A Linux device driver for EIB-TP-UART-IC and kernel 2.6

(Version 0.03a0)

Changelog

Reinhold Buchinger

Matrikelnr: 0125124

Oct. 2004

1 Added functionality

1.1 TPUART state

You can query the state of the TPUART with the `ioctl` command TPUART_GET_STATE. The driver sends a state request message to the TPUART and sets the QUERY_STATE flag in `tpuart_t.flags`. After that the process sleeps on the reading waitqueue until the QUERY_STATE flag is cleared again. This is done within the interrupt handler when a TPUART state indication message is received. It stores the message in `tpuart_state` (also part of the `tpuart_t` structure) and wakes up the sleeping task which returns the state message to the user. This call will always block for a short time until the response message has arrived! You can find more information about the returned Control Field in the README document, Section 5.2.

1.2 Calculating checksum (transmission)

The last octet of an EIB frame is the check octet which is calculated as an odd parity checksum over the set of corresponding bits belonging to the preceeding octets of the frame. In the V0.02 of the driver the check octet was part of the telegram which the user program passed to the driver. Now the user program has to pass the frame without the check octet. The length of the telegram, which you specify in the length field of the message structure, is always the length of the passed data in bytes. Internally this length value is incremented at the calculation to stay consistent with the code of V0.02. The driver calculates the check octet in transmit.c.

1.3 Heartbeat to avoid infinite blocking

The driver in version 0.02 could block infinite in `read/write` without noticing a dead TPUART. To avoid this this version implements a heartbeat. The fundamental principle is described in README, Section 5.4.
The heartbeat never disrupts an ongoing transmission. It always waits with its state message until the frame has been completely transferred to the TPUART. The heartbeat cannot delay an U_AckInformation-Service because the acknowledgment is immediately sent in the ISR.
The heartbeat can be highly delayed by the ISR under extreme heavy load. But this indicates too that the TPUART is still alive and doesn't affect the functionality of the heartbeat.

1.4 Read/Write return values

All return values are summarized in README, Section 5.5. In comparison with version 0.02 the return values EREMOTEIO and ECOMM were added. The driver returns EREMOTEIO whenever the heartbeat has failed. The `write` function sets ECOMM if a negative L_Data.confirm was received.
Additionally the I/O command TPUART_GET_WRITE_STATUS was introduced to receive write error values in the non-blocking case. You get more information about this command in README, Section 5.3.

1.5 Detection of a frame end

In Version 0.2 the detection of a frame end relied on the length information of the received frame. There was a method for detecting a timeout but it wasn't used in valuable way. This version didn't change the processing of the length information but it is now possible to disable this end detection and try to detect the end of a frame via timeout. This should be seen as experimental add-on. You get more information about the problem of detecting a frame end in README, Section 6 and Source Documentation, Section 1.2.

2 Porting the driver to kernel version 2.6

2.1 Task queue replaced by workqueue mechanism

Task queues provide a flexible utility for scheduling execution at a later time without resorting to interrupts. They are used to schedule the bottom halves of transmission and receiving.
The task queue interface was removed in 2.6 kernels and in its place is a new workqueue mechanism. Tasks to be run out of a workqueue need to be packaged in a `struct work_struct` structure. Therefore the type of the variable worker in the `xmit_t` struct changed from `tq_struct` to `work_struct`.
To set up a `work_struct` structure at run time the following macro was used:

```
INIT_WORK(struct work_struct *work,
                  void (*function)(void *), void *data);
```

The default queue provided by the kernel was used for the driver. `Work_struct` structures can be added to this queue with:

```
int schedule_work(struct work_struct *work);
```

2.2 Use spinlocks to protect critical sections

In the 2.6 kernel, it is no longer possible to globally disable interrupts. In particular, the `cli()`, `sti()`, `save_flags()`, and `restore_flags()` functions are no longer available. In version 0.02 the output to the serial port and the updating of the addresses was protected by disabling interrupts. Disabling interrupts globally is not only no longer possible it also doesn't protect critical sections because in 2.6 a preemptive kernel was introduced. This means that kernel code can be interrupted at (almost) any time.
There is one big, important exception: preemption will not happen if the currently-running code is holding a spinlock. In V0.03a0 the protection of critical sections is done by spinlocks. Therefore the variable `spinlock_t lock` was added to `tpuart_t`.
A spinlock works through a shared variable. Any function needing the lock will query it and, seeing that it is not available, will "spin" in a busy-wait loop until it is available. If you use

```
spin_lock_irqsave(spinlock_t *lock, unsigned long flags)
```

not only the lock is acquired but also all interrupts on the local processor are disabled and the current interrupt state is stored in flags.

```
spin_unlock_irqrestore(spinlock_t *lock, unsigned long flags)
```

unlocks the given lock and enables interrupts (depending on the flags value).

2.3 Going to sleep

`Sleep_on()` and `interruptible_sleep_on()` were not removed but they shouldn't be used anymore. Where the old code sleeps because a condition was unfulfilled[1] the sleeping instruction was replaced by a "manual" sleep with the commands

```
prepare_to_wait(&queue, &wait, TASK_INTERRUPTIBLE)
finish_wait(&queue, &wait)
```

This avoids unnecessary sleeps as well. In version 0.02 the driver always slept in `tpuart_write`. To use the recommended

```
int wait_event_interruptible (queue, condition)
```

one needs a condition whether the process should sleep or awake. If the condition is false the process fall asleep. When it is woken up it checks the condition again and gets running only if the condition is true.
The flag WAKE_UP_WRITE was introduced for this purpose. The process sleeps if it is cleared and the flag becomes set before the wake call. Thus we have the same behavior as in the old driver.

2.4 Other changes

- The Makefile was completely rewritten to use the kernel build system for external modules.
- Because of major changes all statements for backward compatibility were removed. This version only works with 2.6 kernels.
- The second argument (the address) of bit operations (`set_bit`, `test_bit`,...) changed from void * to volatile unsigned long *. The variables were adjusted to avoid warnings.
- Include linux/slab.h instead of linux/malloc.h.
- Don't use `MOD_INC_COUNT` and `MOD_DEC_COUNT` anymore.
- `MODULE_PARAM(variable,type)` was replaced by `module_param(name,type, perm)`. The new parameter declaration scheme add type safety and new functionality.
- `MODULE_LICENCE` statement was added to avoid a tainted kernel. Tainting of the kernel is a way of indicating that a proprietary module has been inserted, which is not really true for this driver.
- Struct `tpuart_fops` adapted to ANSI C standard initializer format.
- Added `__exit` to `tpuart_cleanup` to be conform with `tpuart_init`.
- Return type of interrupt handler changed from `void` to `irqreturn_t`. If the interrupt handler recognizes and handles a given interrupt, it should return IRQ_HANDLED. If it knows that the interrupt was not on a device it manages, it can return IRQ_NONE instead. This way a handler can tell the kernel about spurious interrupts.
- The new macro `unsigned int iminor(struct inode *inode)` was used to get the minor number.

1 The process should sleep if `deque_is_empty_irq` returns true in `read` or `deque_is_full_irq` returns true in `write`.

- Bug in version 0.02: changed `TPUART_IOCMAX` in tpuart_main.h from 17 to 21. `TPUART_IOCMAX` is used to check if `ioctl` is called with an valid number.
- Bug in version 0.02: in the ISR, transmit interrupt, the read pointer is also incremented after the last write. Therefore the comparison `dev→tx.read == dev→tx.end` couldn't be true.
- The support for long frames was removed because it becomes obsoleted by extended frame format defined in KNX handbook 1.1 volume 3 supplement 13.

Literature

[RC01] Alessandro Rubini and Jonathan Corbet. Linux Device Drivers, 2nd Edition. O'Reilly, 2001.

[TP01] Siemens. EIB-TP-UART-IC datasheet. 2001.

[Deg] Fachhochschule Deggendorf. EIB-Linux-Treiber - Linux open Source.
http://www.hto.fh-deggendorf.de/komm/building/projekte/linux.html

[EIB99] EIBHandbook. EIB Handbook Series Volume 3.
http://eiba-software.com/eibacom/Volume3.zip 1999

[LWN] Jonathan Corbet. Porting device drivers to the 2.6 kernel.
http://lwn.net/Articles/driver-porting/

[KNB] Kernelnewbies. http://www.kernelnewbies.org/

[SB] Interfacing the Serial/ RS-232 Port. http://www.beyondlogic.org/serial/serial1.htm