

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**

**Universal Serial Bus
Mass Storage Class

Bulk-Only Transport**

**UFI Command
Specification**

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
 */
```

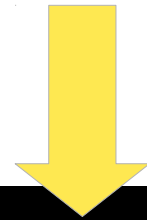
Number of endpoints:

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



```
[ +0.776780] usb 1-1.2: new low-speed USB device number 8 using xhci_hcd
[ +0.107130] usb 1-1.2: config 1 interface 0 altsetting 0 endpoint 0x81 is Bulk; changing to Interrupt
[ +0.000007] usb 1-1.2: config 1 interface 0 altsetting 0 endpoint 0x2 is Bulk; changing to Interrupt
[ +0.002449] usb 1-1.2: New USB device found, idVendor=16c0, idProduct=05dc
[ +0.000006] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ +0.000003] usb 1-1.2: Product: Template
[ +0.000003] usb 1-1.2: Manufacturer: obdev.at
[ +0.077476] usb-storage 1-1.2:1.0: USB Mass Storage device detected
[ +0.000133] usb-storage: probe of 1-1.2:1.0 failed with error -5
```

→ Restricts Low-Speed Bulk Endpoints



Linux Cross Reference

Free Electrons

Embedded Linux Experts

• [Source Navigation](#) • [Diff Markup](#) • [Identifier Search](#) • [Freetext Search](#) •

Version: [2.0.40](#) [2.2.26](#) [2.4.37](#) [3.8](#) [3.9](#) [3.10](#) [3.11](#) [3.12](#) [3.13](#) [3.14](#) [3.15](#) [3.16](#) [3.17](#) [3.18](#) [3.19](#) [4.0](#) [4.1](#) [4.2](#) [4.3](#) [4.4](#)

```
239         cfgno, inum, asnum,
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)) {
```




Linux Cross Reference

Free Electrons

Embedded Linux Experts

• [Source Navigation](#) • [Diff Markup](#) • [Identifier Search](#) • [Freetext Search](#) •

Version: [2.0.40](#) [2.2.26](#) [2.4.37](#) [3.8](#) [3.9](#) [3.10](#) [3.11](#) [3.12](#) [3.13](#) [3.14](#) [3.15](#) [3.16](#) [3.17](#) [3.18](#) [3.19](#) [4.0](#) [4.1](#) [4.2](#) [4.3](#) [4.4](#)

```
239         cfgno, inum, asnum,
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)) {
```

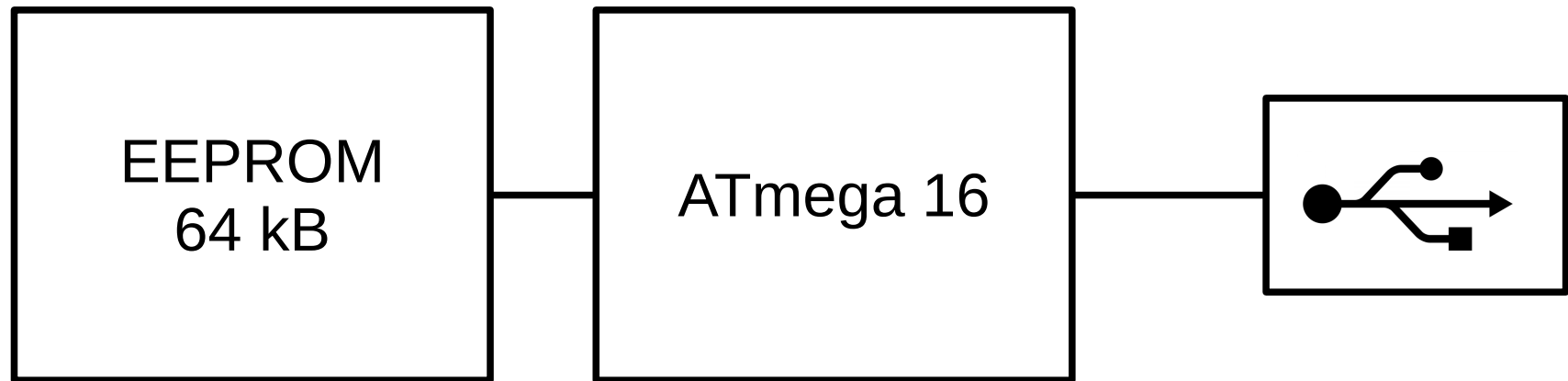
Modified Kernel

Recompile kernel – 2 hours later



```
[ +0.555993] usb 3-8: new low-speed USB device number 21 using xhci_hcd
[ +0.187602] usb 3-8: config 1 interface 0 altsetting 0 endpoint 0x81 is Bulk; NOT changing to Interrupt
[ +0.000002] usb 3-8: config 1 interface 0 altsetting 0 endpoint 0x2 is Bulk; NOT changing to Interrupt
[ +0.002678] usb 3-8: New USB device found, idVendor=16c0, idProduct=05dc
[ +0.000002] usb 3-8: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ +0.000001] usb 3-8: Product: Template
[ +0.000001] usb 3-8: Manufacturer: obdev.at
[ +0.000658] usb-storage 3-8:1.0: USB Mass Storage device detected
[ +0.000171] Vendor: 0x16c0, Product: 0x05dc, Revision: 0x0100
[ +0.000012] Interface Subclass: 0x06, Protocol: 0x50
[ +0.000003] Transport: Bulk
[ +0.000001] Protocol: Transparent SCSI
[ +0.000051] scsi host9: usb-storage 3-8:1.0
[ +0.000001] *** thread sleeping
[ +0.000043] usb-storage 3-8:1.0: waiting for device to settle before scanning
```

How is it done?



AVR:

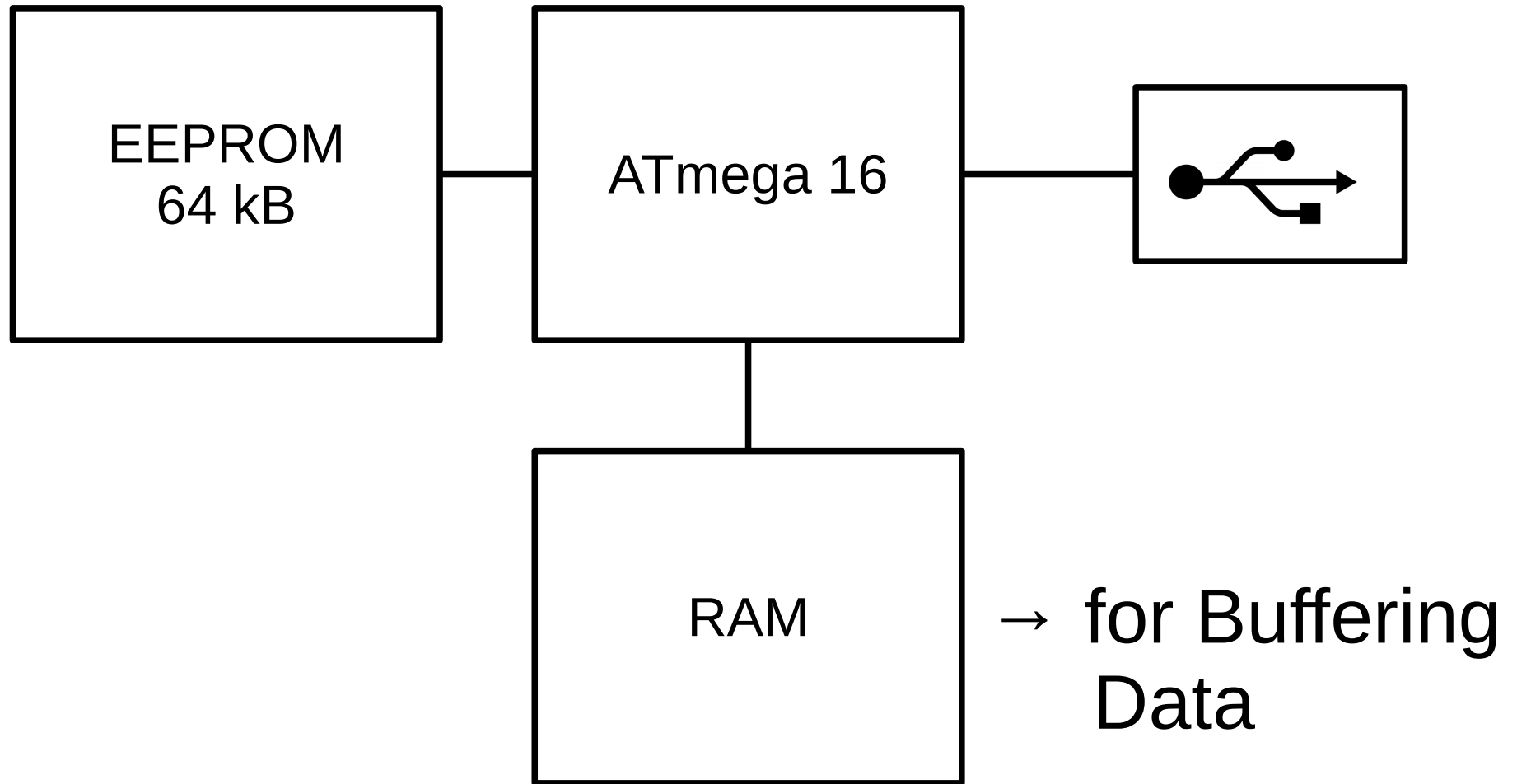
- Implement Descriptor
- Implement Commands
- Read & Write EEPROM

Command Status Wrapper

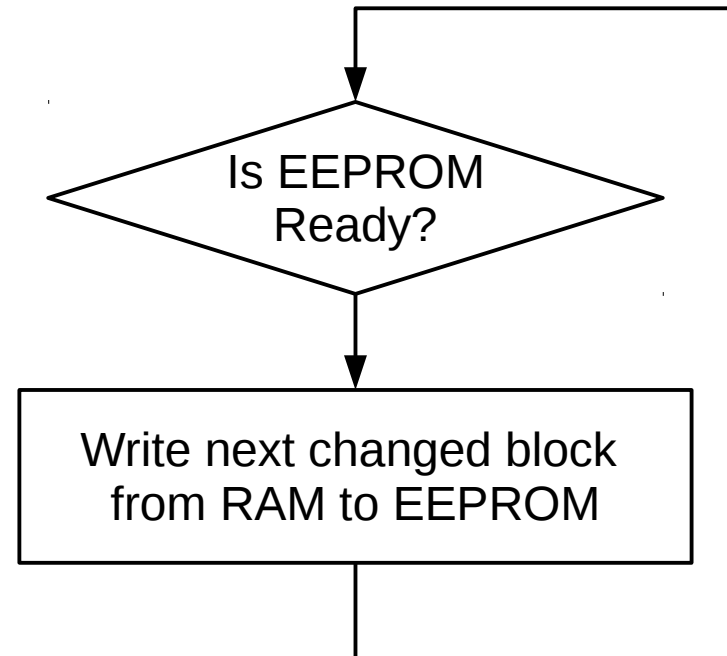
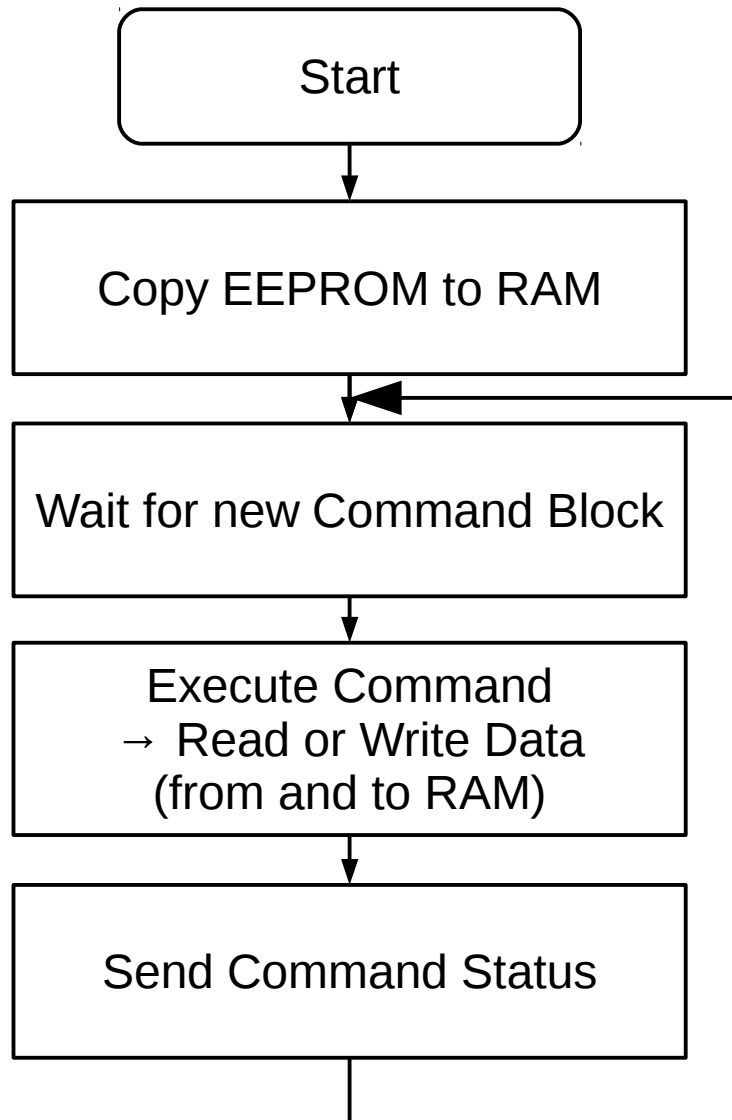
Table 5.2 - Command Status Wrapper

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

With RAM



Software



Command Implementation

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

Command Wrapper

Table 5.1 - Command Block Wrapper

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

Command Status Wrapper

Table 5.2 - Command Status Wrapper

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

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) Logical Block Address (LSB)							
3								
4								
5								
6	Reserved							
7	Transfer Length (MSB)							
8	Transfer Length (LSB)							
9	Reserved							
10	Reserved							
11	Reserved							