USBTMC
1

Department of Metrology and Optoelectronics

Electronic Systems Programming

Lecture title

Programming of measurement instrumentation in Linux

USBTMC
2

Programming of measurement instrumentation in Linux

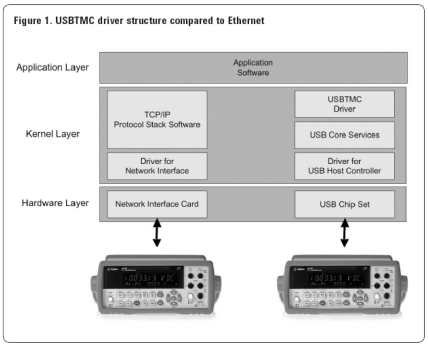
Topics:

1. Overview of USBTMC
2. Interface endpoints
3. Registration with the USB Core
4. Access to the Driver from User Space
5. Installing the USBTMC Driver
6. Using the USBTMC Driver

USBTMC
3

Overview of USBTMC

USBTMC communication model

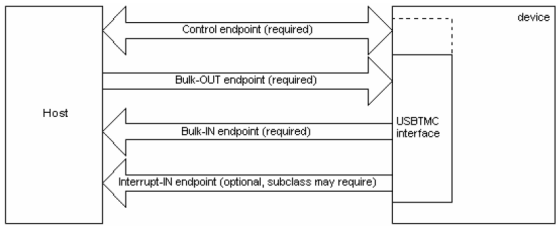


USBTMC
4

Overview of USBTMC

USBTMC communication model

Control endpoint is used to to send standard, class, and vendor-specific requests to the device.

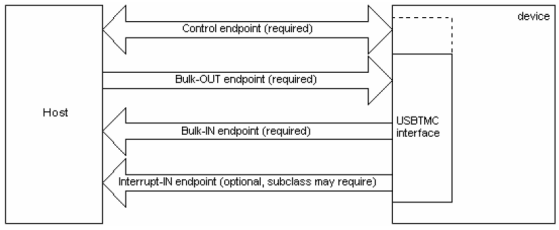


USBTMC
5

Overview of USBTMC

USBTMC communication model

Bulk-OUT endpoint is used to send USBTMC command messages to the device, and the device must process the USBTMC command messages in the order they are received.

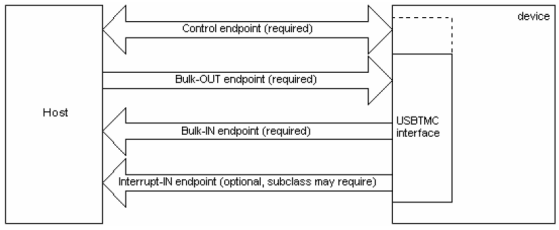


USBTMC
6

Overview of USBTMC

USBTMC communication model

Bulk-IN endpoint is used to receive USBTMC response messages from the device.

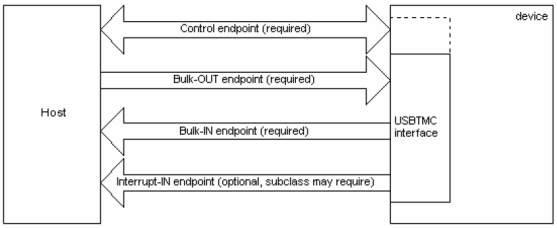


USBTMC
7

Overview of USBTMC

USBTMC communication model

Interrupt-IN endpoint can be used by the device to send notifications to the Host.



USBTMC
8

Programming of measurement instrumentation in Linux

Topics:

1. Overview of USBTMC
2. **Interface endpoints**
3. Registration with the USB Core
4. Access to the Driver from User Space
5. Installing the USBTMC Driver
6. Using the USBTMC Driver

USBTMC
9

Interface endpoints

Bulk-OUT endpoint

USBTMC message Bulk-OUT Header

Offset	Field	Size	Value	Description
0	MsgID	1	Value	Specifies the USBTMC message and the type of the USBTMC message. See Table 2.
1	bTag	1	Value	A transfer identifier. The Host must set bTag different than the bTag used in the previous Bulk-OUT Header. The Host should increment the bTag by 1 each time it sends a new Bulk-OUT Header. The Host must set bTag such that 1<=bTag<=255.
2	bTagInverse	1	Value	The inverse (one's complement) of the bTag. For example, the bTagInverse of 0x5B is 0xA4.
3	Reserved	1	0x00	Reserved. Must be 0x00.
4-11	USBTMC command message specific	8	USBTMC command message specific	USBTMC command message specific. See section 3.2.1.

USBTMC
10

Interface endpoints

Bulk-OUT endpoint

MsgID values

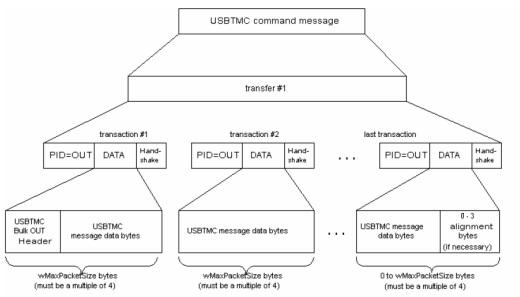
MsgID	Direction OUT=Host-to-device IN=Device-to-Host	MACRO	Description
0	Reserved	Reserved	Reserved
1	OUT	DEV_DEP_MSG_OUT	The USBTMC message is a USBTMC device dependent command message. See section 3.2.1.1.
	IN	(no defined response)	There is no defined response for this USBTMC command message.
2	OUT	REQUEST_DEV_DEP_MSG_IN	The USBTMC message is a USBTMC command message that requests the device to send a USBTMC response message on the Bulk-IN endpoint. See section 3.2.1.2.
	IN	DEV_DEP_MSG_IN	The USBTMC message is a USBTMC response message to the REQUEST_DEV_DEP_MSG_IN. See section 3.3.1.1.
3-125	Reserved	Reserved	Reserved for USBTMC use.
126	OUT	VENDOR_SPECIFIC_OUT	The USBTMC message is a USBTMC vendor specific command message. See section 3.2.1.3.
	IN	(no defined response)	There is no defined response for this USBTMC command message.
127	OUT	REQUEST_VENDOR_SPECIFIC_IN	The USBTMC message is a USBTMC command message that requests the device to send a vendor specific USBTMC response message on the Bulk-IN endpoint. See section 3.2.1.4.
	IN	VENDOR_SPECIFIC_IN	The USBTMC message is a USBTMC response message to the REQUEST_VENDOR_SPECIFIC_IN. See section 3.3.1.2.
128-191	Reserved	Reserved	Reserved for USBTMC subclass use.
192-255	Reserved	Reserved	Reserved for VISA specification use.

USBTMC
11

Interface endpoints

Bulk-OUT endpoint

Bulk-OUT message sent with a single transfer

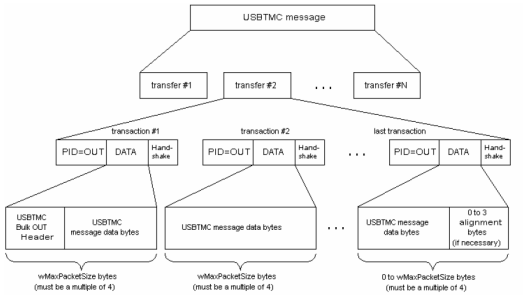


USBTMC
12

Interface endpoints

Bulk-OUT endpoint

Bulk-OUT message sent with multiple transfers



USBTMC
13

Interface endpoints

Bulk-OUT endpoint

Structure of the DEV_DEP_MSG_OUT message (example for *RST command)

Offset (bytes)	Field	Size (bytes)	Value	Description
4...7	TransferSize	4	5	Number of bytes to be transferred (instrument command).
8	bmTransferAttributes	1	0x01	End of message. If bit 0 is set to 1, the instrument message ends with this transfer. Otherwise, the message continues with the next transfer. All other bits are reserved (set to 0).
9...11	Reserved	3	0x000000	Reserved. Set to 0x000000
12...16	Instrument Command	5	"*RST\n"	Instrument command

USBTMC
14

Interface endpoints

Bulk-OUT endpoint

Example code: Sending a SCPI command via a DEV_DEP_MSG_OUT message (1/3)

```
// Setup IO buffer for DEV_DEP_MSG_OUT message
usbtmc_buffer[0]=1; // DEV_DEP_MSG_OUT
usbtmc_buffer[1]=bTag; // Transfer ID (bTag)
usbtmc_buffer[2]=~(bTag); // Inverse of bTag
usbtmc_buffer[3]=0; // Reserved
usbtmc_buffer[4]=command_length&255; // Transfer size (first byte)
usbtmc_buffer[5]=(command_length>>8)&255; //Transfer size (second byte)
usbtmc_buffer[6]=(command_length>>16)&255; //Transfer size (third byte)
usbtmc_buffer[7]=(command_length>>24)&255; //Transfer size (fourth byte)
usbtmc_buffer[8]=1; // Message ends with this transfer
usbtmc_buffer[9]=0; // Reserved
usbtmc_buffer[10]=0; // Reserved
usbtmc_buffer[11]=0; // Reserved
```

USBTMC
15

Interface endpoints

Bulk-OUT endpoint

Example code: Sending a SCPI command via a DEV_DEP_MSG_OUT message (2/3)

```
// Append write buffer (instrument command) to USBTMC message
if(copy_from_user(&(usbtmc_buffer[12]),command_buffer,command_length)) {
        // There must have been an addressing problem
        return -EFAULT;
}

// Add zero bytes to achieve 4-byte alignment
n_bytes=12+command_length;
if(command_length%4) {
        n_bytes+=4-command_length%4;
        for(n=12+command_length;n<n_bytes;n++) usbtmc_buffer[n]=0;
}
```

USBTMC
16

Interface endpoints

Bulk-OUT endpoint

Example code: Sending a SCPI command via a DEV_DEP_MSG_OUT message (3/3)

```
// Create pipe for bulk out transfer
pipe=usb_sndbulkpipe(usb_dev,bulk_out);

// Send bulk URB
retval=usb_bulk_msg(usb_dev,pipe,usbtmc_buffer,n_bytes,
&actual,USBTMC_USB_TIMEOUT);
```

USBTMC
17

Interface endpoints

Bulk-IN endpoint

USBTMC message Bulk-IN Header

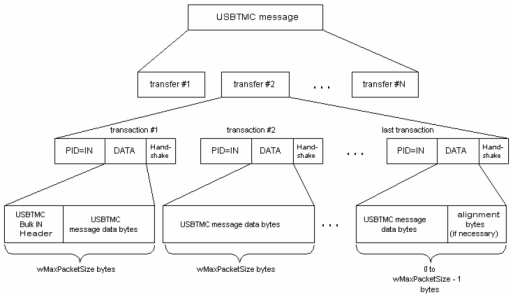
Offset	Field	Size	Value	Description
0	MsgID	1	Value	Must match MsgID in the USBTMC command message transfer causing this response.
1	bTag	1	Value	Must match bTag in the USBTMC command message transfer causing this response.
2	bTagInverse	1	Value	Must match bTagInverse in the USBTMC command message transfer causing this response.
3	Reserved	1	0x00	Reserved. Must be 0x00.
4-11	USBTMC response message specific	8	USBTMC response message specific	USBTMC response message specific. See section 3.3.1.

USBTMC
18

Interface endpoints

Bulk-IN endpoint

Bulk-IN USBTMC message sent with multiple transfers



USBTMC
19

Interface endpoints

Interrupt-IN endpoint

Interrupt-IN DATA payload format

Offset	Field	Size	D7	D6	D5	D4...D0	Explanation
0	bNotify1	1	1	Notification bits			Format of D6...D0 and all notification bytes are defined in the applicable subclass specification
			0	1	Notification bits		Format of D5...D0 and all notification bytes are vendor specific. A Host may ignore vendor specific notifications.
			0	0	Notification bits		Format of D5...D0 and all notification bytes are reserved for USBTMC use.
1	bNotify2	N-1	Notification bytes				Format specified according to D7...D6 in offset 0.

USBTMC
20

Programming of measurement instrumentation in Linux

Topics:

1. Overview of USBTMC
2. Interface endpoints
3. **Registration with the USB Core**
4. Access to the Driver from User Space
5. Installing the USBTMC Driver
6. Using the USBTMC Driver

USBTMC
21

Registration with the USB Core

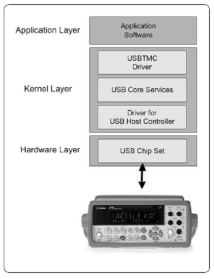
USBTMC driver needs to register with the USB core to interact with it.

A key element of this registration process is telling the USB core which devices an USBTMC driver would like to service when they become available.

Wanted devices can be filtered by various attributes, including the: **devices' vendor ID**, **product ID** or **device class**.

In the context of USBTMC, it is most appropriate to filter by **device class** (application-specific) and **USBTMC subclass**.

The USBTMC driver would then get notified whenever a USBTMC-compatible device is being attached, independent of its vendor or product code.



USBTMC
22

Registration with the USB Core

Example code: Registering a USB higher-level driver with the USB core layer (1/2)

```
static struct usb_device_id usbtmc_devices[] = {
       {.match_flags=USB_DEVICE_ID_MATCH_INT_CLASS |
              USB_DEVICE_ID_MATCH_INT_SUBCLASS,
              // Device class and sub class need to match to be
              // notified by the system
              .bInterfaceClass=254, // 254 = application specific
              .bInterfaceSubClass=3},
                     // 3 = test and measurement class (USBTMC)
       { } // Terminating entry
};
```

USBTMC
23

Registration with the USB Core

Registering a USB higher-level driver with the USB core layer (2/2)

```
static struct usb_driver usbtmc_driver = {
       .name="USBTMC", // Driver name
       .id_table=usbtmc_devices, //Devices serviced by the driver
       .probe=usbtmc_probe, // Probe function
                            // (called when device is connected)
       .disconnect=usbtmc_disconnect // Disconnect function
};

// Register USB driver with USB core
if((retcode=usb_register(&usbtmc_driver))) {
       printk(KERN_ALERT "USBTMC: Unable to register driver\n");
       goto exit_usb_register;
}
```

USBTMC
24

Programming of measurement instrumentation in Linux

Topics:

1. Overview of USBTMC
2. Interface endpoints
3. Registration with the USB Core
4. **Access to the Driver from User Space**
5. Installing the USBTMC Driver
6. Using the USBTMC Driver

Access to the Driver from User Space

USBTMC-compatible instruments are controlled through text commands, typically following the SCPI standard. Hence communicating with a USB instrument is stream-oriented. The most appropriate for such instrument is a character device driver.

Character device drivers need to implement a number of entry points that the system calls in order to interact with a device.

The most basic ones are: **open**, **read**, **write**, **release**.

The **write**() entry point for USBTMC device driver takes the string to be written and wraps it into a **USBTMC DEV_DEP_MSG_OUT** message.

Similarly, the **read**() entry point uses a **DEV_DEP_MSG_IN** message to read data from a device, extract the instrument message part from the return data and copy it to the supplied user buffer.

Access to the Driver from User Space

Device files are created using the **mknod** command, and the ***major number*** specified refers to a character driver behind the (arbitrary) device file name.

The ***minor number*** is typically used to specify which device the driver will control if several devices are being serviced by the same driver.

When a character device driver is loaded into the kernel, it first needs to register its major and minor numbers with the kernel and publish its entry points.

Access to the Driver from User Space

Registering a character device driver (1/3)

```
// Dynamically allocate char driver major/minor numbers
//                              First major/minor number to use

if ((retcode = alloc_chrdev_region(&dev,
   0, // First minor number
   USBTMC_MINOR_NUMBERS,// Number of minor numbers to reserve
   "USBTMCCHR" // Char device driver name
   ))) {
   printk(KERN_ALERT "Unable to allocate major/minor numbers\n");
   goto exit_alloc_chrdev_region;
}
```


Access to the Driver from User Space

Registering a character device driver (2/3)

```
// This structure is used to publish
// the char device driver functions

static struct file_operations fops = {
       .owner=THIS_MODULE,
       .read=usbtmc_read,
       .write=usbtmc_write,
       .open=usbtmc_open,
       .release=usbtmc_release,
       .ioctl=usbtmc_ioctl,
       .llseek=usbtmc_llseek,
};
```


Access to the Driver from User Space

Registering a character device driver (3/3)

```
// Initialize cdev structure for this character device
cdev_init(&cdev,&fops);
cdev.owner = THIS_MODULE;
cdev.ops = &fops;

// Combine major and minor numbers
printk(KERN_NOTICE "USBTMC: MKDEV\n");
devno = MKDEV(MAJOR(dev),n);

// Add character device to kernel list
printk(KERN_NOTICE "USBTMC: CDEV_ADD\n");
if((retcode = cdev_add(&cdev,devno,1))) {
       printk(KERN_ALERT "Unable to add character device\n");
       goto exit_cdev_add;
}
```


Programming of measurement instrumentation in Linux

Topics:

1. Overview of USBTMC
2. Interface endpoints
3. Registration with the USB Core
4. Access to the Driver from User Space
5. **Installing the USBTMC Driver**
6. Using the USBTMC Driver

USBTMC
31

Installing the USBTMC Driver

Compiling the driver

In order to compile source code of the driver „**usbtmc.c**" on the basis **makefile**, one should use **make** command. This will create a kernel object file „**usbtmc.ko**".

Installing the driver

In order to install the driver module in the running kernel one shold use a command

```
insmod ./usbtmc.ko
```

Similarly, the module can be unloaded from the kernel using

```
rmmod usbtmc
```

USBTMC
32

Installing the USBTMC Driver

Installing the driver

To use the driver proper device files must be created, related to the major number the driver uses. A following script can be used to perform that task

```
# Find major number used
major=$(cat /proc/devices | grep USBTMCCHR | awk '{print $1}')
echo Using major number $major
# Remove old device files
rm -f /dev/${module}[0-9]
# Ceate new device files
mknod /dev/${module}0 c $major 0
mknod /dev/${module}1 c $major 1
//...
# Change access mode (RW access for everybody)
chmod 666 /dev/${module}0
chmod 666 /dev/${module}1
//...
```

USBTMC
33

Programming of measurement instrumentation in Linux

Topics:

1. Overview of USBTMC
2. Interface endpoints
3. Registration with the USB Core
4. Access to the Driver from User Space
5. Installing the USBTMC Driver
6. **Using the USBTMC Driver**

USBTMC
34

Using the USBTMC Driver

Interactive instrument control

Obtaining minor numbers of attached instruments:

```
cat /dev/usbtmc0

Minor Number Manufacturer Product Serial Number
001 Agilent Technologies 34980A Switch Measure Unit MY44003719
```

Sending command:

```
echo *RST>/dev/usbtmc1
```

Sending querry:

```
echo *IDN?>/dev/usbtmc1
cat /dev/usbtmc1

Agilent Technologies,34980A,MY44003719,2.19-2.19-2.07-1.05
```

USBTMC
35

Using the USBTMC Driver

Instrument control using file IO system calls

```
#include <stdio.h>
#include <fcntl.h>
main() {
       int myfile;
       char buffer[4000];
       int actual;
       myfile = open("/dev/usbtmc1",O_RDWR);
       if(myfile>0) {
              write(myfile,"*IDN?\n",6);
              actual = read(myfile,buffer,4000);
              buffer[actual] = 0;
              printf("Response:\n%s\n",buffer);
              close(myfile);
       }
}
```