USB Mass Storage

A USB Flash Drive with VUSB

How to get Information?

http://www.usb.org

Universal Serial Bus Mass Storage Class

Specification Overview

Universal Serial Bus Mass Storage Class

UFI Command Specification

Universal Serial Bus Mass Storage Class

Bulk-Only Transport

VUSB and Bulk?

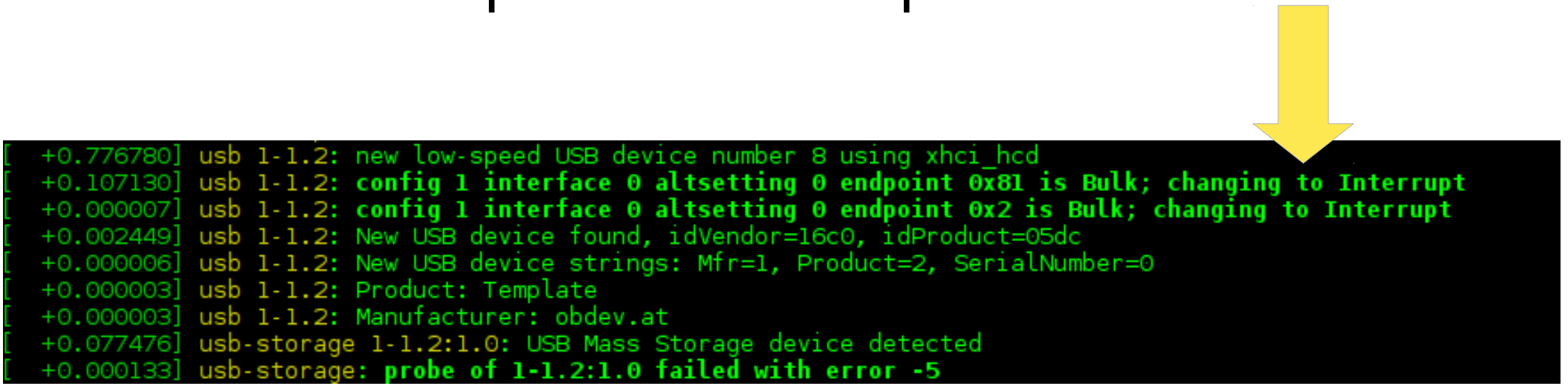
```
/* Name: usbdrv.h
 * Project: V-USB, virtual USB port for Atmel's(r) AVR(r) microcontrollers
 * Author: Christian Starkjohann
 * Creation Date: 2004-12-29
 * Tabsize: 4
 * Copyright: (c) 2005 by OBJECTIVE DEVELOPMENT Software GmbH
 * License: GNU GPL v2 (see License.txt), GNU GPL v3 or proprietary
 */

Number of endpoints:
The driver supports the following endpoints:

- Endpoint 0, the default control endpoint.
- Any number of interrupt- or bulk-out endpoints. The data is sent to
  usbFunctionWriteOut() and USB_CFG_IMPLEMENT_FN_WRITEOUT must be defined
  to 1 to activate this feature. The endpoint number can be found in the
  global variable 'usbRxToken'.
- One default interrupt- or bulk-in endpoint. This endpoint is used for
  interrupt- or bulk-in transfers which are not handled by any other endpoint.
  You must define USB_CFG_HAVE_INTRIN_ENDPOINT in order to activate this
  feature and call usbSetInterrupt() to send interrupt/bulk data.
```

Linux Bulk

Test Low Speed Bulk Endpoints with Linux



→ Restricts Low-Speed Bulk Endpoints

```
240                 d->bEndpointAddress, d->bInterval, n);
241             endpoint->desc.bInterval = n;
242     }
243
244     /* Some buggy low-speed devices have Bulk endpoints, which is
245      * explicitly forbidden by the USB spec.  In an attempt to make
246      * them usable, we will try treating them as Interrupt endpoints.
247      */
248     if (to_usb_device(ddev)->speed == USB_SPEED_LOW &&
249                 usb_endpoint_xfer_bulk(d)) {
250             dev_warn(ddev, "config %d interface %d altsetting %d "
251                 "endpoint 0x%X is Bulk; changing to Interrupt\n",
252                 cfgno, inum, asnum, d->bEndpointAddress);
253             endpoint->desc.bmAttributes = USB_ENDPOINT_XFER_INT;
254             endpoint->desc.bInterval = 1;
255             if (usb_endpoint_maxp(&endpoint->desc) > 8)
256                 endpoint->desc.wMaxPacketSize = cpu_to_le16(8);
257     }
258
259     /*
260      * Some buggy high speed devices have bulk endpoints using
261      * maxpacket sizes other than 512.  High speed HCDs may not
262      * be able to handle that particular bug, so let's warn...
263      */
264     if (to_usb_device(ddev)->speed == USB_SPEED_HIGH
265                 && usb_endpoint_xfer_bulk(d)) {
```

```
240                 d->bEndpointAddress, d->bInterval, n);
241             endpoint->desc.bInterval = n;
242         }
243
244         /* Some buggy low-speed devices have Bulk endpoints, which is
245          * explicitly forbidden by the USB spec.  In an attempt to make
246          * them usable, we will try treating them as Interrupt endpoints.
247          */
248         if (to_usb_device(ddev)->speed == USB_SPEED_LOW &&
249                         usb_endpoint_xfer_bulk(d)) {
250             dev_warn(ddev, "config %d interface %d altsetting %d "
251                 "endpoint 0x%X is Bulk;NOT changing to Interrupt\n",
252                 cfgno, inum, asnum, d->bEndpointAddress);
253
254
255
256
257         }
258
259         /*
260          * Some buggy high speed devices have bulk endpoints using
261          * maxpacket sizes other than 512.  High speed HCDs may not
262          * be able to handle that particular bug, so let's warn...
263          */
264         if (to_usb_device(ddev)->speed == USB_SPEED_HIGH
265                         && usb_endpoint_xfer_bulk(d)) {
```

Modified Kernel

Recompile kernel – 2 hours later

How is it done?

AVR:

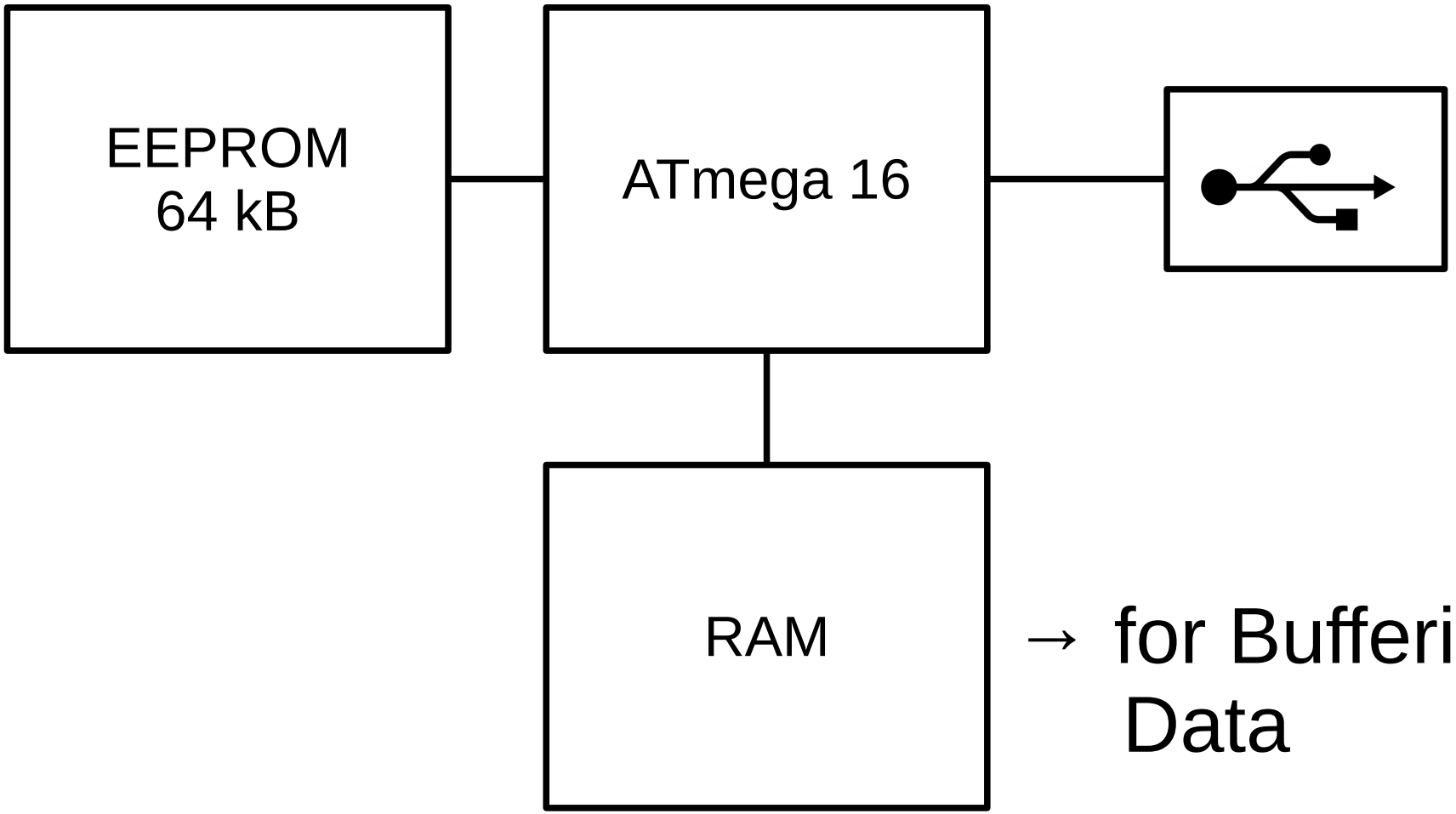
- Implement Descriptor
- Implement Commands
- Read & Write EEPROM

Command Status Wrapper

Table 5.2 - Command Status Wrapper

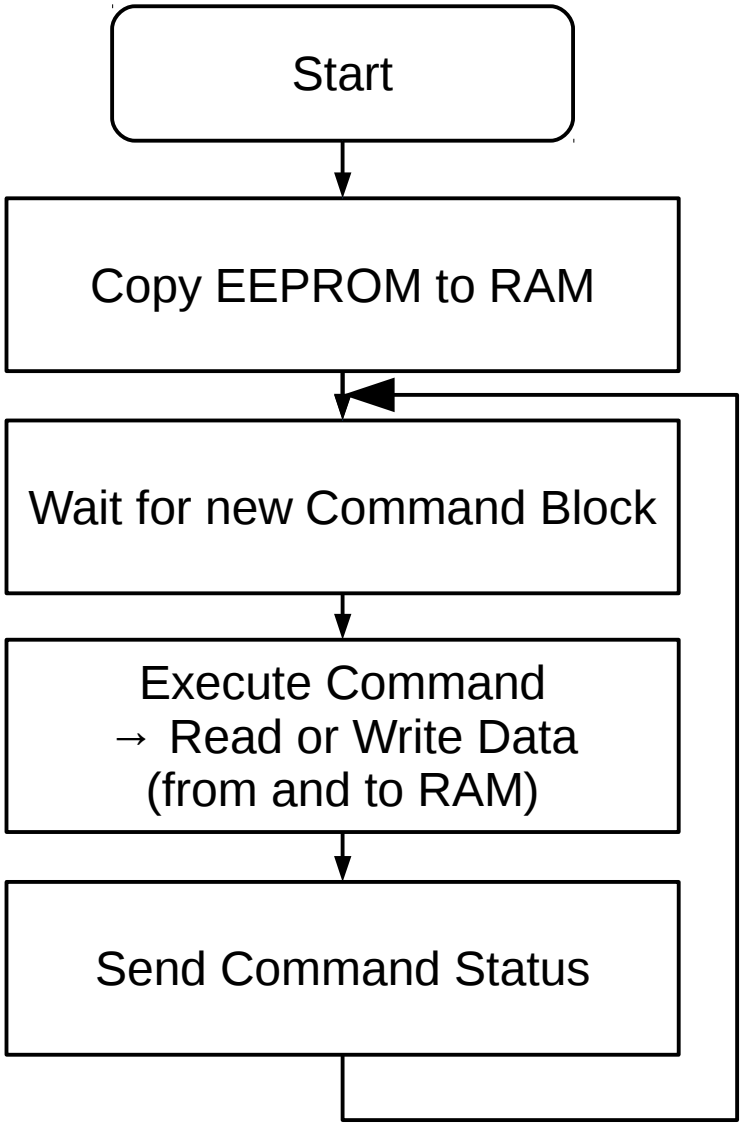
bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

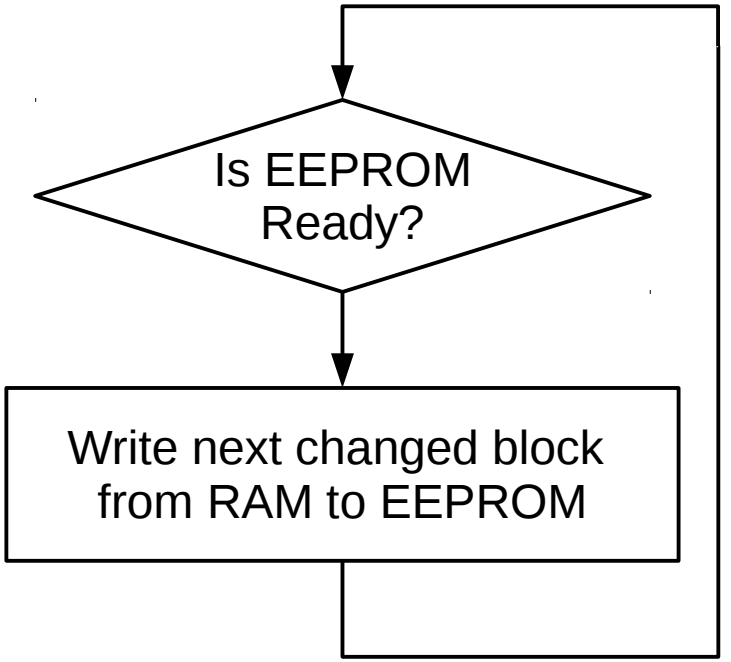
With RAM



→ for Buffering Data

Software


Start
Copy EEPROM to RAM
Wait for new Command Block
Execute Command
→ Read or Write Data
(from and to RAM)
Send Command Status


Is EEPROM
Ready?
Write next changed block
from RAM to EEPROM

Command Implementation

- Receive with usbFunctionWriteOut()
- Recognise command and create correct reponse
- Send with usbSetInterrupt()

Command Wrapper

Table 5.1 - Command Block Wrapper

Byte \ bit	7	6	5	4	3	2	1	0
0-3	*dCBWSignature*							
4-7	*dCBWTag*							
8-11 (08h-0Bh)	*dCBWDataTransferLength*							
12 (0Ch)	*bmCBWFlags*							
13 (0Dh)	Reserved (0)				*bCBWLUN*			
14 (0Eh)	Reserved (0)			*bCBWCBLength*				
15-30 (0Fh-1Eh)	*CBWCB*							

Command Status Wrapper

Table 5.2 - Command Status Wrapper

bit / Byte	7	6	5	4	3	2	1	0
0-3	*dCSWSignature*							
4-7	*dCSWTag*							
8-11 (8-Bh)	*dCSWDataResidue*							
12 (Ch)	*bCSWStatus*							

SCSI like – Command Set

Read (10)	Transfer binary data from the media to the host.	28h
Read (12)	Transfer binary data from the media to the host.	A8h
Read Capacity	Report current media capacity.	25h
Read Format Capacity	Report current media capacity and formattable capacities supported by media.	23h
Request Sense	Transfer status sense data to the host.	03h
Rezero Unit	Position a head of the drive to zero track .	01h
Seek (10)	Seek the device to a specified address.	2Bh
Send Diagnostic	Perform a hard reset and execute diagnostics.	1Dh
Test Unit Ready	Request the device to report if it is ready.	00h
Verify	Verify data on the media.	2Fh
Write (10)	Transfer binary data from the host to the media.	2Ah

Read

Table 25 - READ(10) Command

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (28h)							
1	Logical Unit Number			DPO	FUA	Reserved		RelAdr
2	(MSB)							
3	Logical Block Address							
4								
5								(LSB)
6	Reserved							
7	Transfer Length (MSB)							
8	Transfer Length (LSB)							
9	Reserved							
10	Reserved							
11	Reserved							