on the outside of a meter
- See cutaway's talk on optical serial and ANSI 12.18/19 from Black Hat 2012
 - <http://www.blackhat.com/usa/bh-us-12-briefings.html#Weber>
- Basically, any bugs found here will probably also be viable over network transport that terminates on the internal serial port





SOCSPLOITS AHOY!



FCTRL[6] //SECURE

0xB2[6]	<i>SECURE</i>	R/W	Enables security provisions that prevent external reading of flash memory and CE program RAM. This bit is reset on chip reset and may only be set. Attempts to write zero are ignored.
---------	---------------	-----	--



FCTRL[6] //SECURE

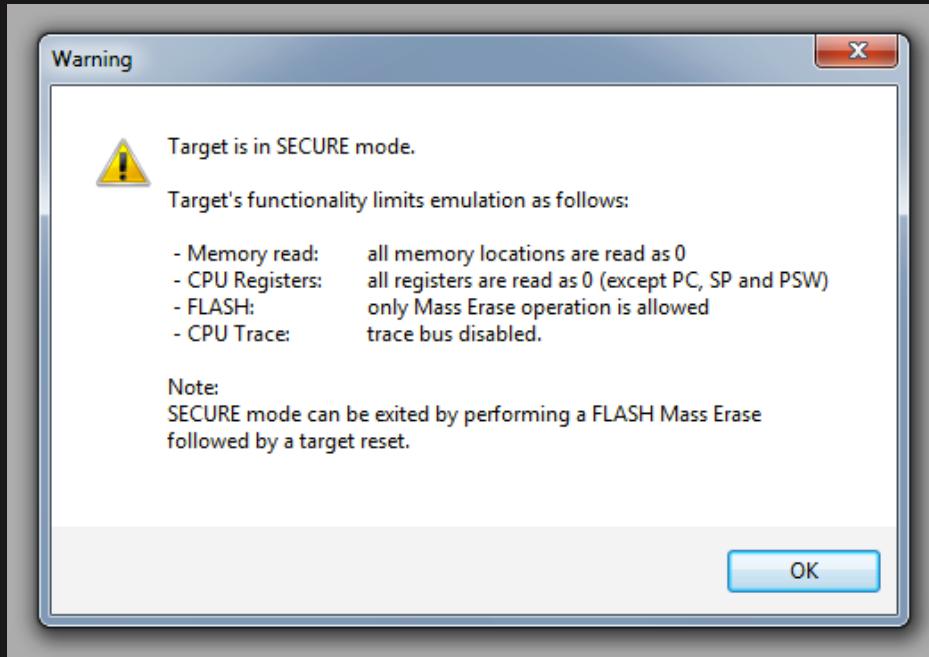
- FLASH:

only Mass Erase operation is allowed

- Mass Erase sets all of flash to 0xFF
- Did... they remember to clear RAM, too?

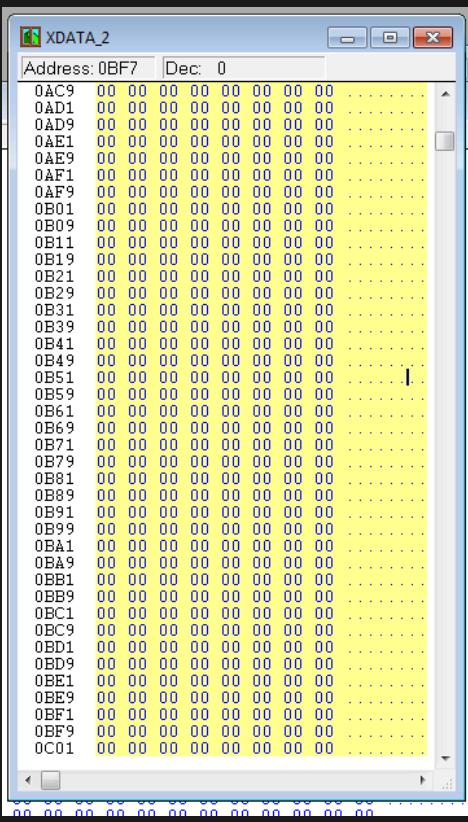


FCTRL[6] //SECURE

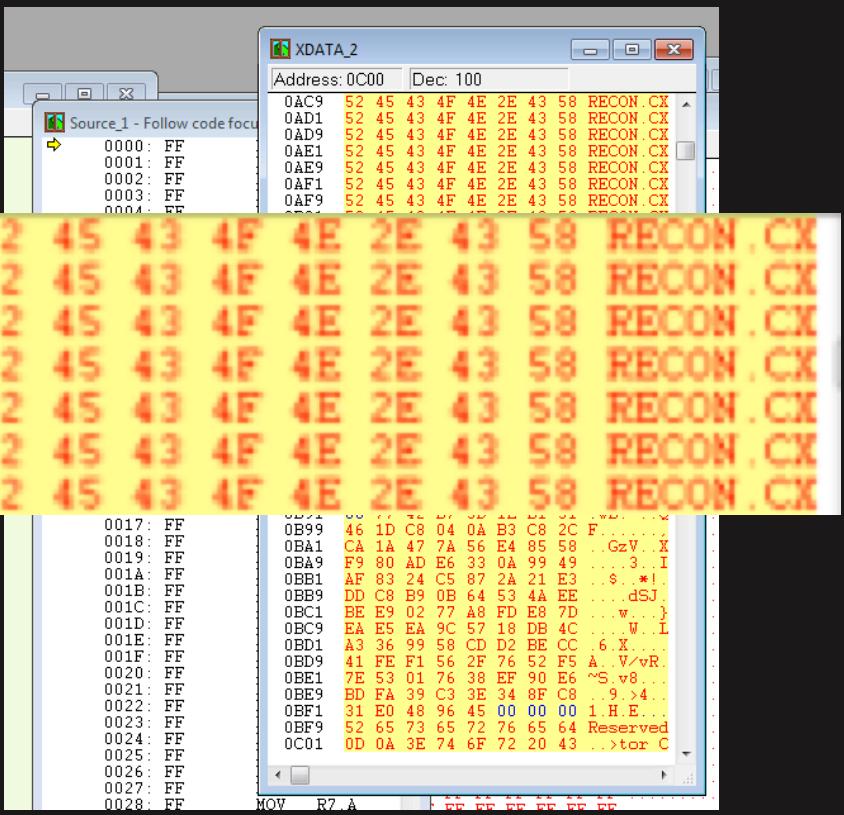


FCTRL[6] //SECURE

FCTRL⁶ == 1



Mass Erased,
FCTRL⁶ == 0



FCTRL[6] //SECURE

- XRAM (and IRAM, but that doesn't really matter) is accessible once flash has been mass erased and hard reset.
- Power's never lost to XRAM, so data is all still present.
- Did you store your crypto keys in data or code?



Gaining code exec

- Remember, code has separate physical locations for code, ram, registers
- There is no concept of executing RAM.. the instruction pointer only ever reads from flash



RESIDENT CODE EXECUTION



Gaining code exec

- Because the stack's in idata and variables are in xdata, normal vulnerabilities are extremely difficult to leverage
- Even after you get execution pointer control, you don't have any real control of the stack



Gaining code exec

- Take a standard overflow. The vuln grants the ability modify XDATA, but pointers and registers are stored in IDATA.
- This isn't necessarily an 8051 thing, just Teridian specific, and an optimization put in place by the most popular compiler for this space (Keil)



Gaining code exec

- Also, because of all these architectural reasons, the compiler knows things won't move around and hard links almost all pointers at compile time



All is not lost

- The ANSI smart meter standard uses concepts of “tables” for data storage, which very well could be implemented as large arrays. The spec tells you what ranges are valid.
- Of course, it’s a common programming error to code only expecting input the spec says you’ll see.



All is not lost

- So, certain types of vulns allow for extreme control of XDATA

```
uint16x_t data table_idx;  
uint16x_t data value;  
uint16x_t data table[];  
//populate table_idx from packet  
table[table_idx] = value;
```



All is not lost

- Assume you can write to arbitrary locations in XRAM
 - (the same type of read vulns could be common too, of course)
- What to write where?



Finding a pointer in a haystack

- The sample code for the demo board contains a great example target
- It implements a software timer interrupt (STM) that allows functions to be executed after a given number of cycles have occurred



Function pointers in XDATA

- This seems to be a common tactic to extend limited hardware timers into several software timers, and should be present across many implementations
- Between modifications to the STM pointers array and timeout arrays, one can gain the ability to execute any location in code memory (flash).



OK, now what?

Our constraints:

- code segment [flash] can't be modified... easily
- impossible to execute data in any other location
- STM implementation only executes functions in the first 64K of memory
 - (real life ones wouldn't necessarily have this restriction, but are more complex to exploit due to banking issues)



OK, now what?

- But in the real world, code blocks exist to do most of the heavy lifting
- You can “ROP” because of the 1-byte alignment, but on Teridian, that’s quite a bit harder
- Just try to leverage the boot loading / reflashing code that is present in some form on every deployed meter



Leveraging STM

```
// DESCRIPTION: 71M652x POWER METER - software timers
// This multiplexes a hardware timer to make many software timers.
// The interface is made to be like that of the hardware timers.

void (*fn_ptrs[ STM_COUNT ]) (void);

uint16x_t * stm_start (uint16_t tick_cnt, uint8_t restart, void (*fn_ptr) (void))
{
// ...
    fn_ptrs[ stm_index ] = fn_ptr;
// ...
}

void stm_run (void)
{
// ...
    if (fn_ptrs[stm_index])           // if there's an expire function.
}                                *(fn_ptrs[stm_index])) (); // call it
```



Leveraging STM

```
CLI/io.c:      pstm = stm_start(milliseconds(30000), 0, io_timer);  
CLI/ser0cli.c:           free_cnt_ptr = stm_start(XMIT_TIMEOUT, 0, ser0_free_timer);  
CLI/ser1cli.c:           free_cnt_ptr = stm_start(XMIT_TIMEOUT, 0, ser1_free_timer);  
IO/eeprom.c:      pstm = stm_start(write_tick_cnt, 0, issue_START);  
IO/eeprom.c:      if (NULL == stm_start(write_tick_cnt, 0, issue_START))  
IO/eeprom.c:      if (NULL == stm_start(write_tick_cnt, 0, issue_START))  
IO/eeprom.c:      if (NULL == stm_start(write_tick_cnt, 0, issue_START))  
IO/eeprom.c:      stm = stm_start(milliseconds(7000), 0, NULL);  
IO/tmr0.c:// coded to no absolute registers because it's called by stm_start,  
LCD_VIM828/lcd_vim828_ext.c:      mux_stm_ptr = stm_start(MUX_DELAY, 0, mux_out_timer);  
Meter/ce_30.c:    stm_start(milliseconds(1000), 0, ce_start); // start a software timer  
Meter/meter.c:    ptimer = stm_start(100, 0, meter_timer_fn);
```



Leveraging STM

- This creates an array of function pointers stored in XDATA that are called very frequently
- Static address for a given compile
- Compute index value to feed the vulnerable function to overwrite this array with arbitrary content



Execution via STM

- The “stack” is logically split via compiler hard links to data and the real stack
- This means you have to find the locations for variable storage in XRAM, modify them directly, then lastly modify the STM pointer and pray ☺



Execution via STM

- With STM-style pointer execution, you have exec, but no contextual information about variables/registers/etc
- Painful. My code crashed a lot. Lots of tricks to make it more reliable, though
- End state, resident code exec against this demo meter software



Demo.new()



GLITCHING



Killing the Secure Bit

- Preboot is a hardware count of <= 60 cycles
- Teridian suggests enabling SECURE at the front of Preboot

```
STARTUP1:  
    CLR    0xA8^7      ; Disable interrupts  
    MOV    0B2h, #40h   ; Set security bit.  
    MOV    0E8h, #0FFh   ; Refresh nonmaskable watchdog
```

- Translates to real life ASM:
 - [vdd supplied] -> ljmp, ljmp, [set Secure Bit], whatever



Constraints

- 10Mhz
- Have to hit cycle 3
- Don't care what happens after cycle 3
- Probable we can stomp over everything with GoodFet
- Probable we can target the 3rd cycle with the Datenkrake



The Goodfet approach



The Datenkrake approach

- $1 \wedge (\circ - \circ \wedge)$
- $(\neg \circ \square \circ) \vee \sim 0$



Thanks, greetz, l33ts

cutaway, atlas, travis goodspeed, drspringfield

Questions?

Nathan Keltner

@natronkeltner

nkeltner@accuvant.com

Josh Thomas

@m0nk_dot

jthomas@accuvant.com





ACCUVANT

THE AUTHORITATIVE SOURCE FOR INFORMATION SECURITY

1125 17th Street, Suite 1700, Denver, CO 80202

800.574.0896

sales@accuvant.com

www.accuvant.com

fin

