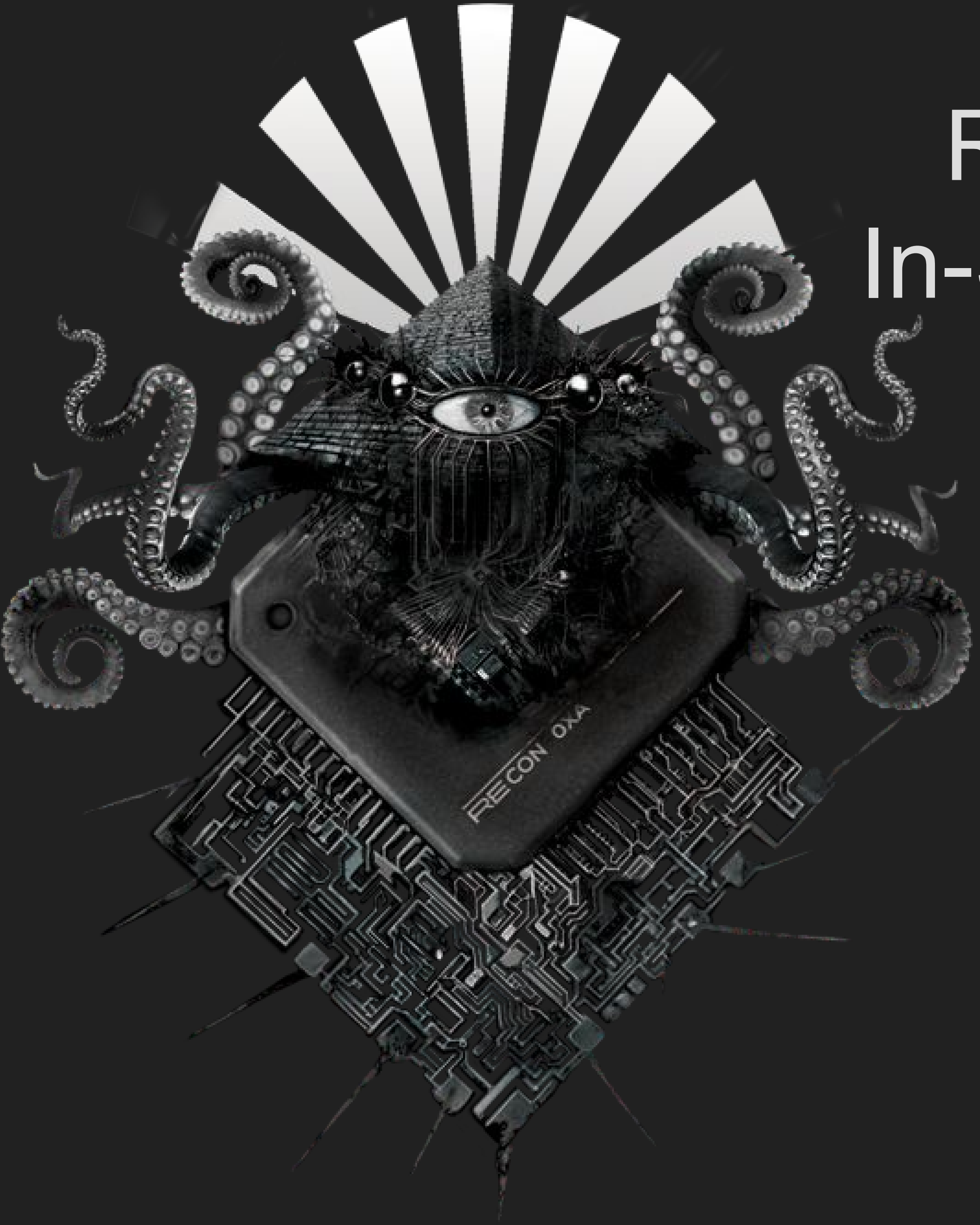


Reverse Engineering In-System-Configuration Controllers

Jessy Diamond Exum (diamondman)

Initial Project
3-4 year in the making
7400 logic based processor

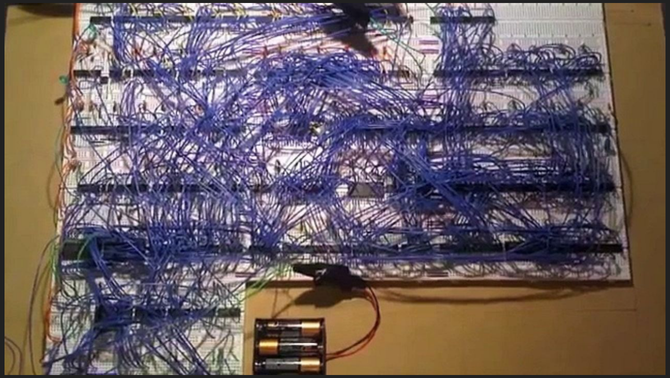
Agenda

1. An attempt to build a Processor (and how it ended in flames)
2. A Walkthrough of reversing the Digilent & Xilinx JTag Controllers
3. A New Hope: Generalizing Controller Access
   (efficiently speak to devices with any type of controller)
4. Questions

Enter the rabbit hole

Wanted to make a processor

- 7400 logic (informed by college)
- Took Coursera class (Computer Architecture by David Wentzlaff)!!!!!!!!!
- Logic board hell not worth it
  - boards were huge, power hungry, read only
- Solution: FPGA
- Problem: FPGA

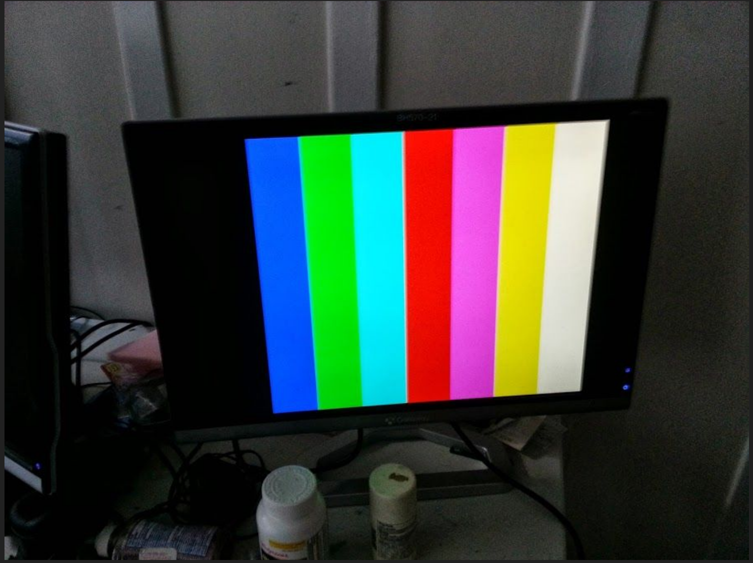


Via http://store.digilentinc.com/

Prerequisites

Xilinx Spartan 3e dev board Digilent

- Ethernet, VGA, PS2 (keyboard), decent price
- USB plug and play (Jtag controller built in).
- Did blinky light examples with schematic capture
- Xilinx tools not great at schematic capture. (note powerpoint vs photoshop)
- Learned Verilog: an HDL (words not schematic, faster to work with, industry standard) Lessons+samples: www.asic-world.com/verilog/

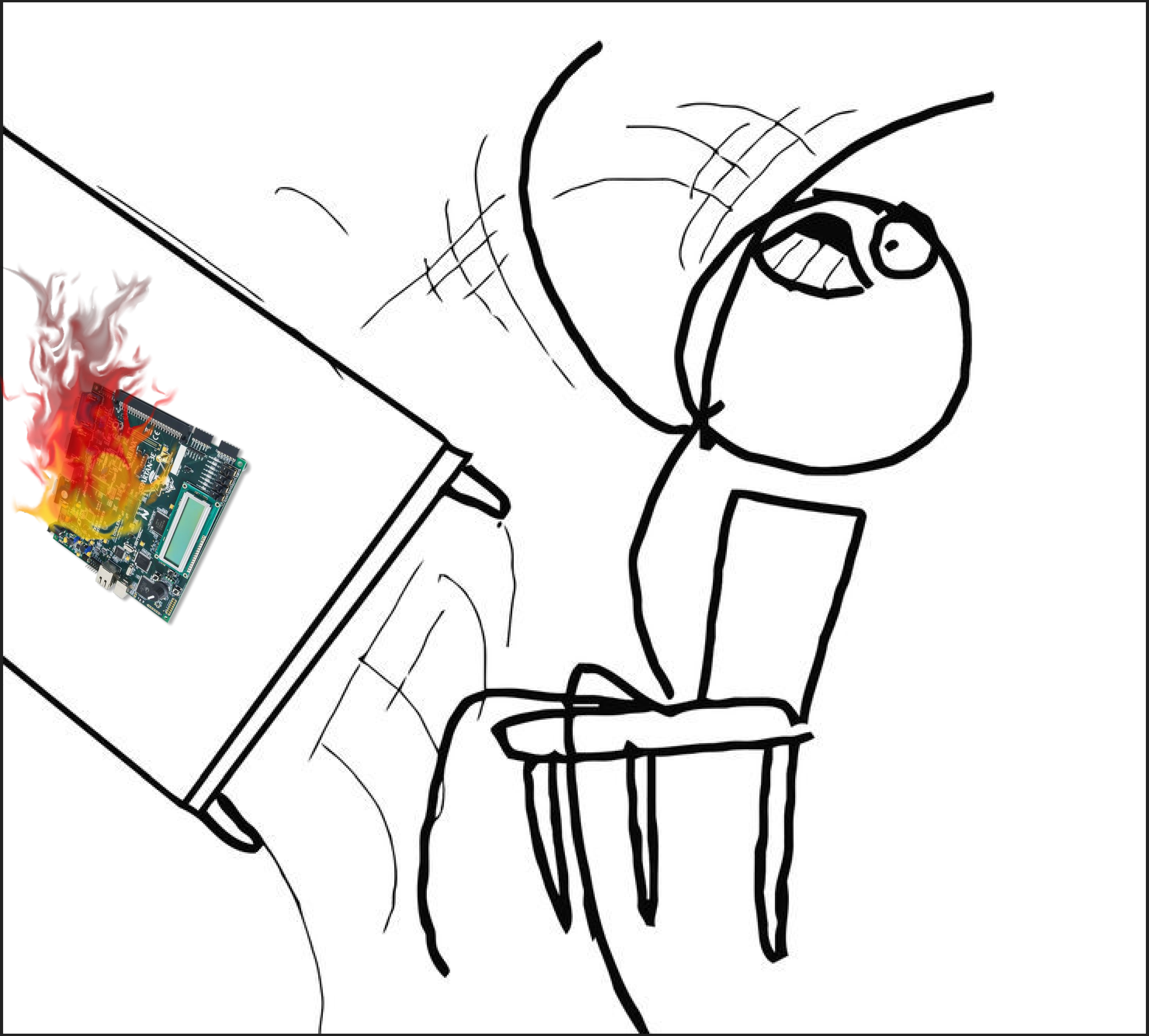


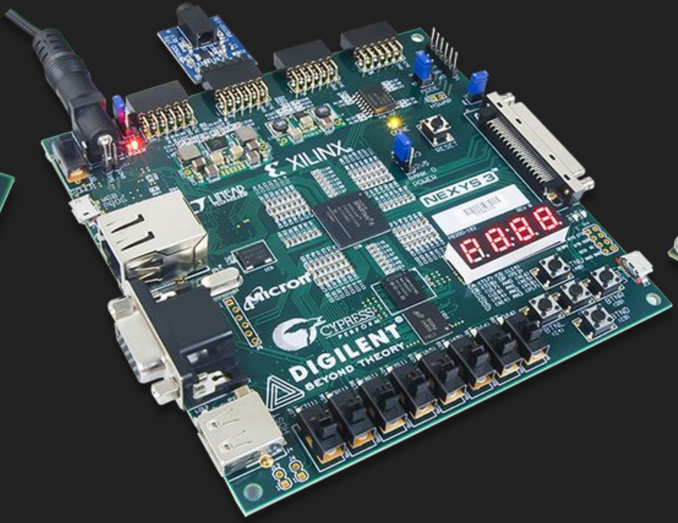
Getting Started

- Wanted to write a video driver (to control VGA monitor), because it is cool. ~one week. First time implementing electrical protocol (ADD FB post)
  - Challenge 1: Story about pixel by pixel not working/clock limitation.
  - Challenge 2: Xilinx's configuration tool (impact) only worked on Windows, Linux kernel 2.5 and older, and libusb drivers would not load (mystery at the time).

The Slippery Slope

- Load failure because Xilinx’s iMPACT manually loading libusb from centos location. Different in Debian. Not using LD to do it automatically.
- LD_Preload, and remaining issues
- Challenging to debug/reverse engineer because proprietary, 15 gigs of binaries, C++/.net/java, and against EULA
- Considered switching vendors, e.g. Altera, except **they were all broken.**
- Not that it would matter…





Section II

No board, no plan, time to reflect.

Bought several progressively better Digilent boards:

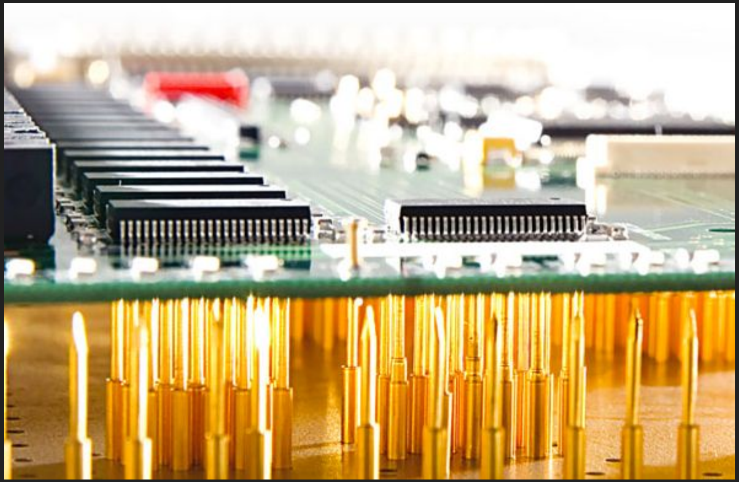
- Coolrunner 2 Starter Board (XC2C256) – cheap CPLD
- Basys 2 (Spartan 3e) – low end FPGA
- Nexys 2 (Spartan 3e) – low end, but better board
- Nexys 3 (Spartan 6) – Intermediate board
- Atlys (Spartan 6) – high end chip and board

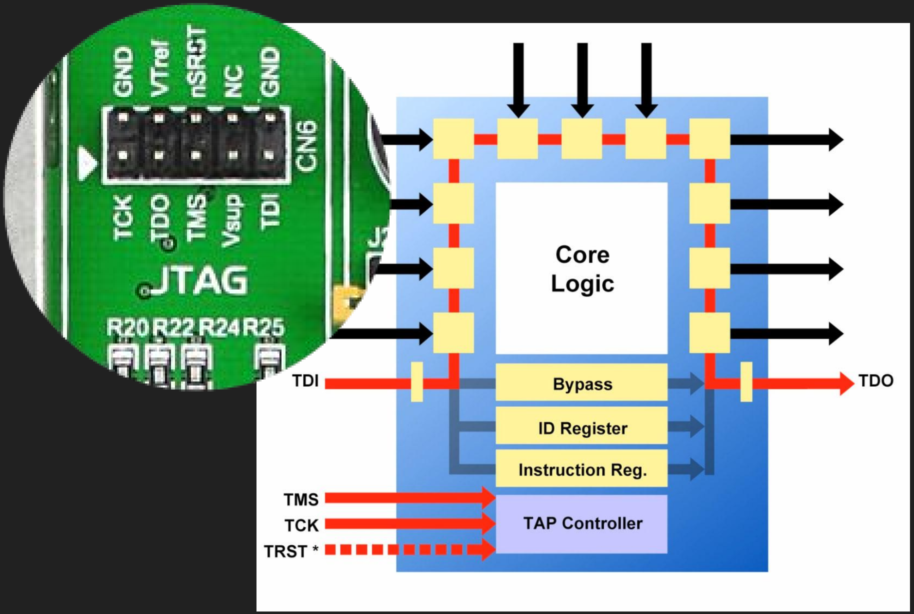
New (slightly irrational) Goals...

- Make open tools to compile HDL and flash chips
- Must work with Linux
- Support multiple Digilent boards

Which means… I needed to know how Digilent’s boards worked:

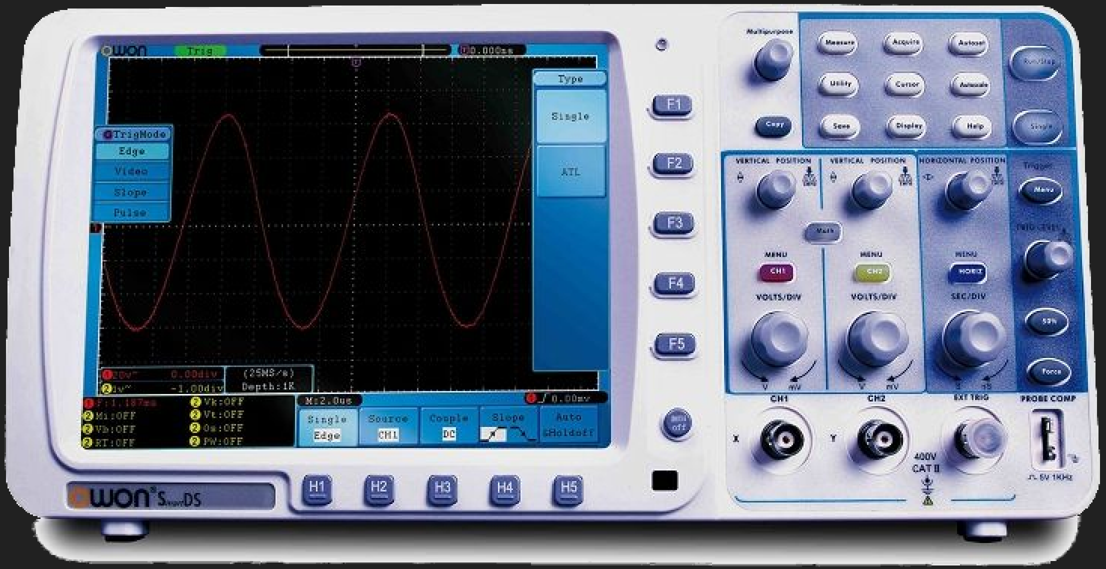
- Programming Xilinx Chips
  - Jtag & oscilloscope
- Digilent USB Commands
  - usb, wireshark





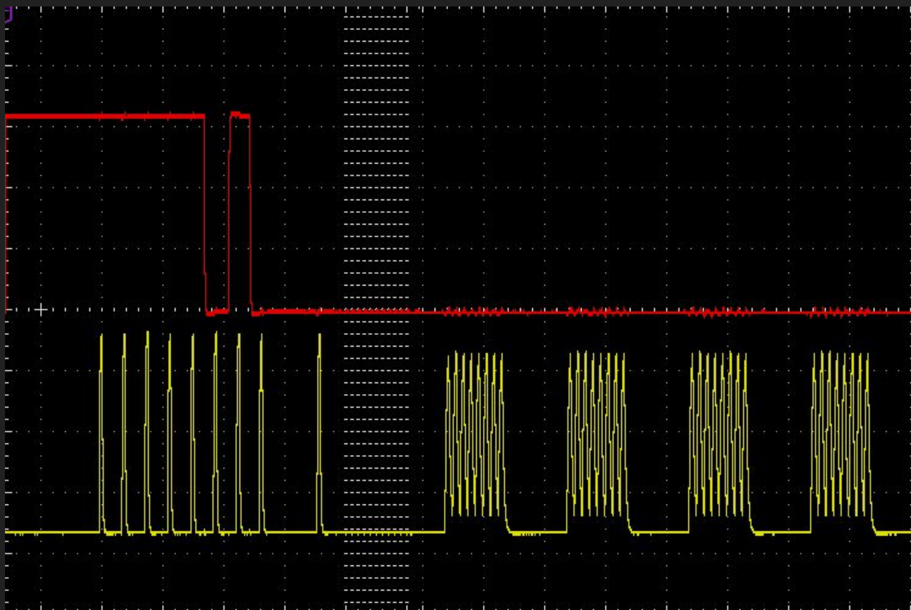
What the Jtag is Jtag?

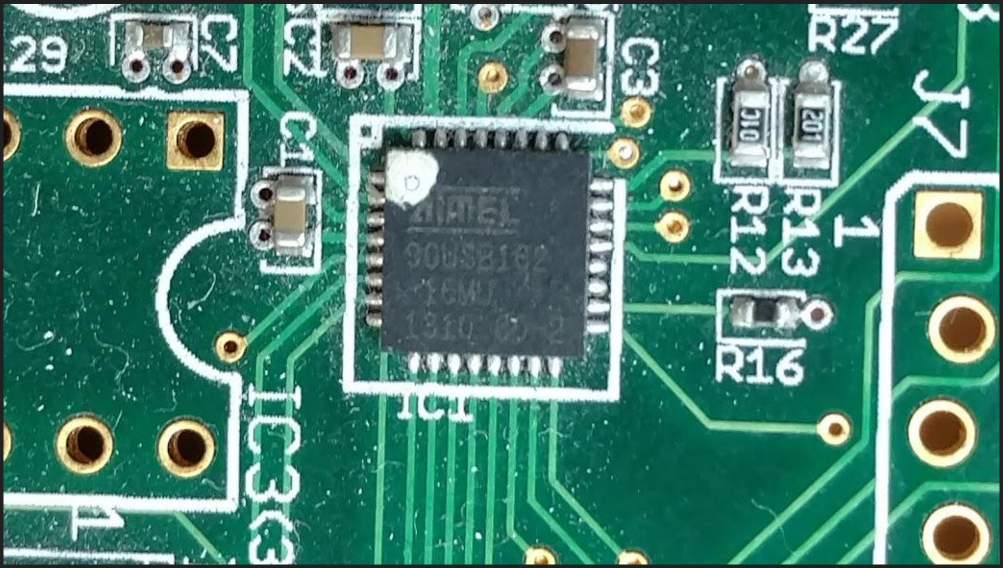
- Thought JTAG was just a programming protocol
- Found youtube videos (EEVBlog) on history http://youtu.be/TlWILeC5BUs
- What is ISC?
- Highly Extensible. Pros and cons.
- Has auto detect
- TMS pin control chip state
- Learned init process

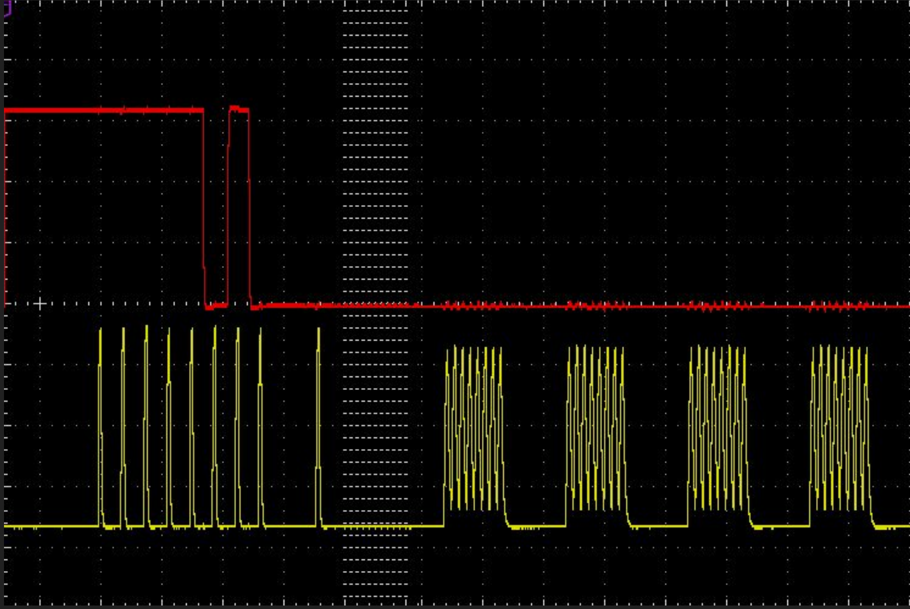


Observing the programming:

- Had an oscilloscope (Amazing purchase)
- Probed Clock and TMS
- Detected board (Adept)
- Captured Waves
- Waveform was correct

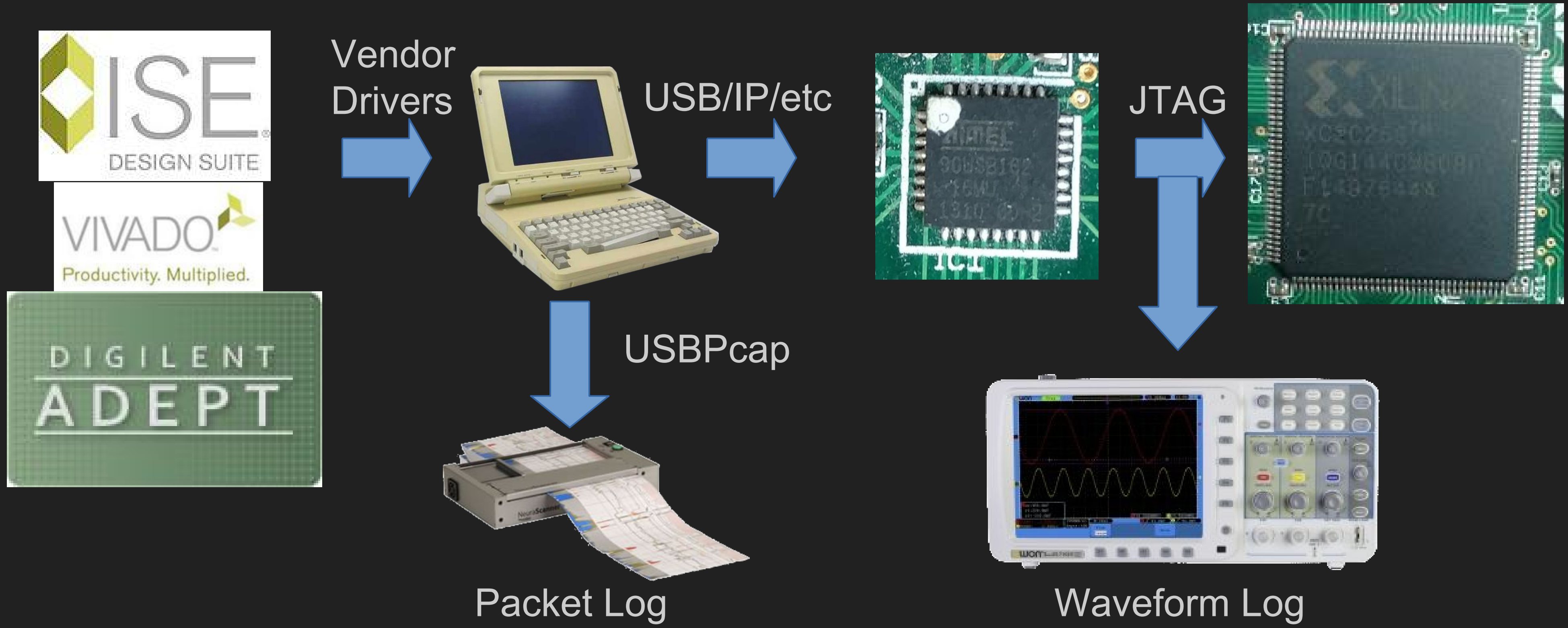






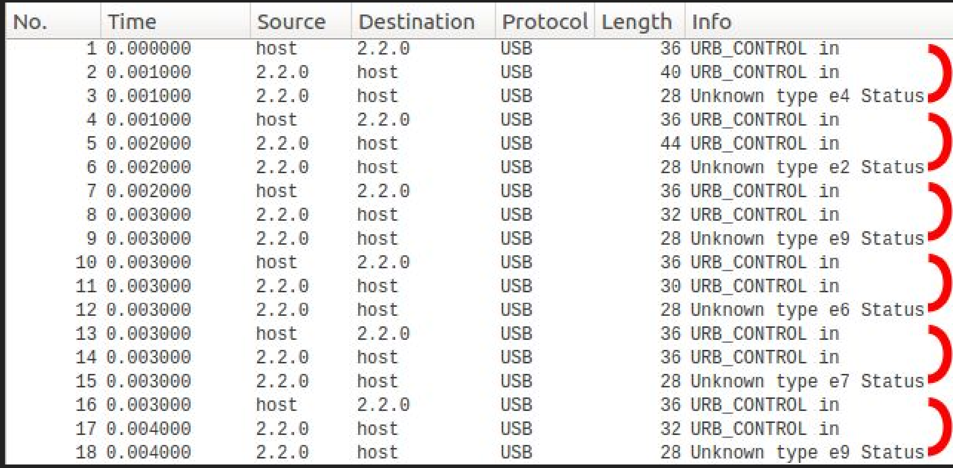
Learning how to initialize the board

- Already have waveforms
- Need to know how to control board
- USB->controller->JTAG
- Wireshark



Inducing Cause and Effect:

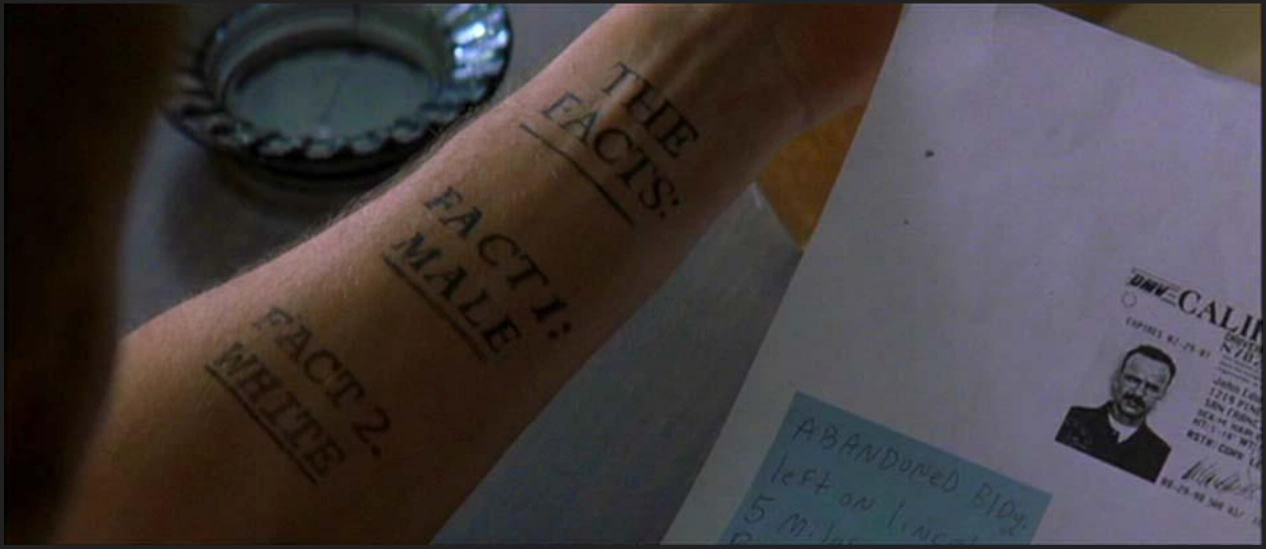
- Captured packets of JTAG initialization
- Python packet replay
  - Concerns
  - Results



```
No.   Time       Source   Destination  Protocol  Length  Info
    1 0.000000   host     2.2.0        USB           36  URB_CONTROL in
    2 0.001000   2.2.0    host         USB           40  URB_CONTROL in
    3 0.001000   2.2.0    host         USB           28  Unknown type e4 Status
    4 0.001000   host     2.2.0        USB           36  URB_CONTROL in
    5 0.002000   2.2.0    host         USB           44  URB_CONTROL in
    6 0.002000   2.2.0    host         USB           28  Unknown type e2 Status
    7 0.002000   host     2.2.0        USB           36  URB_CONTROL in
    8 0.003000   2.2.0    host         USB           32  URB_CONTROL in
    9 0.003000   2.2.0    host         USB           28  Unknown type e9 Status
   10 0.003000   host     2.2.0        USB           36  URB_CONTROL in
   11 0.003000   2.2.0    host         USB           30  URB_CONTROL in
   12 0.003000   2.2.0    host         USB           28  Unknown type e6 Status
   13 0.003000   host     2.2.0        USB           36  URB_CONTROL in
   14 0.003000   2.2.0    host         USB           36  URB_CONTROL in
   15 0.003000   2.2.0    host         USB           28  Unknown type e7 Status
   16 0.003000   host     2.2.0        USB           36  URB_CONTROL in
   17 0.004000   2.2.0    host         USB           32  URB_CONTROL in
   18 0.004000   2.2.0    host         USB           28  Unknown type e9 Status
```

Interpreting Packets

- Reading (http://www.usbmadesimple.co.uk http://www.beyondlogic.org/usbnutshell/usb3.shtml)
- Categorized
- Mistakes (dealing with ‘facts’)
- Rules for beginners (remember sammy jankis)
- Editing replay
- Command set overview



New Sources:

- Pace Slowed
- Found Adept SDK (by digilent)
    - Digilent Only (as expected)
    - Exposed functionality
    - New Commands
    - Gave names to variables/parameters
- Wrote C program using Adept SDK
- Results (learned bit options, check theories).

- **Remaining messages:**
- Several message initialization (mostly read)
- Commands 0xE8 (seed) and 0xEC (check).
  - 0xE8 write with random
  - 0xEC read
  - 0xE8 write with 0
- Random each time
- Used IDA PRO to debug sdk program
  - Stepped into initialization function
  - C++ Mess. Class per board.
  - Address Space Layout Randomization
- Found USB code sending 0xE8
  - Parameter was based on…… uptime?!
- USB code for sending 0xEC.
  - xors of seed with ‘Digi’
- Purpose?
- Other command findings

Making a new Program:

- Called it adapt
- Converted packets into python functions
- Talking to chip vs programming
- Intro to IEEE 1532 (BSDL)
  - Motivation
  - Solution
  - Failure
- Jed and BSDL parsers
- Naive vision corrected
  - Andrew Zonenberg (PhD RPI, IOActive, Recon 2015 “From Silicon to Compiler”)
  - XC2C256 address space translation CSV
  - XC2C256 Graycode address
  - BSDL license issues
- Wrote code for flashing XC2C256
  - Issues

MORE BOARDS!!

- Checking other boards
- Plan: Talk to board → Program Chip
- Observed USB packets of other Digilent boards
  - Atlas, Digilent Nexys 2, Digilent Nexys 3
  - All had same API as first board
  - Initialized with big blob of 0xA0 messages
  - Without 0xA0, responds with name only
- All boards with 0xA0 have different USB chip
  - First board had an Atmel AVR instead.
  - Controller chip is Cypress EZ USB fx2
  - EZ USB chips and firmware.
- Did not want to deal with firmware. Let’s support more controllers...

CONTROLLERS!

- Initially only cared about programming dev boards
- Learned external controllers used more often
- Controller per vendor
- No instructions for cross vendor use. Why?
  - JTAG controllers electrically compatible! Should work
  - Drivers!
  - Make my own drivers? Documentation?
- Matching hardware is like a fashion statement

- Only Benefits Vendors
- Unacceptable, I can RE more controllers.
- Decided to start with Xilinx’s Controller

Setting up the Xilinx Platform Cable USB

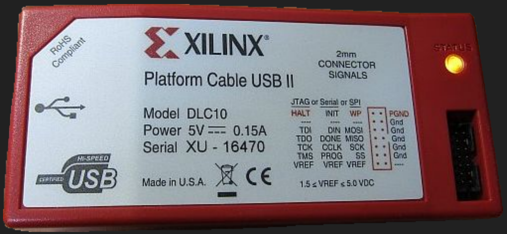
- Monitored programming on windows
- Replay problems
- Monitored powering up on windows
  - Big blob of 0xA0 messages: Firmware!
  - Took apart: **Cypress EZ-USB fx2!!!!**
- Kernel driver role in linux
- fxload and udev

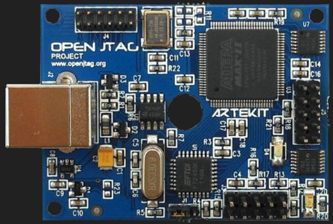
Reversing the Platform Cable’s Protocol

- Very different than Digilent’s
  - Many commands for settings
  - One JTAG command (Digilent had many)
  - Full Control of all pins all the time
  - Believed to be a 16 bit parameter for transaction count
- Found documentation from old RE effort
  - Described JTAG data format (correct)
  - Warning against %4 transitions (wrong)
- Extended documentation
  - Speed setting
  - 256 different 0x20 messages (lazy OEM)

Adding Xilinx Platform Cable software support

- Wrote controller autodetect
- Improved my API
  - Track JTAG state machine
  - Functions for state select
  - functions for direct register writing
- Existing abstractions were based on Digilent’s functions...
- Flashing XC2C256 worked with Platform Cable
- Platform Cable slower than expected (stats?)
- How Xilinx iMPACT does it
- Limited by Digilent based API
- Code too inflexible to allow fast operation
  - Need to investigate other controller APIs





MORE CONTROLLERS!

- Purchased more controllers:
  - Altera USB Blaster
    - Found OpenOCD documentation
    - Similar to Digilent API
  - OpenJTAG controller
    - Documentation provided by manufacturer
    - TOTALLY different than what I had seen
    - Keeps track of state machine for you!
    - Easy to use (no manual state tracking)
    - No fine grain control

Dealing with controller API types

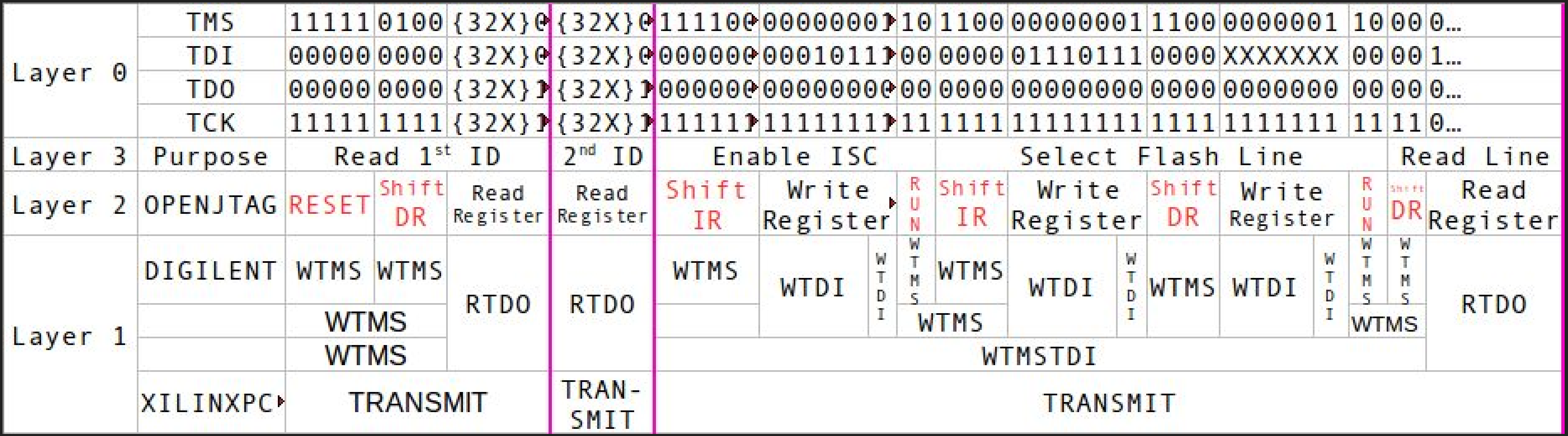
- Three known types:
  - One command controls all JTAG lines: Xilinx PCUSB
  - Many commands specifying different combinations of lines to write/hold at value: Digilent, Altera
  - State machine control (hide raw bit access): OpenJTAG
- Xilinx, Digilent, and Alterra controller API conceptually the same: grouped bit control
- Very hard to implement OpenJTAG driver in system build for bit manipulation

Pattern Emerging

- Layers of JTAG activity (high level to low level)
  - Chip Operations (Flash firmware/Erase)
  - Jtag Register Read/Write
  - Jtag State Machine Changing
  - Jtag bit manipulation

- Chip operations -> Register Read/Write is easy
- Register Read/Write -> State Machine Changes is easy
- State Machine Changes -> JTAG pin activity is easy
- **Going backwards is not easy. Similar to decompiling.**

- We Need a Compiler/Translator and an Optimizer



Layer 0	TMS	11111	0100	{32X}0	{32X}0	111100	00000001	10	1100	00000001	1100	0000001	10	00	0…
	TDI	00000	0000	{32X}0	{32X}0	000000	0001011	00	0000	01110111	0000	XXXXXXX	00	00	1…
	TDO	00000	0000	{32X}1	{32X}1	000000	0000000	00	0000	00000000	0000	0000000	00	00	0…
	TCK	11111	1111	{32X}1	{32X}1	111111	1111111	11	1111	11111111	1111	1111111	11	11	0…
Layer 3	Purpose	Read 1st ID			2nd ID	Enable ISC			Select Flash Line					Read Line	
Layer 2	OPENJTAG	RESET	Shift DR	Read Register	Read Register	Shift IR	Write Register	RUN	Shift IR	Write Register	Shift DR	Write Register	RUN	Shift DR	Read Register
Layer 1	DIGILENT	WTMS	WTMS	RTDO	RTDO	WTMS	WTDI WTDI	WTMS	WTMS	WTDI WTDI	WTMS	WTDI WTDI	WTMS	WTMS	RTDO
		WTMS						WTMS					WTMS		
		WTMS				WTMSTDI									
	XILINXPC	TRANSMIT			TRANSMIT	TRANSMIT									

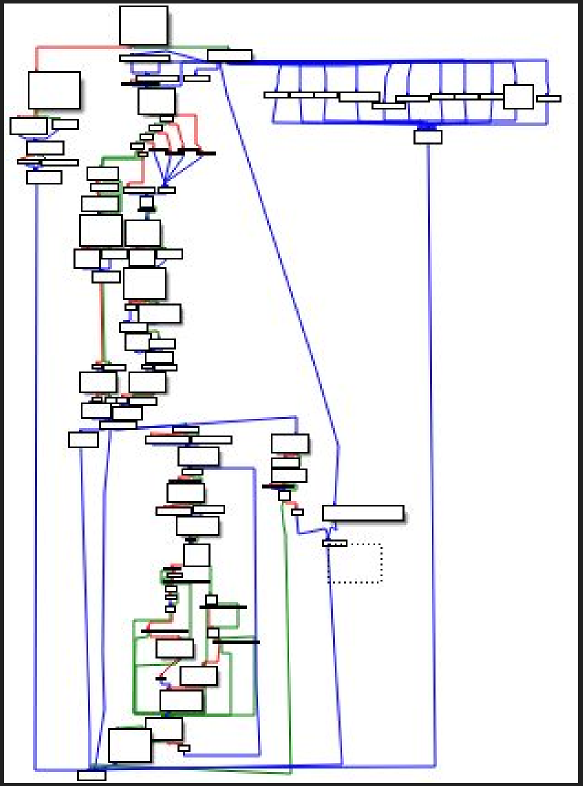
LAYER	EXAMPLES
4	Program/Erase/Validate Device
3	Execute JTAG Command (Write line of configuration data)
2	Load/Read Register, Change JTAG State
1	Any commands that reads and/or writes 1 or more values from any combination of TMS, TDI, TDO, and TCK.}
0	JTAG Electrical Activity on TMS/TDO/TCK

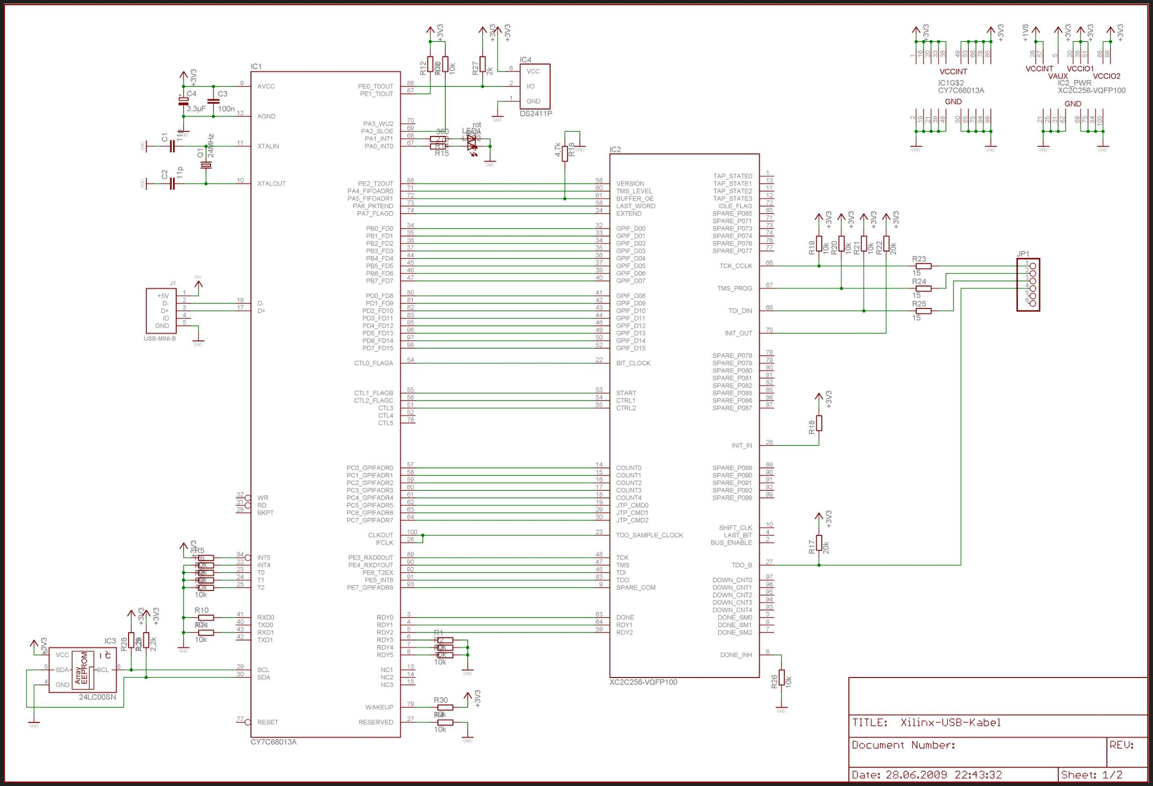
Layers and Optimizer:

- Layers and translation operation
- Requirements
- Python implementation (Lazy, results): Source https://github.com/diamondman/Adapt

Usability Issues of new tools

- Should work out of the box
- Requires target BSDL and address translation files, move information to Chip Driver
- Controllers require firmware
  - Firmware redistribution issues
  - The ‘Correct’ way of getting Xilinx firmware
    - Register Xilinx account
    - Agree to multiple EULAs
    - Download and install ‘ISE tools’ (15 GB)
    - **Copy 21.8 kb file**
- Decided all controller firmware should be open. (Inner Stallman, he would say ‘free software’)





Preparing to dissect:

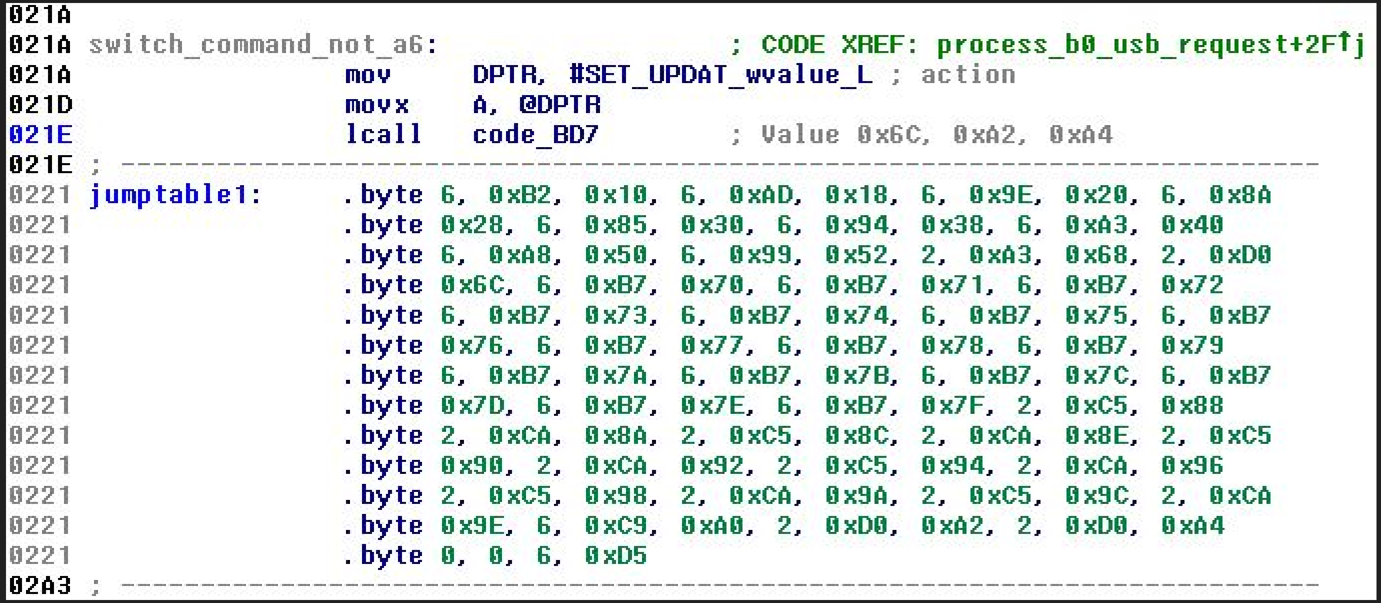
- EULA, possible workaround: Google search ‘xusb_emb.hex’.
- Found schematic at http://www.mikrocontroller.net/
- Hardware Accelerated: Coolrunner 2 (XC2C256)!!!!
- Behavior of devices (data passing)

Opening up the Xilinx Platform Cable:

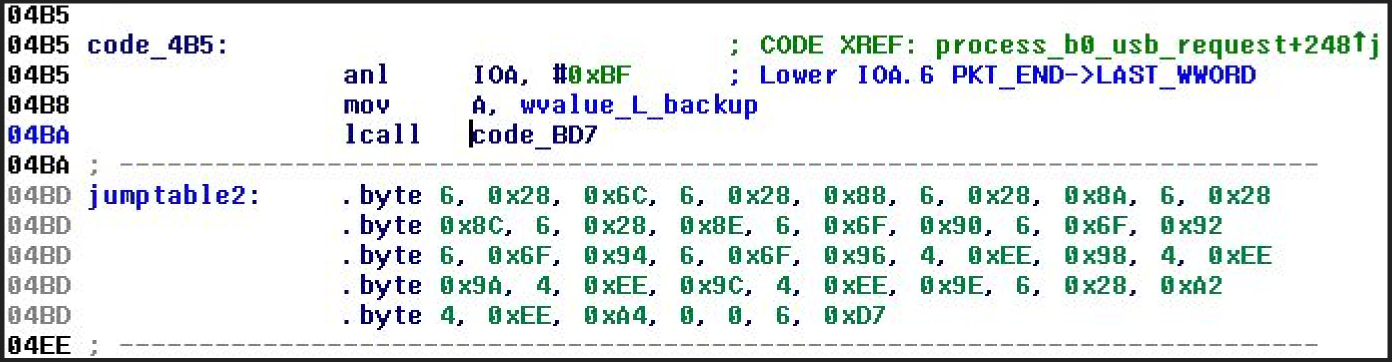
- Cypress EZ-USB fx2 architecture
  - Intel 8051 based. (Harvard Architecture)
  - 256 BYTE stack, including R0-R8
  - Attached USB hardware; interaction via shared memory ‘registers’ and interrupts.
  - Attached ‘GPIF waveform’ hardware. Interaction via shared memory ‘registers’ and interrupts.
  - Heavily extended interrupt system
  - Reading the Reference manual was NECESSARY.
- Loading into IDA PRO
  - No architecture auto-detected from hex file.
  - IDA 6.5 did not have EZ USB fx2 option.
    - Memory segment values obtained from Manual
    - Hand entered missing information from Manual.
    - ida 6.6 added better support, but not perfect
      - Still unaware of 2nd stage interrupts (leaves a lot of code as binary blobs; entry point not detected.

Inside the Firmware (IDA Pro)

- Less common architecture, no auto decompile (ASM only)
- Missing interrupts: many unknown blobs.
- Remaining blobs:
  - Many stubs with no entry points
  - Unknown blobs after function calls to ‘code_BD7’
    - Data was not valid/reasonable instructions



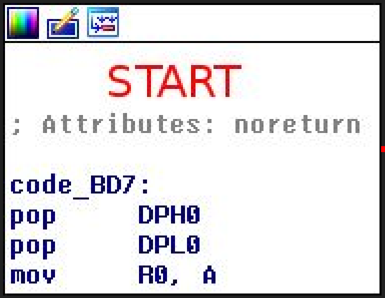
```
021A
021A switch_command_not_a6:                    ; CODE XREF: process_b0_usb_request+2F↑j
021A                 mov     DPTR, #SET_UPDAT_wvalue_L ; action
021D                 movx    A, @DPTR
021E                 lcall   code_BD7         ; Value 0x6C, 0xA2, 0xA4
021E ; ---------------------------------------------------------------------------
0221 jumptable1:     .byte 6, 0xB2, 0x10, 6, 0xAD, 0x18, 6, 0x9E, 0x20, 6, 0x8A
0221                 .byte 0x28, 6, 0x85, 0x30, 6, 0x94, 0x38, 6, 0xA3, 0x40
0221                 .byte 6, 0xA8, 0x50, 6, 0x99, 0x52, 2, 0xA3, 0x68, 2, 0xD0
0221                 .byte 0x6C, 6, 0xB7, 0x70, 6, 0xB7, 0x71, 6, 0xB7, 0x72
0221                 .byte 6, 0xB7, 0x73, 6, 0xB7, 0x74, 6, 0xB7, 0x75, 6, 0xB7
0221                 .byte 0x76, 6, 0xB7, 0x77, 6, 0xB7, 0x78, 6, 0xB7, 0x79
0221                 .byte 6, 0xB7, 0x7A, 6, 0xB7, 0x7B, 6, 0xB7, 0x7C, 6, 0xB7
0221                 .byte 0x7D, 6, 0xB7, 0x7E, 6, 0xB7, 0x7F, 2, 0xC5, 0x88
0221                 .byte 2, 0xCA, 0x8A, 2, 0xC5, 0x8C, 2, 0xCA, 0x8E, 2, 0xC5
0221                 .byte 0x90, 2, 0xCA, 0x92, 2, 0xC5, 0x94, 2, 0xCA, 0x96
0221                 .byte 2, 0xC5, 0x98, 2, 0xCA, 0x9A, 2, 0xC5, 0x9C, 2, 0xCA
0221                 .byte 0x9E, 6, 0xC9, 0xA0, 2, 0xD0, 0xA2, 2, 0xD0, 0xA4
0221                 .byte 0, 0, 6, 0xD5
02A3 ; ---------------------------------------------------------------------------
```



```
04B5
04B5 code_4B5:                                 ; CODE XREF: process_b0_usb_request+248↑j
04B5                 anl     IOA, #0xBF       ; Lower IOA.6 PKT_END->LAST_WWORD
04B8                 mov     A, wvalue_L_backup
04BA                 lcall   code_BD7
04BA ; ---------------------------------------------------------------------------
04BD jumptable2:     .byte 6, 0x28, 0x6C, 6, 0x28, 0x88, 6, 0x28, 0x8A, 6, 0x28
04BD                 .byte 0x8C, 6, 0x28, 0x8E, 6, 0x6F, 0x90, 6, 0x6F, 0x92
04BD                 .byte 6, 0x6F, 0x94, 6, 0x6F, 0x96, 4, 0xEE, 0x98, 4, 0xEE
04BD                 .byte 0x9A, 4, 0xEE, 0x9C, 4, 0xEE, 0x9E, 6, 0x28, 0xA2
04BD                 .byte 4, 0xEE, 0xA4, 0, 0, 6, 0xD7
04EE ; ---------------------------------------------------------------------------
```

More on Unknown Blobs in Firmware:

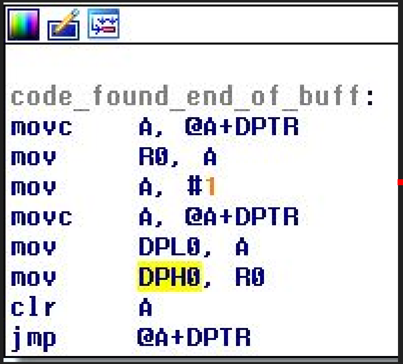
- The unknown function called before blobs was strange
  - Took one argument through a register
  - Immediately pops the return address off of Stack
  - Does not return at end, instead jumps to calculated address
  - Loops around incrementing an address
  - Calls to this function are followed by invalid code and then multiple code blocks without references.
  - Inexperienced, ended up asking IRC



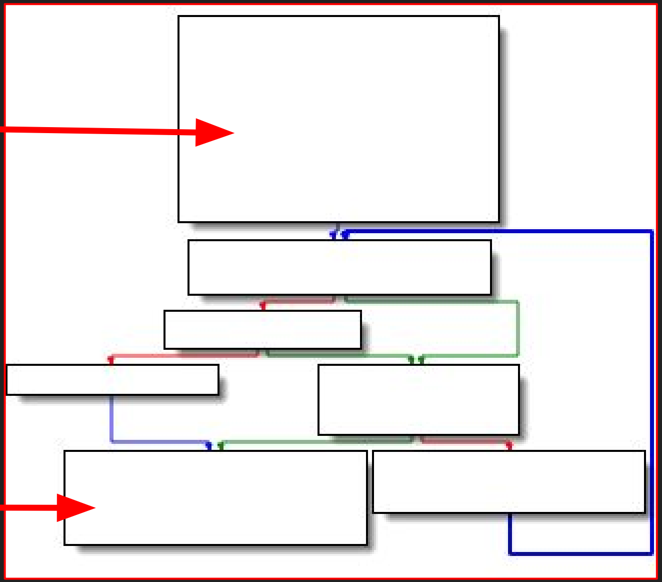
START

```
; Attributes: noreturn

code_BD7:
pop     DPH0
pop     DPL0
mov     R0, A
```

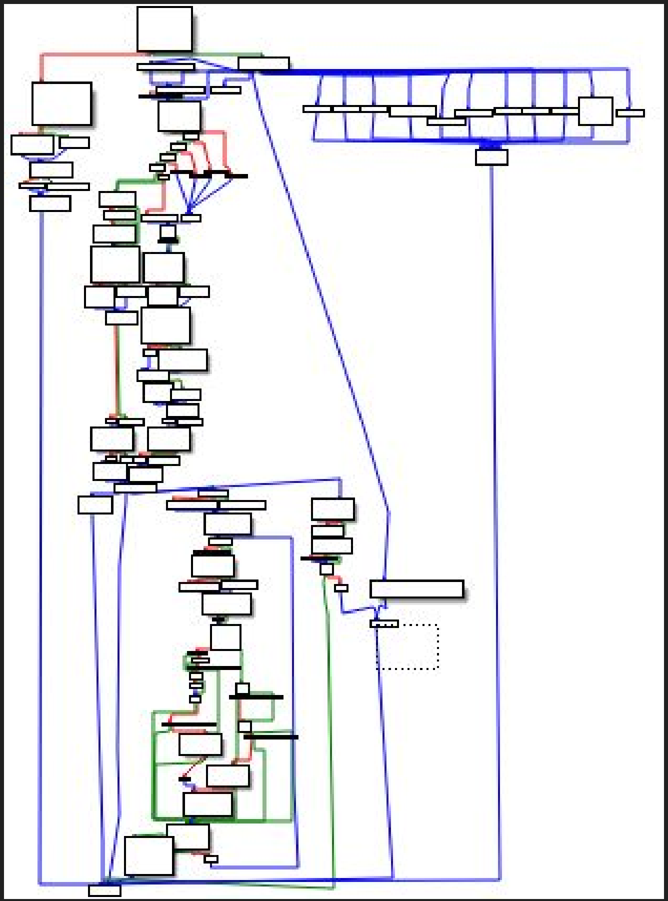


```
code_found_end_of_buff:
movc    A, @A+DPTR
mov     R0, A
mov     A, #1
movc    A, @A+DPTR
mov     DPL0, A
mov     DPH0, R0
clr     A
jmp     @A+DPTR
```



The Answer to the blobs after function calls:

- Switch statements!
- Part of code generated from the Keil Compiler
- Blobs were lists of conditions and jump locations
- Mysterious function popd return address; the address of the jump table, and returned context to calling function
- Realized what I was in for…
- Able to list of all commands



Other strange Keil compiler artifacts:

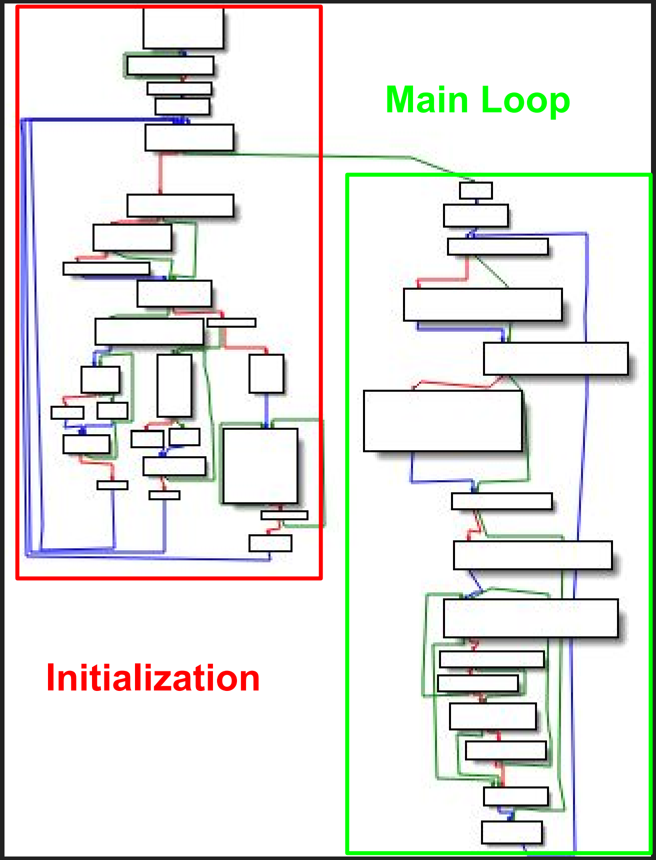
- Functions for C pointer dereference
  - Harvard architecture requires different instructions for code access and RAM access.
  - General C style pointer must specify memory type
  - 3 byte pointers, accessed with functions to read/write each data type size
- 32 bit math functions
  - 8 bit processor must do 32 bit math in software
  - Case in code where 32 bit number right shifted 8 bits instead of just reading the 2nd byte
- Expect to find things that make no sense, particularly in proprietary compilers or compilers for less common architectures.

Looking for functionality:

- Transaction count is actually 24 bit (16,777,216)
- New commands discovered
  - Command for single bit reading/writing
  - Command to initialize CPLD firmware upgrade
- Looked at how the processor controls the CPLD
  - Uses the ‘GPIF’ feature of the fx2.
    - Hardware controlled state machine for implementing electrical protocols
  - GPIF configured with data from uninitialized RAM.
- One more unknown binary blog in firmware.
  - 761 bytes!
  - About time to look into that….

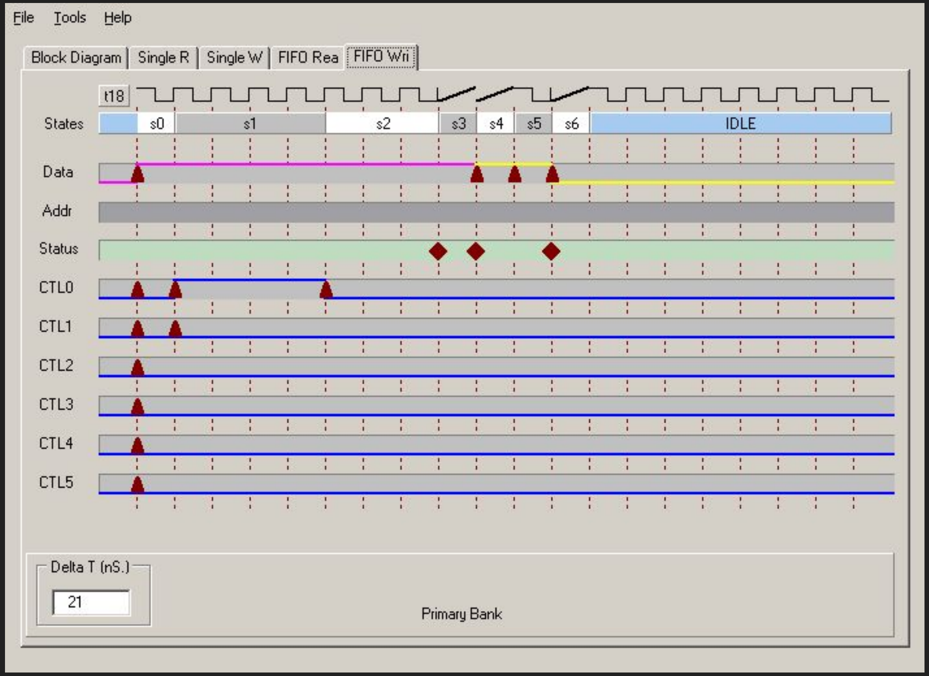
Memory Initialization Confusion:

- The final Blob of binary data is read at program start
- Code loops over addresses from the data blob
- Based on the data, writes blocks of data to addresses in a segment
- Harvard Architecture has no automatic memory initialization
- In ideal cases, code is efficient
- Translated hand optimized asm into python and fed it the blob
- All used RAM was initialized
- Contents were not modified, no reason to copy to RAM from CODE.



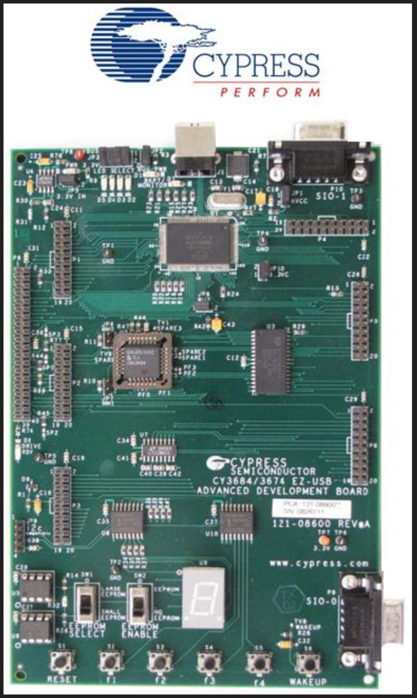
GPIF Configuration Data:

- Point of GPIF hardware...
- Extracted config data for one GPIF configuration
- Interpreting data:
  - By hand is doable but a pain
  - Cypress provides a GUI tool
    - Imports specially formatted C files
    - Produced C file from config
- Have everything to build alternate firmware



Assembling a Firmware toolchain:

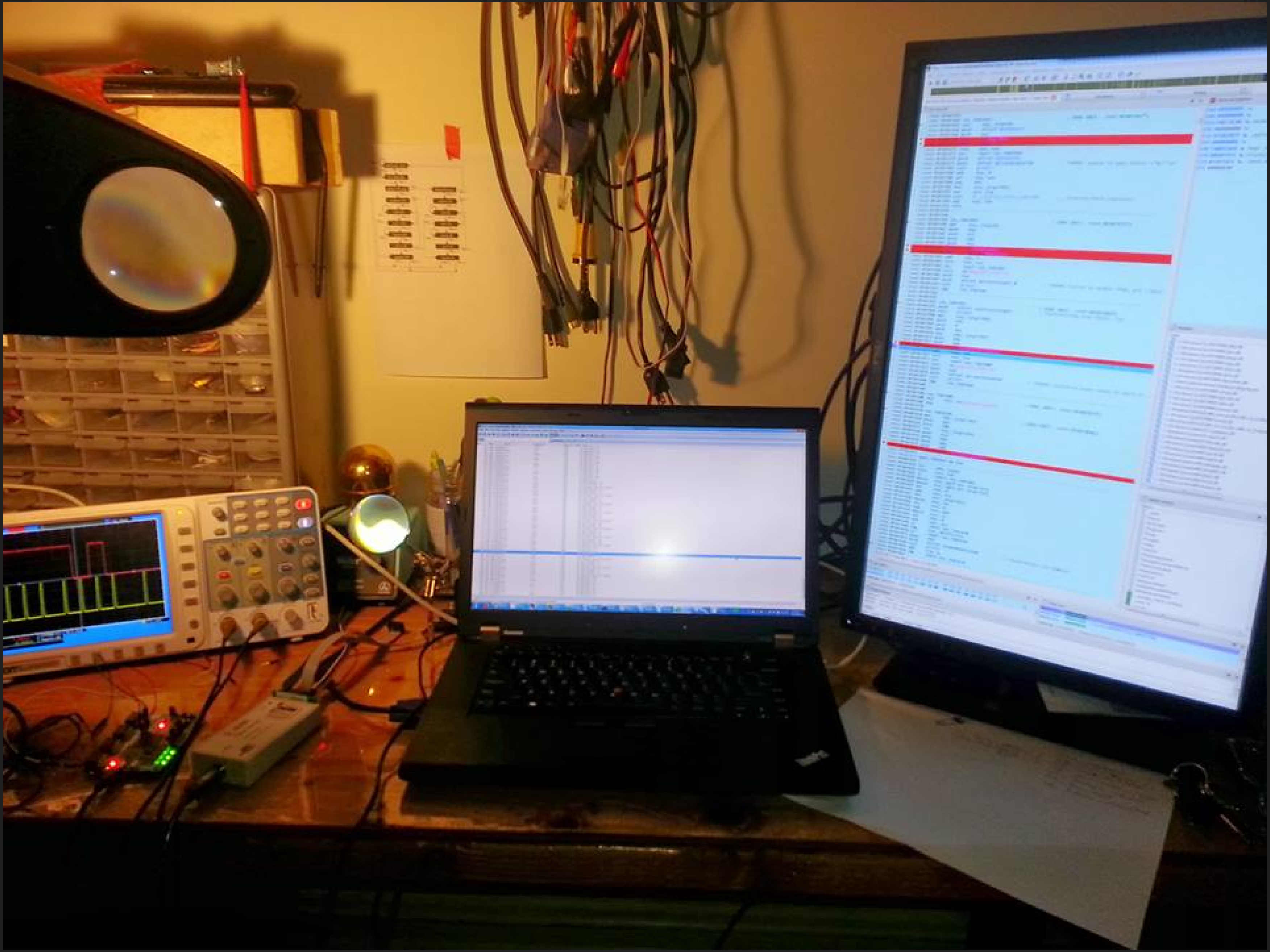
- First firmware goal
- Target: NOT the controller. (why)
- Breakout board:
  - Cypress breakout
  - 3rd party (ebay/alibaba)
- Compiler: SDCC (NOT KEIL)
- Firmware Loader: fxload
- First test
- Peripheral Library: fx2lib (djmuhlestein)
- USB Descriptor Table
- Tests (with USB)
- Debugging
- Basic commands implemented
- Lesson on reversing vs implementing



Moving to the Target Hardware:

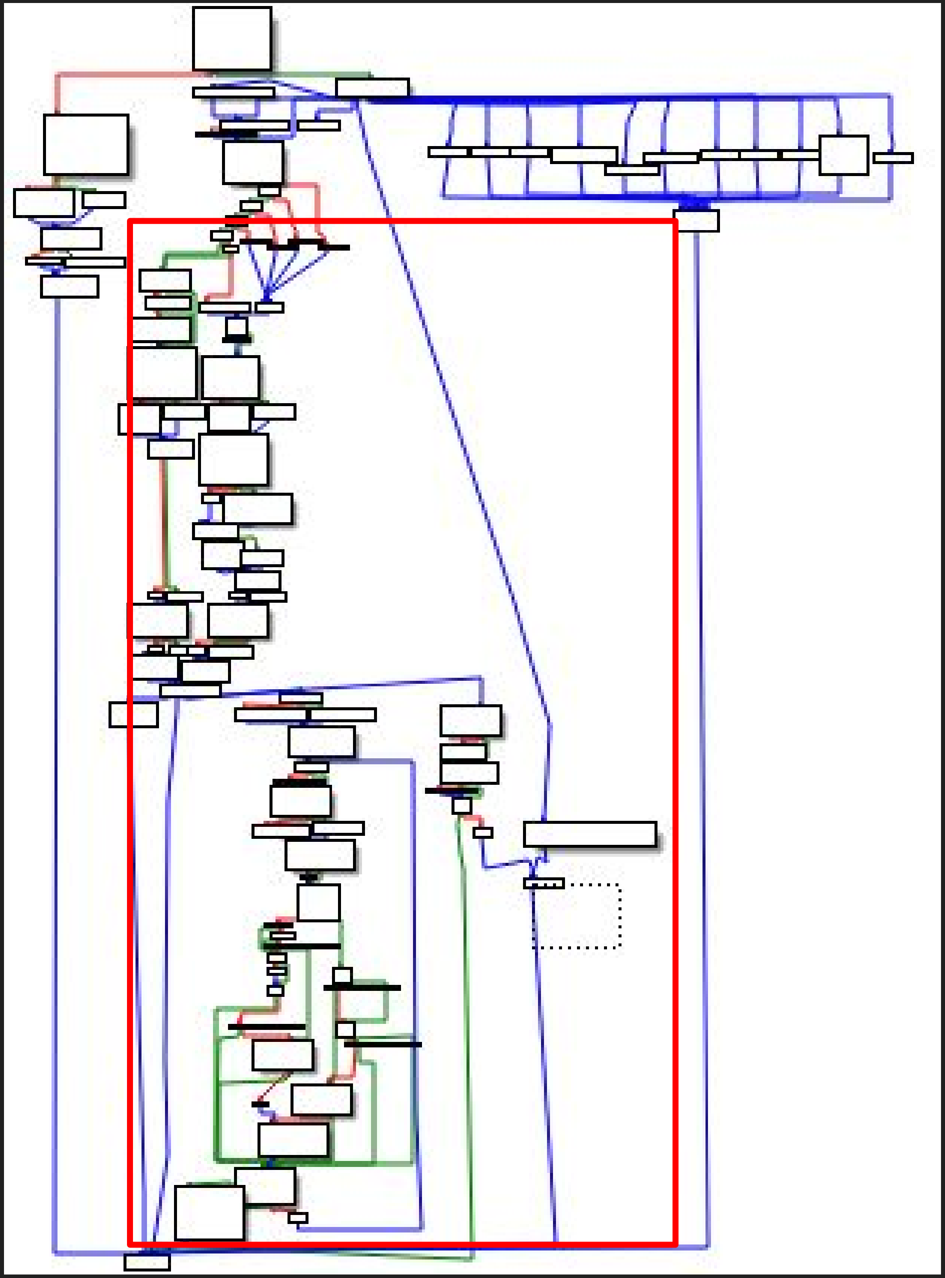
- Limitations of dev boards (no CPLD)
  - Possible solution: attach CPLD
    - Issues (CPLD firmware)
- Issues with real hardware
  - Debugging (no serial)
  - Had 2 color LED and USB messages
- Problems encountered:
  - Races
  - Infinite loops/missed conditions

It worked. Source available from https://github.com/diamondman/adapt-xpcusb-firmware



Xilinx Platform Cable Work Remaining:

- Unknown commands
  - Observations
  - Assumptions
  - SPI surprise
- Check USB high speed
- Check Power Save Mode
- Tests
- Improved Docs (story)
- Packaging



Brief Firmware Work on Digilent (Atlys board):

- Acquiring firmware:
  - No local files
  - Blobs in shared libraries
  - Issue with static analysis
- Wireshark to the rescue
  - Decoding 0xA0 messages
  - Producing hex file
- Found peripheral monitoring format (voltage)
- Firmware Template and build system

Section III
Putting this to good use.

Immediately Useful: firmware and docs
Long Term: JTAG 'layers' and translator/optimizer

Revisiting Open Tools:

- Controller Drivers and their issues
- Priority of open tools vs vendor tools
- Chip Drivers
- Technical Debt
    - Driver Interface
    - Modern tool features (OpenOCD)
- Controller Support as a chore

Moving Forward:

- Controller Support Library (Proteus ISC)
    - Shared Drivers
    - Optimized for controller protocol!
    - Common Interface?
    - Comes with open firmware
    - Programmer access to all layers
    - Lower barrier of entry (new tools)

Library Considerations and future work:

- Library or Service (Pulse?)
- Language
- Support for multiple ISC protocols
    - Early assumptions, and reality
- Needs

Special Thanks

- Danukeru (getting me to apply)

- Dr. Andrew Zonenberg (Coolrunner 2)
- David Carne (8051)
- John McMasters (community)
- All of silicon pr0n

- Mek Karpeles (...mess)
- Caitlin Morgan (listening to every version)
- Friends (input on proposal)

QUESTIONS?