USB Flash Drive Forensics

Philip A. Polstra, Sr.

University of Dubuque

USB Basics

- History
- Hardware
- Software

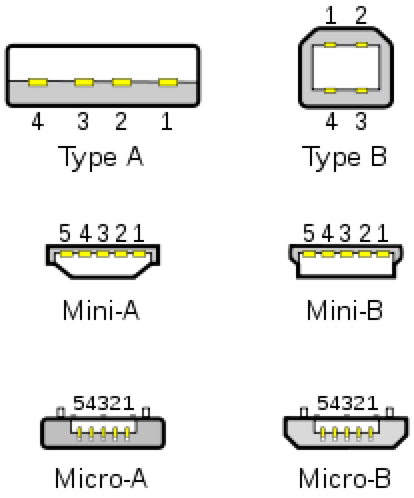
History

- 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)

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



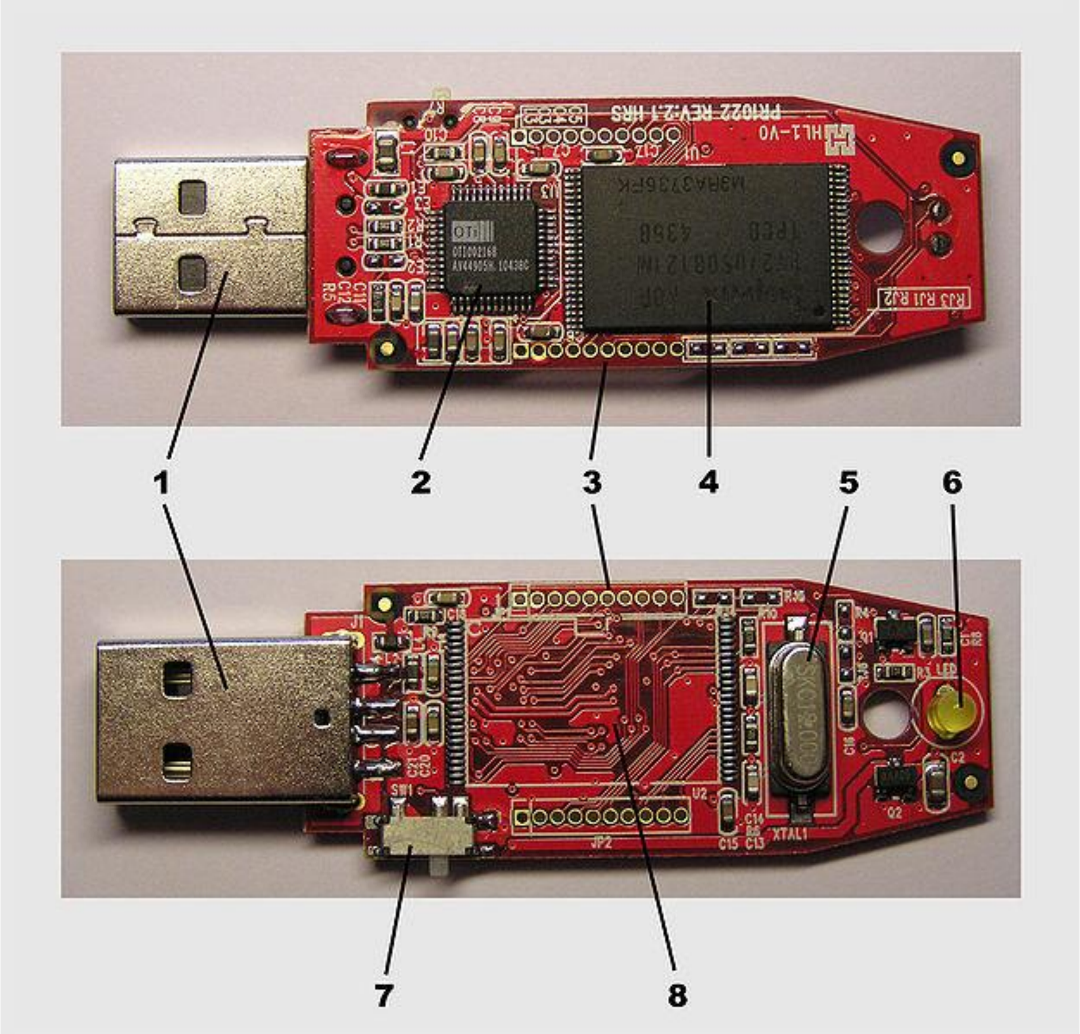
Software

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

USB Flash Drives

- Hardware
- Software
- Filesystems
- Windows

Hardware



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

USB Flash Drives and Windows

- Connecting a Drive
- Blocking write operations
- Who was here?

Connecting a Drive

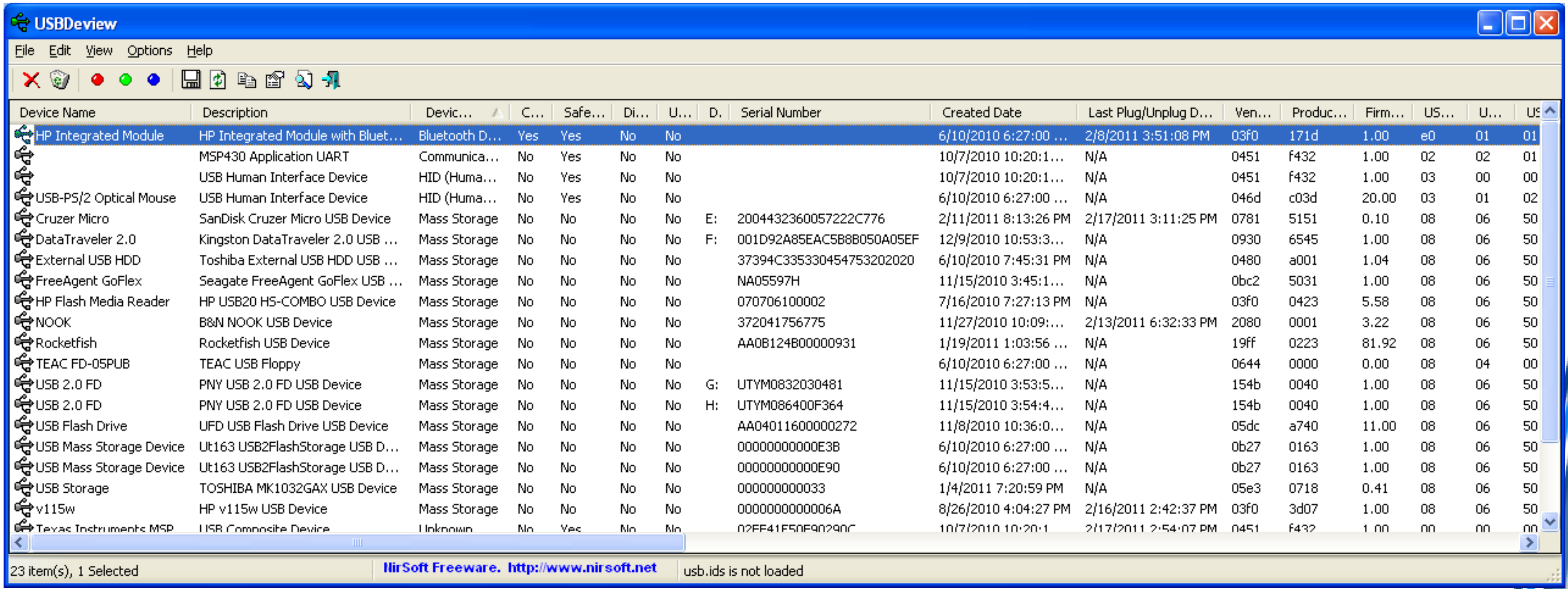
- Device is connected
- Hub detects
- Host (PC) is informed of new device
- Hub determines device speed capability as indicated by location of pull-up resistors
- Hub resets the device
- Host determines if device is capable of high speed (using chirps)
- Hub establishes a signal path
- Host requests 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

Blocking Write Operations (sometimes)

- Some flash drives have write-protect switches (somewhat rare)
- HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\StorageDevicePolicies\ WriteProtect
  - Blocks writing to ALL USB devices
- Commercial write-blockers
- Microcontroller-based device (discussed later)

Who Was Here?

- Windows records all USB device connections in registry
- Utilities such as USBDeview will easily display this information



Device Name	Description	Devic...	C...	Safe...	Di...	U...	D.	Serial Number	Created Date	Last Plug/Unplug D...	Ven...	Produc...	Firm...	US...	U...	US
HP Integrated Module	HP Integrated Module with Bluet...	Bluetooth D...	Yes	Yes	No	No			6/10/2010 6:27:00 ...	2/8/2011 3:51:08 PM	03f0	171d	1.00	e0	01	01
	MSP430 Application UART	Communica...	No	Yes	No	No			10/7/2010 10:20:1...	N/A	0451	f432	1.00	02	02	01
	USB Human Interface Device	HID (Huma...	No	Yes	No	No			10/7/2010 10:20:1...	N/A	0451	f432	1.00	03	00	00
USB-PS/2 Optical Mouse	USB Human Interface Device	HID (Huma...	No	Yes	No	No			6/10/2010 6:27:00 ...	N/A	046d	c03d	20.00	03	01	02
Cruzer Micro	SanDisk Cruzer Micro USB Device	Mass Storage	No	No	No	No	E:	2004432360057222C776	2/11/2011 8:13:26 PM	2/17/2011 3:11:25 PM	0781	5151	0.10	08	06	50
DataTraveler 2.0	Kingston DataTraveler 2.0 USB ...	Mass Storage	No	No	No	No	F:	001D92A85EAC5B8B050A05EF	12/9/2010 10:53:3...	N/A	0930	6545	1.00	08	06	50
External USB HDD	Toshiba External USB HDD USB ...	Mass Storage	No	No	No	No		37394C335330454753202020	6/10/2010 7:45:31 PM	N/A	0480	a001	1.04	08	06	50
FreeAgent GoFlex	Seagate FreeAgent GoFlex USB ...	Mass Storage	No	No	No	No		NA05597H	11/15/2010 3:45:1...	N/A	0bc2	5031	1.00	08	06	50
HP Flash Media Reader	HP USB20 HS-COMBO USB Device	Mass Storage	No	No	No	No		070706100002	7/16/2010 7:27:13 PM	N/A	03f0	0423	5.58	08	06	50
NOOK	B&N NOOK USB Device	Mass Storage	No	No	No	No		372041756775	11/27/2010 10:09:...	2/13/2011 6:32:33 PM	2080	0001	3.22	08	06	50
Rocketfish	Rocketfish USB Device	Mass Storage	No	No	No	No		AA0B124B00000931	1/19/2011 1:03:56 ...	N/A	19ff	0223	81.92	08	06	50
TEAC FD-05PUB	TEAC USB Floppy	Mass Storage	No	No	No	No			6/10/2010 6:27:00 ...	N/A	0644	0000	0.00	08	04	00
USB 2.0 FD	PNY USB 2.0 FD USB Device	Mass Storage	No	No	No	No	G:	UTYM0832030481	11/15/2010 3:53:5...	N/A	154b	0040	1.00	08	06	50
USB 2.0 FD	PNY USB 2.0 FD USB Device	Mass Storage	No	No	No	No	H:	UTYM086400F364	11/15/2010 3:54:4...	N/A	154b	0040	1.00	08	06	50
USB Flash Drive	UFD USB Flash Drive USB Device	Mass Storage	No	No	No	No		AA04011600000272	11/8/2010 10:36:0...	N/A	05dc	a740	11.00	08	06	50
USB Mass Storage Device	Ut163 USB2FlashStorage USB D...	Mass Storage	No	No	No	No		00000000000E3B	6/10/2010 6:27:00 ...	N/A	0b27	0163	1.00	08	06	50
USB Mass Storage Device	Ut163 USB2FlashStorage USB D...	Mass Storage	No	No	No	No		00000000000E90	6/10/2010 6:27:00 ...	N/A	0b27	0163	1.00	08	06	50
USB Storage	TOSHIBA MK1032GAX USB Device	Mass Storage	No	No	No	No		000000000033	1/4/2011 7:20:59 PM	N/A	05e3	0718	0.41	08	06	50
v115w	HP v115w USB Device	Mass Storage	No	No	No	No		000000000006A	8/26/2010 4:04:27 PM	2/16/2011 2:42:37 PM	03f0	3d07	1.00	08	06	50
Texas Instruments MSP	USB Composite Device	Unknown	No	Yes	No	No		02FE41F50F90290C	10/7/2010 10:20:1	2/17/2011 2:54:07 PM	0451	f432	1.00	00	00	00

23 item(s), 1 Selected | NirSoft Freeware. http://www.nirsoft.net | usb.ids is not loaded

Forensics

- Flash Drive as Memory
- Flash Drive as Storage Media

Flash Drive as Memory

- 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

Flash Drive as Storage Media

- 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
  - Should check for additional logical units (max LUN >0)
- Should check reported versus 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

Fun(?) with Microcontrollers

- Chip Choice
- Talking to Flash Drives
- A Simple Duplicator
- Creating an Image Without a Computer
- Computer Connected Microcontroller

Chip Choice

- FTDI Vinculum II dual USB host 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

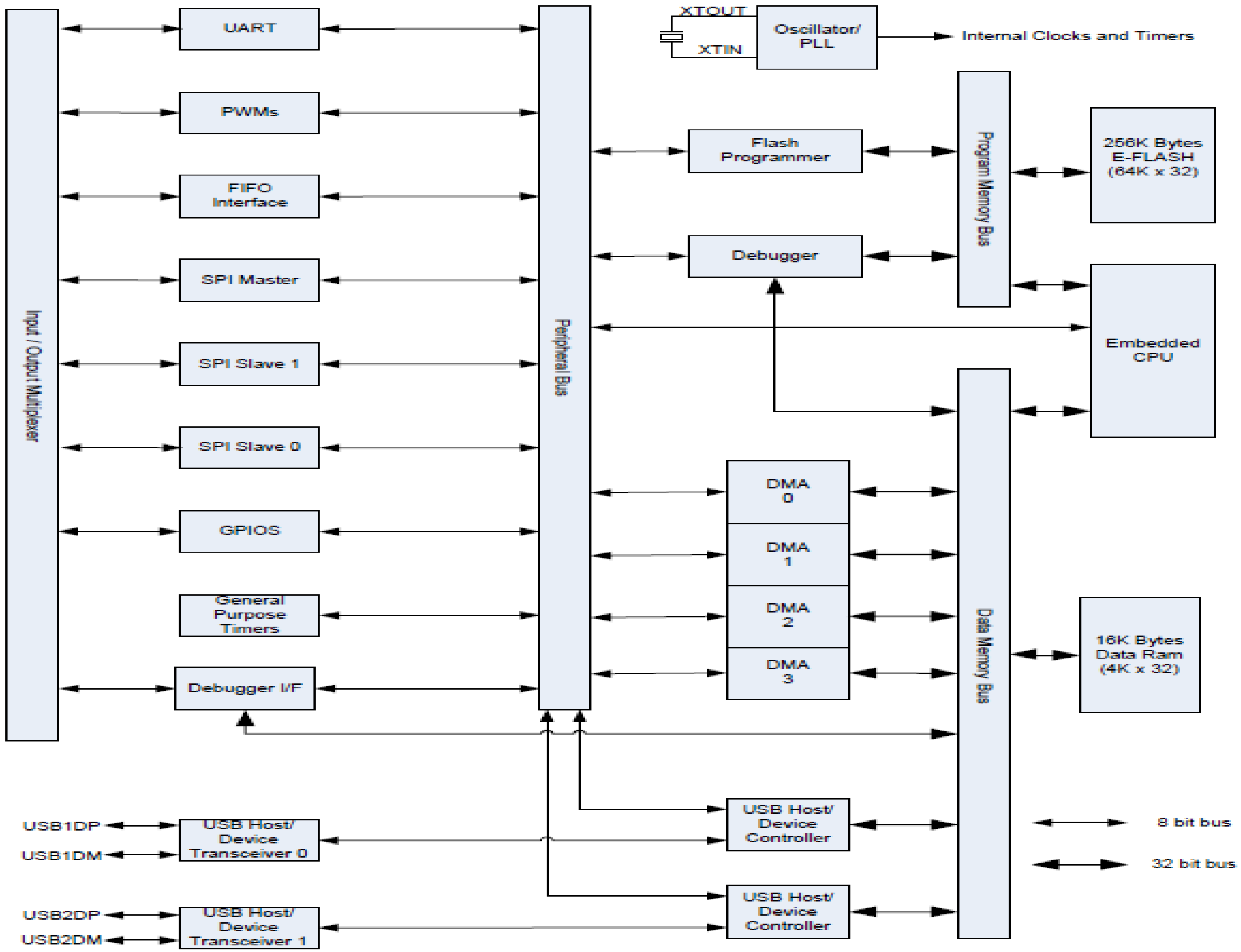
Chip Choice (continued)

FTDI Chip

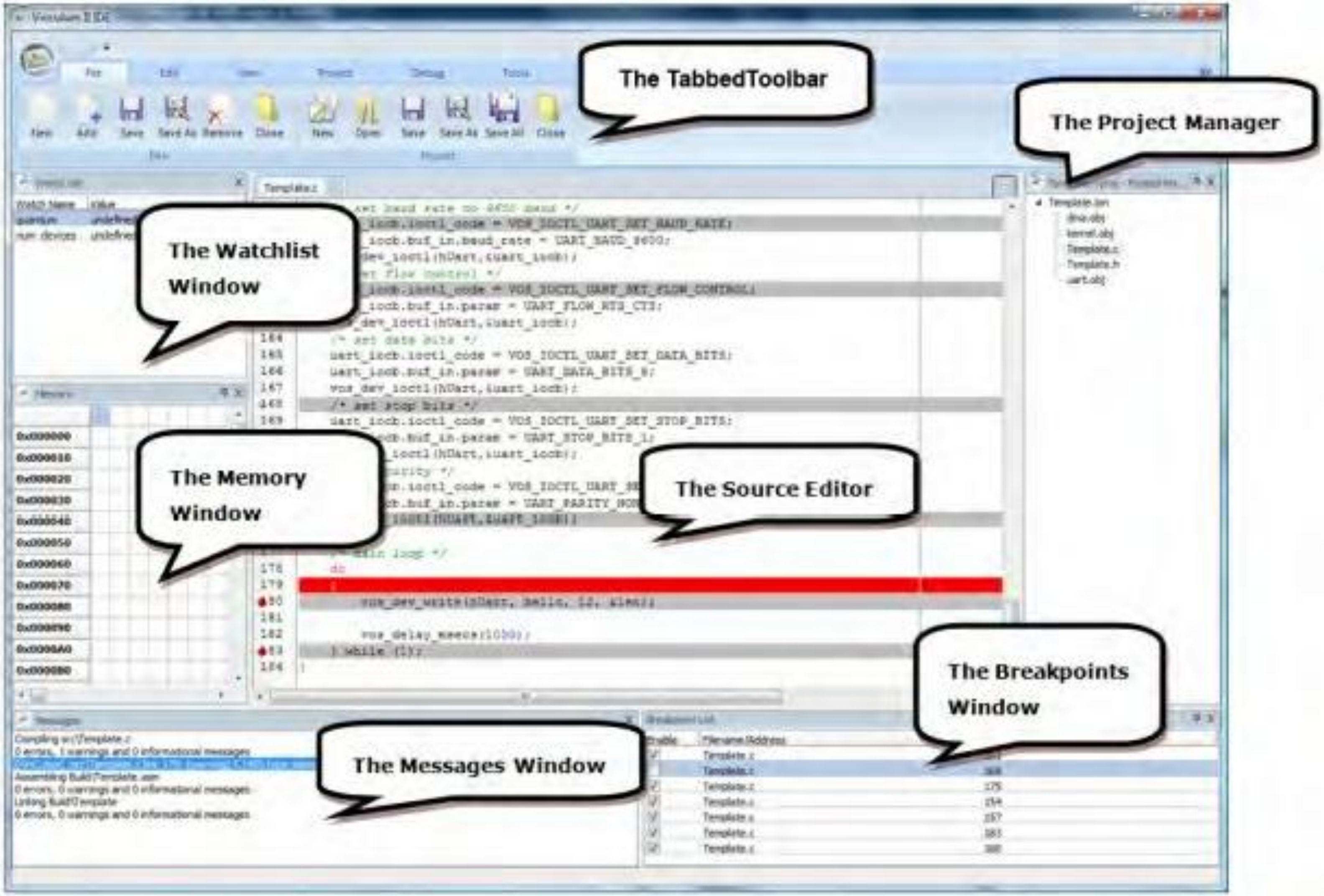
VINCULUM

BINDING USB TECHNOLOGIES

Chip Choice (continued)



Chip Choices (continued)



```
Template.c
 Template.bin
 drvr.obj
 kernel.obj
 Template.c
 Template.h
 uart.obj
```

Watch Name	Value
quantum	undefined
num_devices	undefined

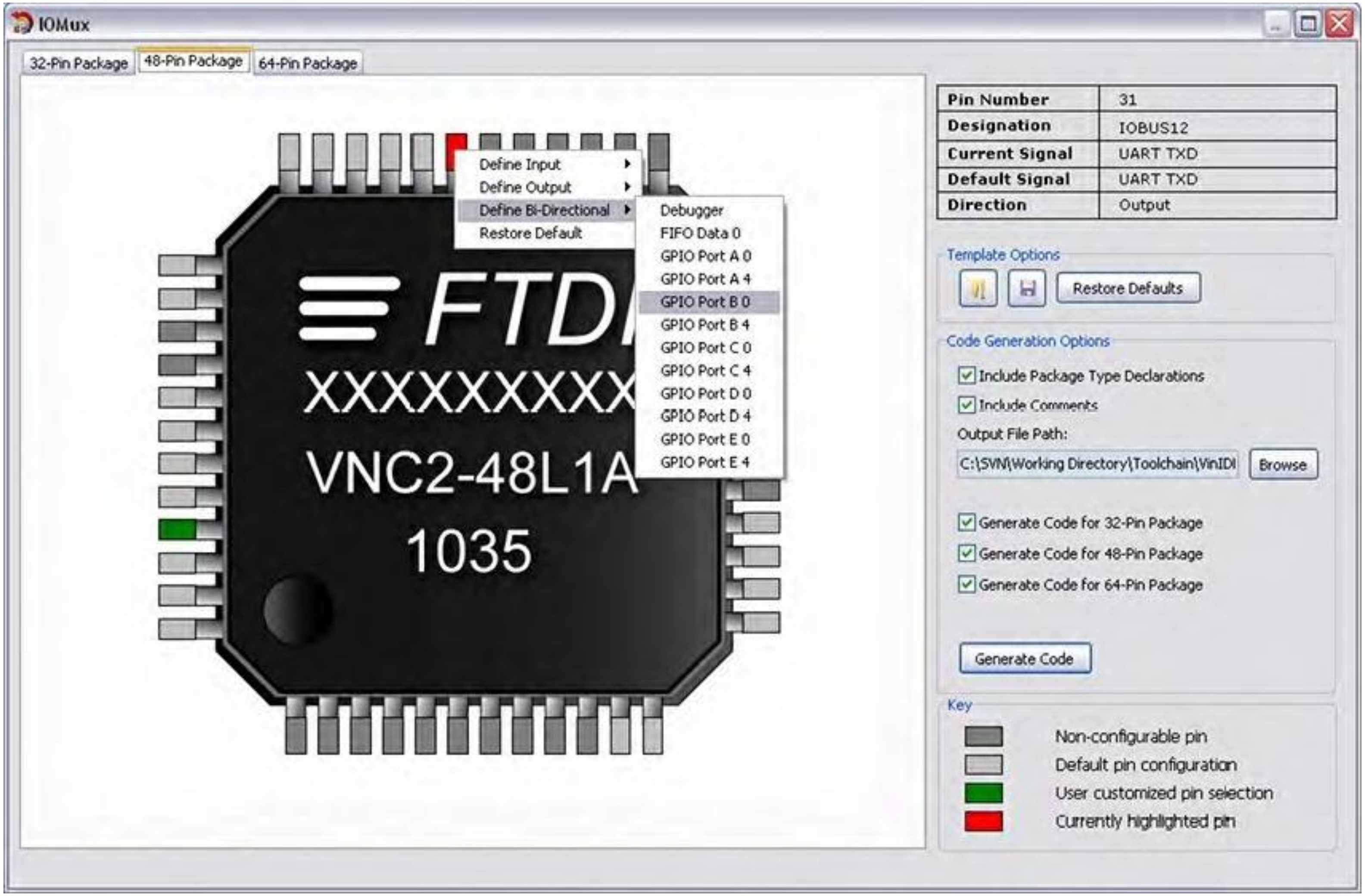
```
0x000000
0x000010
0x000020
0x000030
0x000040
0x000050
0x000060
0x000070
0x000080
0x000090
0x0000A0
0x0000B0
```

```
    /* set baud rate to 9600 baud */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_BAUD_RATE;
    uart_iocb.buf_in.baud_rate = UART_BAUD_9600;
    vos_dev_ioctl(hUart,&uart_iocb);
    /* set flow control */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_FLOW_CONTROL;
    uart_iocb.buf_in.param = UART_FLOW_RTS_CTS;
    vos_dev_ioctl(hUart,&uart_iocb);
    /* set data bits */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_DATA_BITS;
    uart_iocb.buf_in.param = UART_DATA_BITS_8;
    vos_dev_ioctl(hUart,&uart_iocb);
    /* set stop bits */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SET_STOP_BITS;
    uart_iocb.buf_in.param = UART_STOP_BITS_1;
    vos_dev_ioctl(hUart,&uart_iocb);
    /* set parity */
    uart_iocb.ioctl_code = VOS_IOCTL_UART_SE
    uart_iocb.buf_in.param = UART_PARITY_NON
    vos_dev_ioctl(hUart,&uart_iocb);

    /* main loop */
    do
    {
        vos_dev_write(hUart, hello, 12, &len);

        vos_delay_msecs(1000);
    } while (1);
}
```

```
Compiling src\Template.c
0 errors, 1 warnings and 0 informational messages
[illegible]
Assembling Build\Template.asm
0 errors, 0 warnings and 0 informational messages
Linking Build\Template
0 errors, 0 warnings and 0 informational messages
```

Enable	Filename/Address	
☑	Template.c	[illegible]
	Template.c	168
☑	Template.c	175
☑	Template.c	154
☑	Template.c	157
☑	Template.c	183
☑	Template.c	188

Chip Choice (continued)



Pin Number	31
Designation	IOBUS12
Current Signal	UART TXD
Default Signal	UART TXD
Direction	Output

Template Options

Code Generation Options

☑ Include Package Type Declarations

☑ Include Comments

Output File Path:

C:\SVN\Working Directory\Toolchain\VinIDI

☑ Generate Code for 32-Pin Package

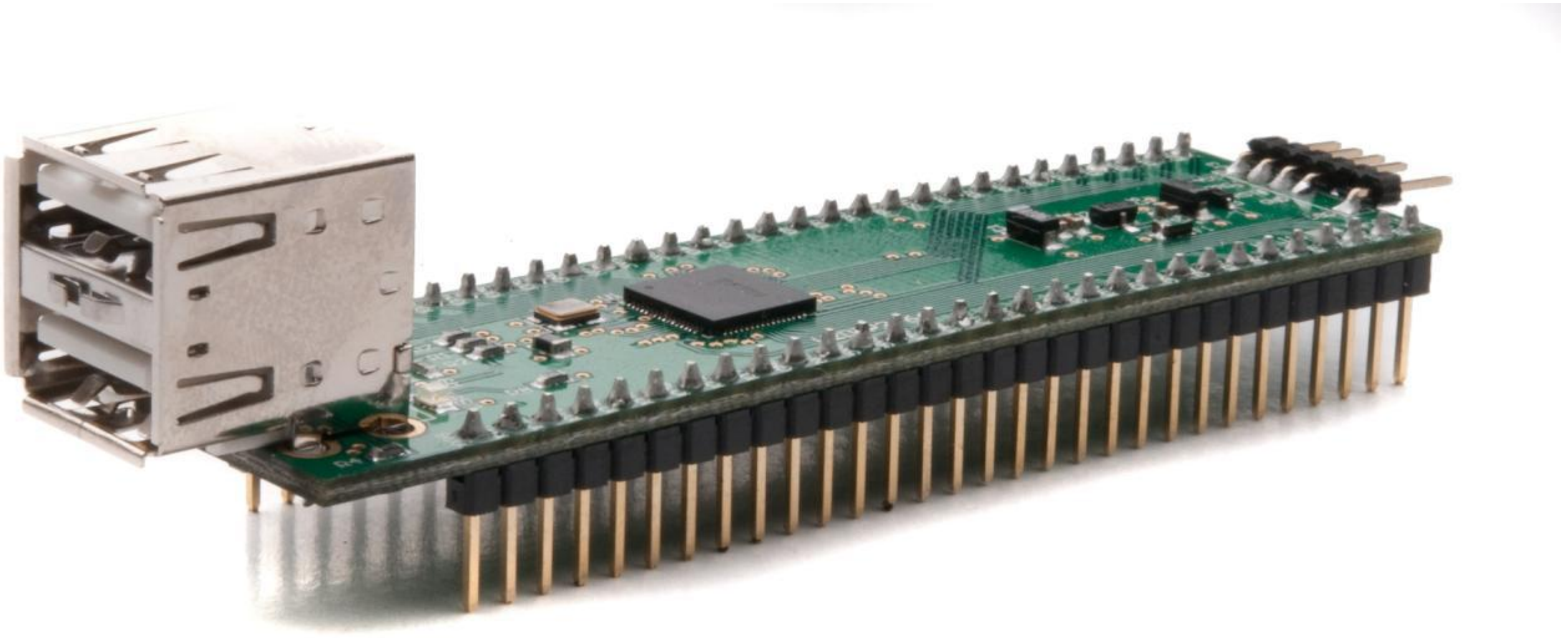
☑ Generate Code for 48-Pin Package

☑ Generate Code for 64-Pin Package

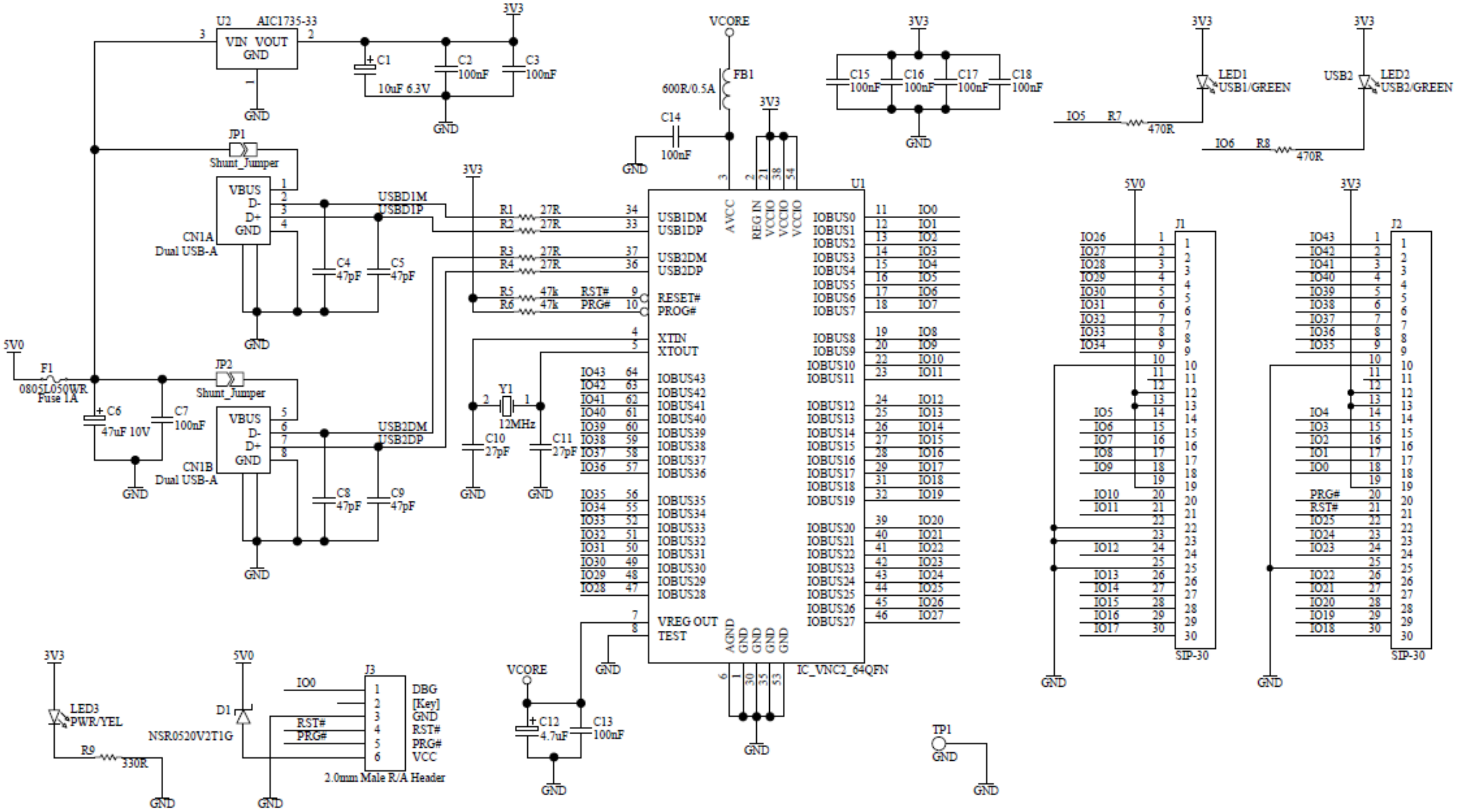
Key

▨	Non-configurable pin
□	Default pin configuration
🟩	User customized pin selection
🟥	Currently highlighted pin

Chip Choice (continued)



Chip Choice (continued)



Jumper Box		
JP1	CLOSED	CN1-A Host
JP1	OPEN	CN1-A Slave
JP2	CLOSED	CN1-B Host
JP2	OPEN	CN1-B Slave

A Simple Duplicator

- Insert a flash drive to be copied
- Insert a target drive for copy
  - Ideally the identical model
  - Should be at least the same size
  - Should use identical block size
- A sector by sector copy is performed
  - Should work on majority of drives examined
  - Requires approximately 11 minutes/GB

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
    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
    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;
```

A Simple Duplicator (continued)

```
void BOMSFindDevice()
{
  VOS_HANDLE hUsb2, hBoms;
  usbhost_device_handle *ifDev2;
  usbhost_ioctl_cb_t hc_iocb;
  usbhost_ioctl_cb_class hc_iocb_class;
  fat_context fatContext;
  msi_ioctl_cb_t boms_iocb;
  boms_ioctl_cb_attach_t boms_att;
  // find BOMS class device
  hc_iocb_class.dev_class = USB_CLASS_MASS_STORAGE;
  hc_iocb_class.dev_subclass = USB_SUBCLASS_MASS_STORAGE_SCSI;
  hc_iocb_class.dev_protocol = USB_PROTOCOL_MASS_STORAGE_BOMS;
  hc_iocb.ioctl_code = VOS_IOCTL_USBHOST_DEVICE_FIND_HANDLE_BY_CLASS;
  hc_iocb.handle.dif = NULL;
  hc_iocb.set = &hc_iocb_class;
  hc_iocb.get = &ifDev2;
  if (vos_dev_ioctl(hUsb2, &hc_iocb) != USBHOST_OK)
  {
    // no BOMS class found
  }
  // now we have a device, intialise a BOMS driver for it
  hBoms = vos_dev_open(VOS_DEV_BOMS);
  // boms_attach
  boms_att.hc_handle = hUsb2;
  boms_att.ifDev = ifDev2;
  boms_iocb.ioctl_code = MSI_IOCTL_BOMS_ATTACH;
  boms_iocb.set = &boms_att;
  boms_iocb.get = NULL;
  if (vos_dev_ioctl(hBoms, &boms_iocb) != MSI_OK)
  {
    // could not attach to device
  }
  // device has been found and opened
  // now detach from the device
  boms_iocb.ioctl_code = MSI_IOCTL_BOMS_DETACH;
  vos_dev_ioctl(hBoms, &boms_iocb)
}
```

A Simple Duplicator (continued)

```
VOS_DEVICE hBoms;

unsigned char fat_readSector(unsigned long sector, char *buffer)
{
  // transfer buffer
  msi_xfer_cb_t xfer;
  // completion semaphore
  vos_semaphore_t semRead;
  unsigned char status;
  vos_init_semaphore(&semRead, 0);
  xfer.sector = sector;
  xfer.buf = buffer;
  // 512 byte sector specific to keep it simple
  xfer.total_len = 512;
  xfer.buf_len = 512;
  xfer.status = MSI_NOT_ACCESSED;
  xfer.s = &semRead;
  xfer.do_phases = MSI_PHASE_ALL;
  status = vos_dev_read(hBoms, (unsigned char *)&xfer, sizeof(msi_xfer_cb_t ), NULL);
  If (status == MSI_OK)
  {
    status = FAT_OK;
  }
  else
  {
    status |= FAT_MSI_ERROR;
  }
  return status;

}
```

Creating an Image without a Computer

- Insert a drive to be imaged
- Attach a USB external hard drive (may require own power)
- An appropriate image file is automatically created on the hard drive

Computer Connected Microcontro

- Capable of simple copy and image creation without attachment to a computer
- Interfaced to an Arduino board via SPI
  - Arduino has become very popular thanks to ease of use
  - Large number of Arduino libraries are available
  - Arduino USB connection to PC is used for communication/control
- Accepts commands from the PC
- Provides status to the PC
- Allows guaranteed write-blocked access to the USB drive

- FTDI has released new VNC2-based Arduino clone: Vinculo
  - Arduino form factor with additional row of pins
  - Can use Arduino shields or expanded Vinculo shields
  - Requires VNC2 Debug Module to program
  - Forces one USB port to be a slave (for PC connection)
  - Should be fairly easy to use as a write-blocker
  - Interesting possibilities to interface with a VNC2 development module
    - 3 USB hosts
    - PC Connection
    - 2 Microcontrollers
    - Could reduce source and destination confusion

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
- Real Digital Forensics by Keith Jones, et. al
- Windows Forensic Analysis (2nd ed.) by Harlan Carvey
- http://www.arduino.cc
- File System Forensic Analysis by Brian Carrier

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

Questions?