Preventing "Oh Shit" Moments for €20 or Less

Philip A. Polstra, Sr.

University of Dubuque

black hat EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

WWW.BLACKHAT.COM

Roadmap

- Why this talk?
- Brief history of USB
- How does USB work?
- It’s all descriptors and endpoints
- Bulk-only Mass Storage Devices
- Keeping your toys intact
- Microcontrollers are fun (and cheap)
- How can I have more fun with USB?

Why this talk?

- USB flash drives have become defacto standard for storing and exchanging info
- Everyone uses them, but few understand them
- Having your Katana drive deleted by antivirus sucks
- A cheap way of write-blocking your thumb drives doesn't suck

Brief History or USB

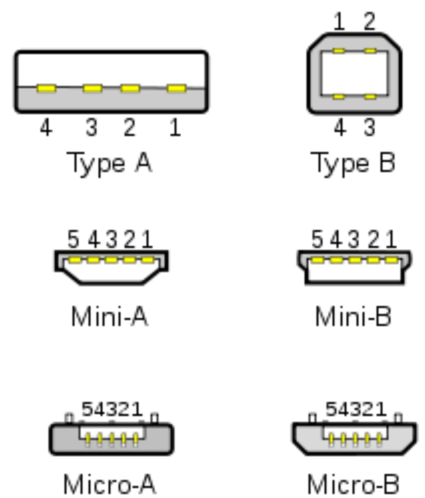
- Non-universal serial, PS/2 ports, & LPT
- 1996 USB 1.0 (1.5 or 12 Mbps)
- 1998 USB 1.1
- 2000 USB 2.0 (1.5, 12, or 480 Mbps)
- Long pause
- 2008 USB 3.0 (up to 5 Gbps)

HOW DOES USB WORK?

Hardware

- Simple 4-wire connection (power, ground, 2 data wires)
- Cabling prevents improper connections
- Hot pluggable
- Differential voltages provide greater immunity to noise
- Cable lengths up to 16 feet are possible

Pin	Name	Cable color	Description
1	VBUS	Red	+5 V
2	D−	White	Data −
3	D+	Green	Data +
4	GND	Black	Ground



Software

- Automatic configuration
- No settable jumpers
- Enumeration
- Standard device classes with corresponding drivers
  - HID
  - Printer
  - Audio
  - **Mass Storage**

Connecting a Device

- Device is connected
- Hub detects
- Host (PC) is informed of new device
- Hub determines device speed capability as indicated by location of pull-up resistors
- Hub resets the device
- Host determines if device is capable of high speed (using chirps)
- Hub establishes a signal path
- Host requests descriptor from device to determine max packet size
- Host assigns an address
- Host learns devices capabilities
- Host assigns and loads an appropriate device driver (INF file)
- Device driver selects a configuration

IT'S ALL DESCRIPTORS AND ENDPOINTS

Endpoints

- The virtual wire for USB communications
- All endpoints are one way (direction relative to host)
- Packet fragmentation, handshaking, etc. done by hardware (usually)
- High bit of address tells direction 1=in 0=out
- Types of endpoints
  - Control
  - Bulk transport
  - Interrupt
  - Isochronous

Control Endpoints

- Primary mechanism for most devices to communicate with host
- Every device must have at least one in and out control endpoint EP0
- Device must respond to standard requests
  - Get/set address, descriptors, power, and status
- Device may respond to class specific requests
- Device may respond to vendor specific requests

Control Endpoints (continued)

- May have up to 3 transport stages: Setup, Data, Status
- Setup stage
  - Host sends Setup token then data packet containing setup request
  - If device receives a valid setup packet, an ACK is returned
  - Setup request is 8 bytes
    - 1st byte is bitmap telling type of request & recipient (device, interface, endpoint)
    - Remaining bytes are parameters for request and response
- Data stage (optional) – requested info transmitted
- Status stage – zero length data packet sent as ACK on success

Interrupt & Isochronous Endpoints

- Interrupt endpoints
  - Used to avoid polling and busy waits
  - Keyboards are a good example
  - Usually low speed (allows for longer cables, etc.)
- Isochronous endpoints
  - Guaranteed bandwidth
  - Used primarily for time-critical apps such as streaming media

Bulk Endpoints

- No latency guarantees
- Good performance on an idle bus
- Superseded by all other transport types
- Full (8-64 byte packets) & high speed (512 byte packets) only
- Used extensively in USB flash drives (and external hard drives)
- Transactions consist of a token packet, 0 or more data packets, and an ACK handshake packet (if successful)

Descriptors

- They describe things (duh!)
- Have a standard format
  - 1st byte is the length in bytes (so you known when you're done)
  - 2nd byte determines type of descriptor
  - Remaining bytes are the descriptor itself
- Common types
  - Device: tells you basic info about the device
  - Configuration: how much power needed, number of interfaces, etc.
  - Interface: How do I talk to the device
  - Endpoint: Direction, type, number, etc.
  - String: Describe something in unicode text

Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	18 bytes
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	0x200
4	bDeviceClass	1	Class	Class Code
5	bDeviceSubClass	1	SubClass	Subclass Code
6	bDeviceProtocol	1	Protocol	Protocol Code
7	bMaxPacketSize	1	Number	Maxi Packet Size EP0
8	idVendor	2	ID	Vendor ID
10	idProduct	2	ID	Product ID
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manu Descriptor
15	iProduct	1	Index	Index of Prod Descriptor
16	iSerialNumber	1	Index	Index of SN Descriptor
17	bNumConfigurations	1	Integer	Num Configurations

Configuration Descriptor (header)

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size in Bytes
1	bDescriptorType	1	Constant	0x02
2	wTotalLength	2	Number	Total data returned
4	bNumInterfaces	1	Number	Num Interfaces
5	bConfigurationValue	1	Number	Con number
6	iConfiguration	1	Index	String Descriptor
7	bmAttributes	1	Bitmap	b7 Reserved, set to 1. b6 Self Powered b5 Remote Wakeup b4..0 Reserved 0.
8	bMaxPower	1	mA	Max Power in mA/2

Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	9 Bytes
1	bDescriptorType	1	Constant	0x04
2	bInterfaceNumber	1	Number	Number of Interface
3	bAlternateSetting	1	Number	Alternative setting
4	bNumEndpoints	1	Number	Number of Endpoints used
5	bInterfaceClass	1	Class	Class Code
6	bInterfaceSubClass	1	SubClass	Subclass Code
7	bInterfaceProtocol	1	Protocol	Protocol Code
8	iInterface	1	Index	Index of String Descriptor

Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor (7 bytes)
1	bDescriptorType	1	Constant	Endpoint Descriptor (0x05)
2	bEndpointAddress	1	Endpoint	b0..3 Endpoint Number. b4..6 Reserved. Set to Zero b7 Direction 0 = Out, 1 = In
3	bmAttributes	1	Bitmap	b0..1 Transfer Type 10 = Bulk b2..7 are reserved. I
4	wMaxPacketSize	2	Number	Maximum Packet Size
6	bInterval	1	Number	Interval for polling endpoint data

String Descriptors

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of Descriptor in Bytes
1	bDescriptorType	1	Constant	String Descriptor (0x03)
2	bString	n	Unicode	Unicode Encoded String

Note: String 0 is a special case that lists available languages.
Most common is 0x0409 – U.S. English

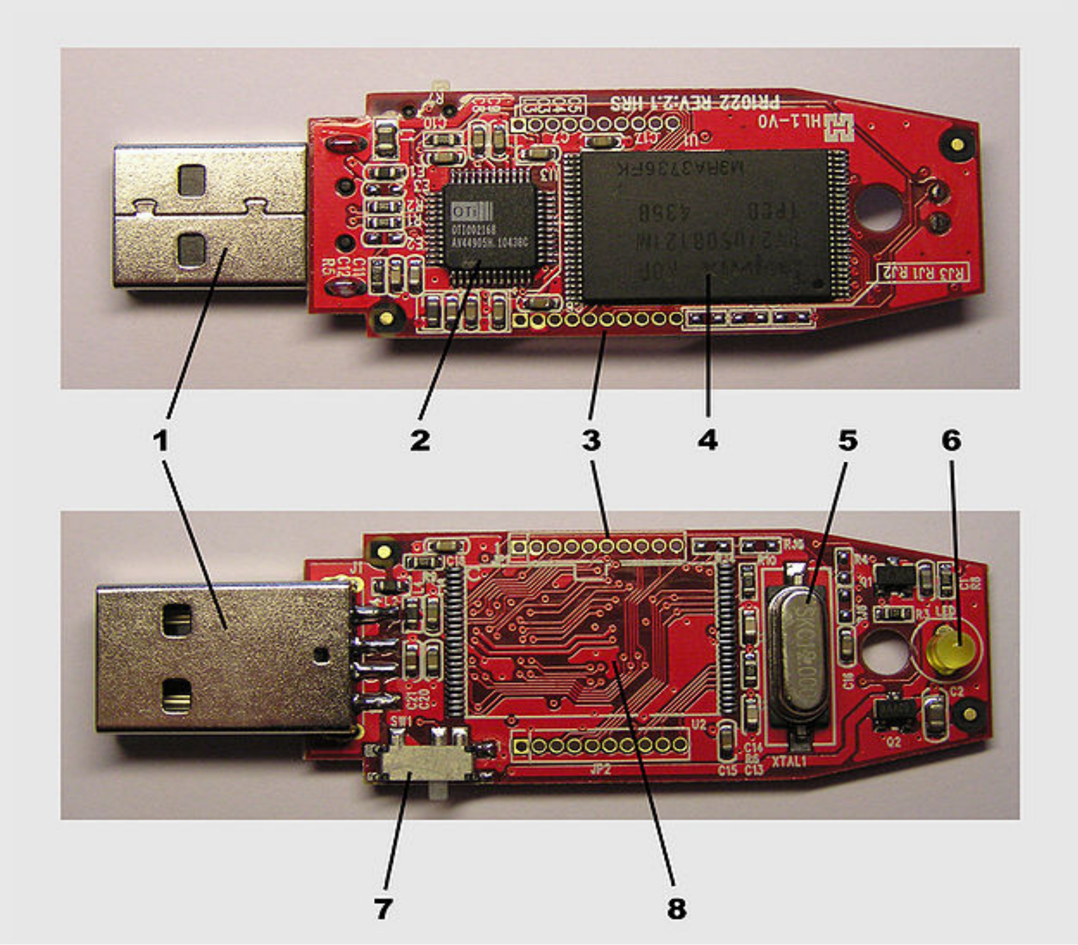
Now that we have learned a little about general devices, without further delay…

BULK-ONLY MASS STORAGE DEVICES

USB Flash Drives

- Hardware
- Software
- Filesystems
- Talk to a flash drive

Hardware



Hardware (continued)

- Typically utilize NAND flash memory
- Memory degrades after 10,000 write cycles
- Most chips not even close to high-speed USB speed (480 Mbps)
- Can only be written in blocks (usually 512, 2048, or 4096 bytes)
- Chips are somewhat easily removed from damaged drives for forensic recovery
- Some controllers have JTAG capability which can be used for memory access
- Some controller chips steal some flash memory for themselves

Hardware (continued)

- Nearly all flash drives present themselves as SCSI hard drives
- "Hard drive" sectors are typically 512, 2048, or 4096 bytes
- SCSI transparent command set is used
- Most drives are formatted as one partition or logical unit
  - Additional logical units can hide info from Windows machines
- Reported size may not match actual media size
  - Info can be hidden in higher sectors
  - Some cheap drives are out there that grossly over report size
  - A typical 512 byte sector needs 16 bytes for error correction

Software

- Usually implemented in firmware within specialized controller chips
- Must:
  - Detect communication directed at drive
  - Respond to standard requests
  - Check for errors
  - Manage power
  - Exchange data

Filesystems

- Most preformatted with FAT or FAT32
- NTFS
- TrueFFS
- ExtremeFFS
- JFFS
- YAFFS
- Various UNIX/Linux file systems

Talking to a Flash Drive

- Bulk-Only Mass Storage (aka BBB) protocol used
  - All communications use bulk endpoints
  - Three phases: CBW, data-transport (optional), CSW
  - Commands sent to drive using a Command Block Wrapper (CBW)
  - CBW contains Command Block (CB) with actual command
  - Nearly all drives use a (reduced) SCSI command set
  - Commands requiring data transport will send/receive on bulk endpoints
  - All transactions are terminated by a Command Status Wrapper (CSW)

Command Block Wrapper

```
typedef struct _USB_MSI_CBW {
    unsigned long dCBWSignature; //0x43425355 "USBC"
    unsigned long dCBWTag; // associates CBW with CSW response
    unsigned long dCBWDataTransferLength; // bytes to send or receive
    unsigned char bCBWFlags; // bit 7 0=OUT, 1=IN all others zero
    unsigned char bCBWLUN; // logical unit number (usually zero)
    unsigned char bCBWCBLength; // 3 hi bits zero, rest bytes in CB
    unsigned char bCBWCB[16]; // the actual command block (>= 6 bytes)
} USB_MSI_CBW;
```

Command Block

- 6-16 bytes depending on command
- Command is first byte
- Format Unit Example:

```
typedef struct _CB_FORMAT_UNIT {
    unsigned char OperationCode; //must be 0x04
    unsigned char LUN:3; // logical unit number (usually zero)
    unsigned char FmtData:1; // if 1, extra parameters follow command
    unsigned char CmpLst:1; // if 0, partial list of defects, 1, complete
    unsigned char DefectListFormat:3; //000 = 32-bit LBAs
    unsigned char VendorSpecific; //vendor specific code
    unsigned short Interleave; //0x0000 = use vendor default
    unsigned char Control;
} CB_FORMAT_UNIT;
```

Command Block (continued)

- Read (10) Example:

```
typedef struct _CB_READ10 {
    unsigned char OperationCode; //must be 0x28
    unsigned char RelativeAddress:1; // normally 0
    unsigned char Resv:2;
    unsigned char FUA:1; // 1=force unit access, don't use cache
    unsigned char DPO:1; // 1=disable page out
    unsigned char LUN:3; //logical unit number
    unsigned long LBA; //logical block address (sector number)
    unsigned char Reserved;
    unsigned short TransferLength;
    unsigned char Control;
} CB_READ10;
```

Command Block (continued)

- Some Common SCSI Commands:

FORMAT_UNIT=0x4, //required

INQUIRY=0x12, //required

MODE_SELECT6=0x15,

MODE_SELECT10=0x55,

MODE_SENSE6=0x1A,

MODE_SENSE10=0x5A,

READ6=0x08, //required

READ10=0x28, //required

READ12=0xA8,

READ_CAPACITY10=0x25, //required

READ_FORMAT_CAPACITIES=0x23,

REPORT_LUNS=0xA0, //required

REQUEST_SENSE=0x03, //required

SEND_DIAGNOSTIC=0x1D, //required

START_STOP_UNIT=0x1B,

SYNCHRONIZE_CACHE10=0x35,

TEST_UNIT_READ=0x00, //required

VERIFY10=0x2F,

WRITE6=0x0A, //required

WRITE10=0x2A,

WRITE12=0xAA

Command Status Wrapper

- Read Sense command can be used for details on failed operations

```
typedef struct _USB_MSI_CSW {
    unsigned long dCSWSignature; //0x53425355 “USBS”
    unsigned long dCSWTag; // associate CBW with CSW response
    unsigned long dCSWDataResidue; // difference between requested data and actual
    unsigned char bCSWStatus; //00=pass, 01=fail, 02=phase error, reset
} USB_MSI_CSW;
```

Now that we know how bulk-only mass storage devices work…

HOW DO I KEEP MY TOYS INTACT?

Blocking Write Operations

- Free ways
  - Some flash drives have write-protect switches (somewhat rare)
  - HKEY_LOCAL_MACHINE\SYSTEM \CurrentControlSet\Control\ StorageDevicePolicies\ WriteProtect
    - Blocks writing to ALL USB devices
- Non-free ways
  - Commercial write-blockers (seem to be pricey)
  - Microcontroller-based device (discussed next)

Enough background. Let the fun begin…

MICROCONTROLLERS ARE FUN (AND CHEAP)

Fun with Microcontrollers

- Chip Choice
- A Microcontroller-Based Write Blocker

Chip Choice Options

- AVR (as found in Arduino family)
  - Cheap
  - Well understood
  - Loads of code out there
  - Too underpowered to do USB without external components (<20MHz)
- PIC family
  - Relatively cheap
  - Programming somewhat more involved than AVR
  - Newer chips SMD only, not easy DIP package
  - Some USB device code, but not host code out there

Chip Choice Winner

- None of the above
- FTDI Vinculum II
  - Relatively new chip
  - A little faster than AVRs (48 MHz)
  - Real-time multi-threaded OS
  - Libraries for several standard USB classes
    - BOMS is one – but we can't use it for this project, unfortunately
  - Unlike AVR, different pin packages differ only with GPIO lines available
    - Same flash memory
    - Same RAM

Chip Choice

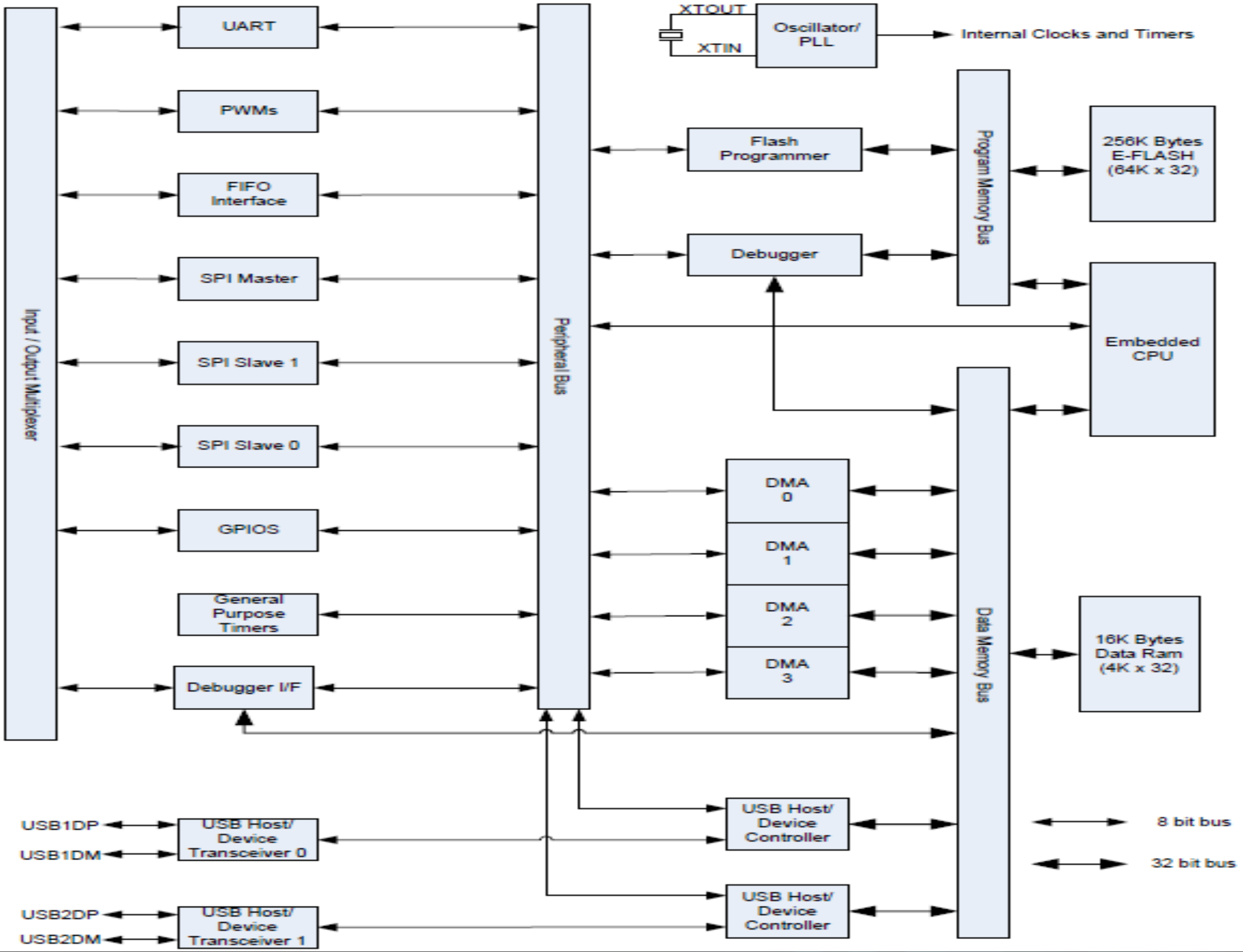
- FTDI Vinculum II dual USB host/slave controller
  - 2 full-speed USB 2.0 interfaces (host or slave capable)
  - 256 KB E-flash memory
  - 16 KB RAM
  - 2 SPI slave and 1 SPI master interfaces
  - Easy-to-use IDE
  - Simultaneous multiple file access on BOMS devices
- Several development modules available
  - Convenient for prototyping (only SMD chips available)
  - Cheap enough to embed in final device
  - One format is Arduino clone (Vinco)

Chip Choice (continued)

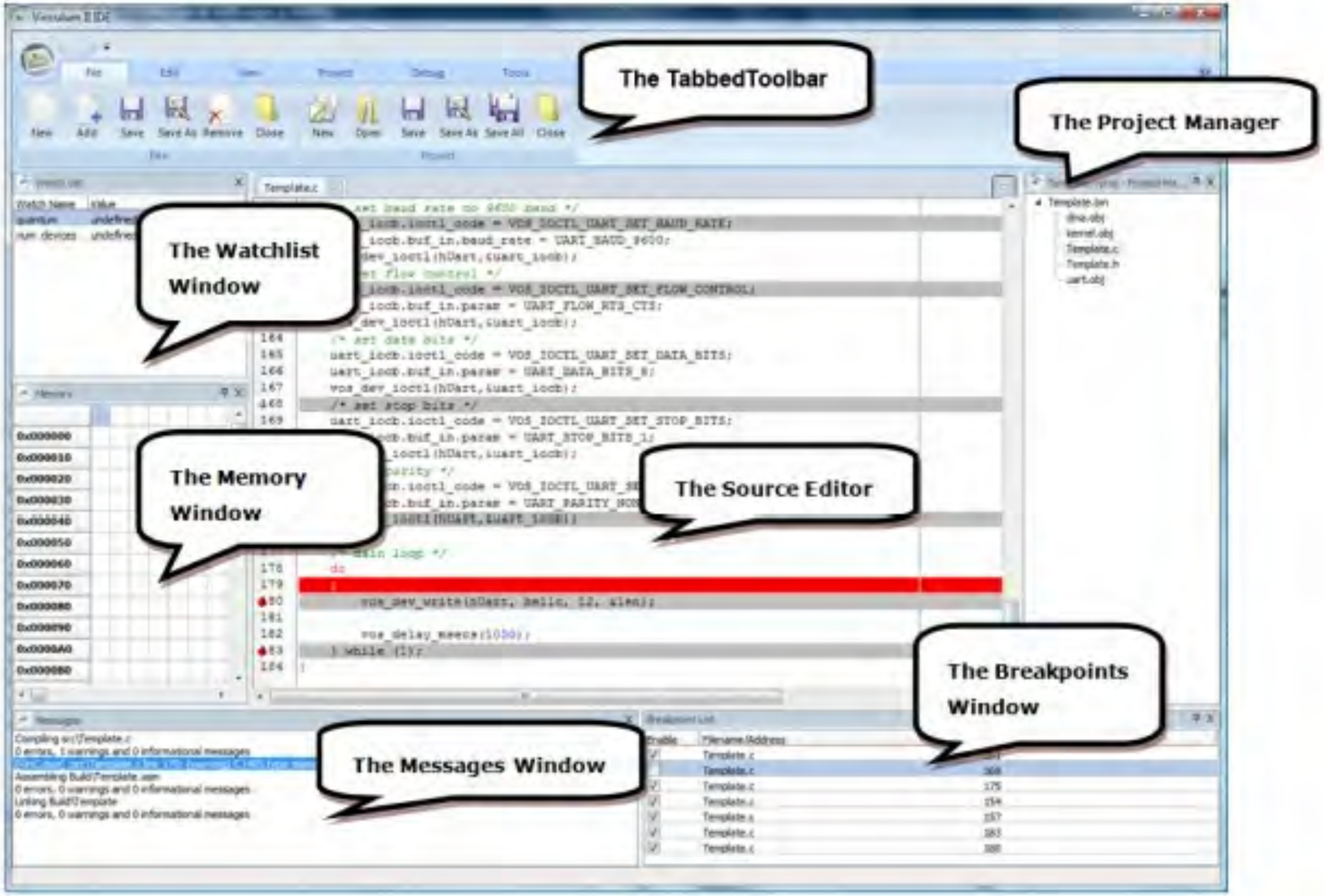
FTDI Chip

VINCULUM
BINDING USB TECHNOLOGIES

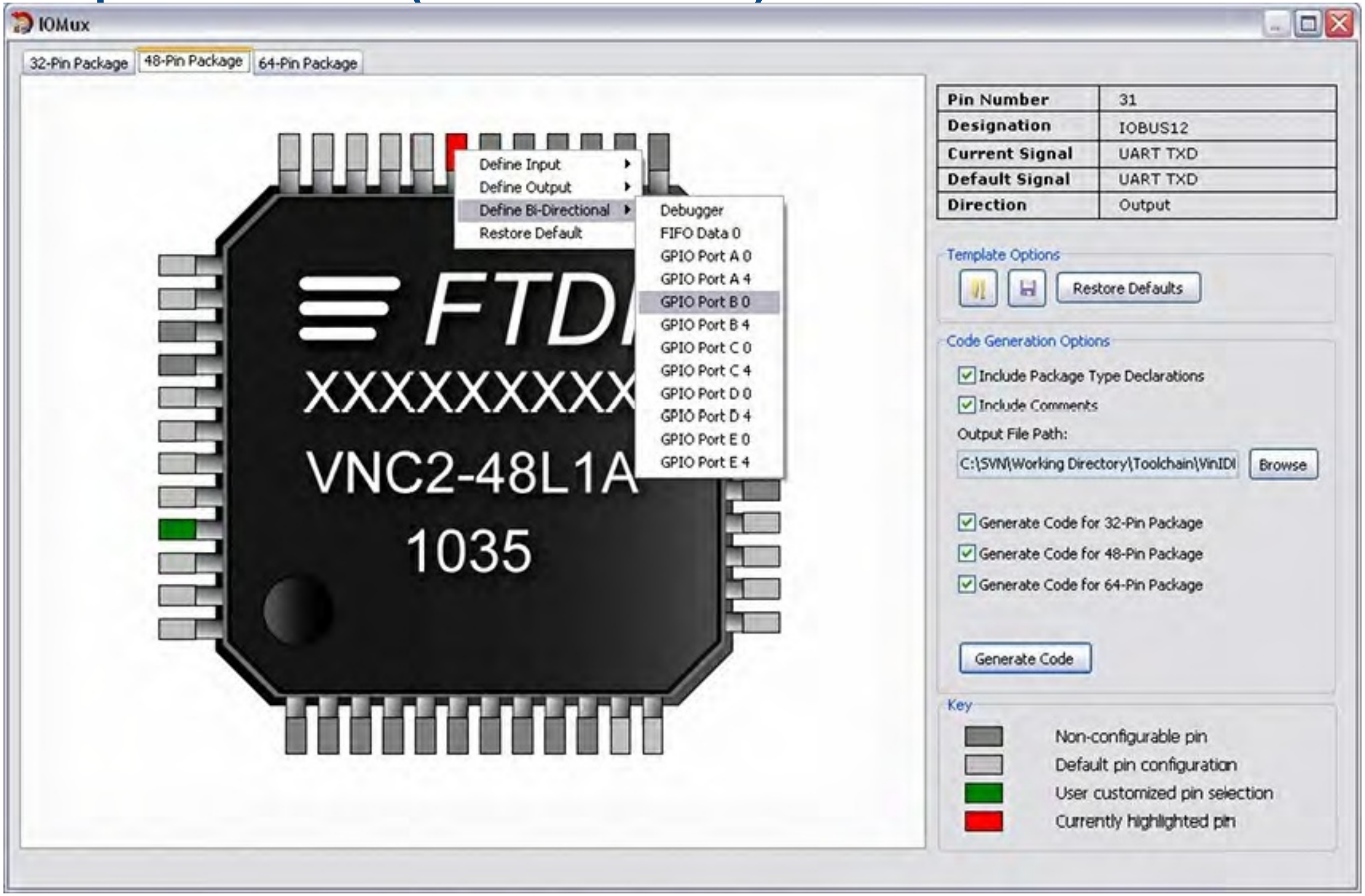
Chip Choice (continued)



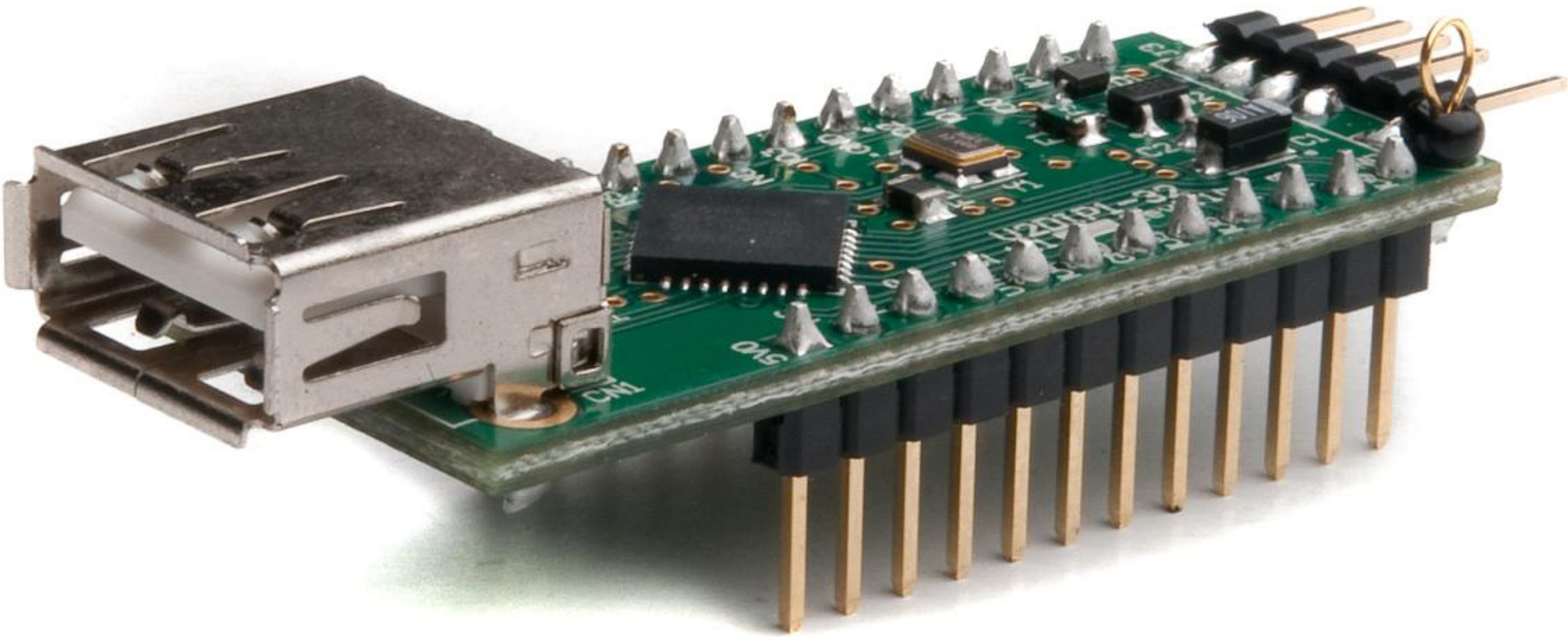
Chip Choice (continued)



Chip Choice (continued)



Package A - Small & only 4 Pins to Solder



Package B – Slightly Larger-No Soldering

Microcontroller-Based Write Blocker

- Need to block bad command blocks that could modify our drive
- If we are lazy we just block the “bad” commands
- Best practice and future proofing would dictate white listing instead
- All VNC2 chips have the same memory and flash
  - Only difference is number of GPIO lines available
  - Same software will almost run on both packages
    - Vinco board requires toggling a GPIO line to provide power to host port

Write Blocker High-Level Design

- One thread associated with slave port to appear as a BOMS device
  - One thread watches control endpoint and services requests from host
- One thread associated with the host port for talking to flash drive
  - Thread enumerates the device and gets endpoints. Then periodically checks to see if the drive is still there
- Main thread bridges slave and host
  - Non-CBW packets (data packets) are passed through to host port
  - Whitelisted CBWs are also passed on to host port
- USB Host & Slave drivers built in to VOS create additional threads
  - Trying to do this yourself-> more complex & no improvement

Allowed Commands

FORMAT_UNIT=0x4, //required

INQUIRY=0x12, //required

MODE_SELECT6=0x15,

MODE_SELECT10=0x55,

MODE_SENSE6=0x1A,

MODE_SENSE10=0x5A,

READ6=0x08, //required

READ10=0x28, //required

READ12=0xA8,

READ_CAPACITY10=0x25, //required

READ_FORMAT_CAPACITIES=0x23,

REPORT_LUNS=0xA0, //required

REQUEST_SENSE=0x03, //required

SEND_DIAGNOSTIC=0x1D, //required

START_STOP_UNIT=0x1B,

SYNCHRONIZE_CACHE10=0x35,

TEST_UNIT_READ=0x00, //required

VERIFY10=0x2F,

WRITE6=0x0A, //required

WRITE10=0x2A,

WRITE12=0xAA

The Main Thread

- Waits for CBW packets to arrive on Bulk Out endpoint
- Calls appropriate handler function based on command
  - Whitelisted commands:
    - Forward CBW to drive
    - Perform Data phase (if any) with drive and forward to PC
    - Received CSW from device and forward to PC
  - Non-whitelisted commands:
    - ACK CBW
    - Fake Data phase (if any)
    - Return CSW to PC
      - Some commands return success because Windows is unhappy with failures

Main Loop

```
usbSlaveBoms_readCbw(cbw, slaveBomsCtx);
switch (cbw->cb.formated.command)
{
		case BOMS_INQUIRY:
		handle_inquiry(cbw);
		break;
		…
}
```

Example Handler

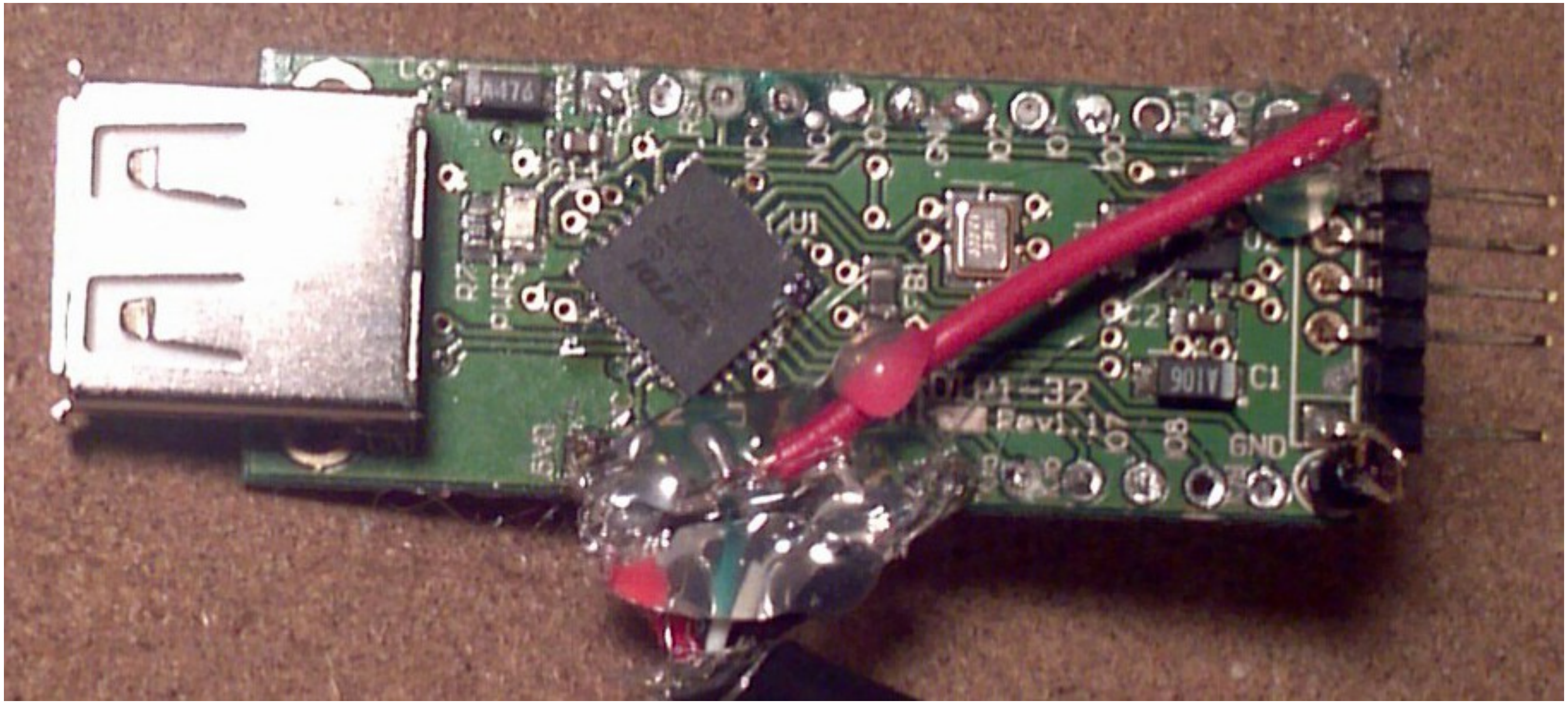
```
void handle_inquiry(boms_cbw_t *cbw)
{
        unsigned char buffer[64];
        unsigned short responseSize;
        boms_csw_t csw;
        if (forward_cbw_to_device(cbw))
        {
                if (responseSize = receive_data_from_device(&buffer[0], 36))
                {
                        forward_data_to_slave(&buffer[0], responseSize);
                        if (receive_csw_from_device(&csw))
                        {
                                forward_csw_to_slave(&csw);
                        }
                }
        }
}
```

Complications

- Windows & Linux treat drives differently
  - Windows will try to look for and autoplay media
  - Windows doesn't appear to see other than first LUN
  - Early prototype experience
    - Worked fine under Linux
    - Caused BSoD on Windows (exploit?)
  - Linux seems to pull in a lot of data up front
  - Windows misbehaves if you correctly fail some commands such as Write

Recommend Usage

- Block writes on Windows
  - Allows you avoid risking damaging your flash drives loaded with tools
- For forensics examination Linux is recommended
  - Windows might miss or mishandle upper LUNs
  - Linux has all the non-FAT filesystems you might encounter
  - You were probably running Linux already
  - Remember that even plugging in a drive will change timestamps!
    - This can hamper your investigation
    - This can contaminate evidence if you end up in court



And now what you really wanted to see…

IT'S DEMO TIME!

References

- USB Complete: The Developers Guide (4th ed.) by Jan Axelson
- USB Mass Storage: Designing and Programming Devices and Embedded Hosts by Jan Axelson
- http://www.usb.org
- http://www.ftdichip.com for more on VNC2
- http://seagate.com for SCSI references
- Embedded USB Design by Example by John Hyde
- My 44Con USB Flash Drive Forensics Video http://www.youtube.com/watch?v=CIVGzG0W-DM

- **All schematics and source code are available on request via e-mail to ppolstra@dbq.edu**

Lessons Learned

- Sometimes straightforward ugly design is best
  - Adding more threads didn't work out
    - Complexity increased
    - No real performance difference
    - Introduced possible timing issues & crashes
  - Quasi object-oriented design didn't work out either
    - Differences in commands made generalized handler tricky
    - Micocontrollers are not the same as desktop CPUs
      - Not a lot of RAM so have to watch function calls and stack usage
      - Thread model is not as sophisticated
- Open source makes the world better
  - If FTDI libraries were open source, this project would have been easier

Questions?

black hat® EUROPE

March 14-16, 2012
NH Grand Krasnapolsky Hotel
Amsterdam, Netherlands

PLEASE REMEMBER TO PROVIDE FEEDBACK ON THIS TALK