

[PATCH] Add a non-blocking I2C interface

Source: <http://linux.derkeiler.com/Mailing-Lists/Kernel/2005-01/8599.html>

From: Corey Minyard (minyard_at_acm.org)

Date: 01/28/05

Date: Fri, 28 Jan 2005 09:08:06 -0600

To: Bukie Mabayoje <bukiemab@gte.net>

Here's the code that I have so far for adding a non-blocking interface to the I2C interface. I've debated whether to do this as a patch or just post the files, because the patch is about half the size of the files. I've decided on the diff for now, it seems to be fairly readable. This is relative to 2.6.11-rc2.

I have not extensively tested this patch. I've done eeprom operations on the piix4 and i801 chips both with and without updating the those interface for non-blocking operation. I'm going to test the block transfers with the IPMI driver when I get access to a system (this weekend). This is posted for comment.

This patch doesn't actually change much. It adds a queue entry that is used to pass around the message data (and is obviously used for queuing :). But this mostly is using the queue entry for data, breaking the operations into smaller functions, and doing the polling in the i2c-core code instead of the drivers themselves (for non-blocking capable drivers). Function for unchanged drivers should be unchanged.

This does not implement SMB Alert, which should improve the performance of the IPMI SMB driver. I'll do that in the future and just poll the interface for now.

I'll post the changes to the i801 driver next.

This patch requires the fixes to the completion code that Mike Waychison posted a few days ago (<http://marc.theaimsgroup.com/?l=linux-kernel&m=110669761400454&w=2>). Otherwise the `wait_for_completion_interruptible()` and `wait_for_completion_timeout()` are broken.

In case you missed the previous discussion, I need a non-blocking interface in the I2C code for use by the IPMI SMB driver. The driver needs to be able to do things like store panic information in the SEL, ping the watchdog timer while in a panic, and power the system off. The current I2C driver requires a task context, but you can't exactly do

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

things at panic time with a task context.

-Corey

Index: linux-2.6.11-rc2/drivers/i2c/i2c-core.c

```
-----  
--- linux-2.6.11-rc2.orig/drivers/i2c/i2c-core.c 2005-01-26 15:59:53.000000000 -0600  
+++ linux-2.6.11-rc2/drivers/i2c/i2c-core.c 2005-01-28 08:54:00.000000000 -0600  
@@ -30,8 +30,14 @@  
#include <linux/init.h>  
#include <linux/idr.h>  
#include <linux/seq_file.h>  
+#include <linux/completion.h>  
#include <asm/uaccess.h>  
  
+static int i2c_stop_timer(struct i2c_adapter * adap);  
+static void i2c_start_timer(struct i2c_adapter * adap,  
+ struct i2c_op_q_entry * entry);  
+  
+#define USEC_PER_JIFFIE (1000000 / HZ)  
  
static LIST_HEAD(adapters);  
static LIST_HEAD(drivers);  
@@ -134,11 +140,26 @@  
    }  
  
    adap->nr = id & MAX_ID_MASK;  
+ spin_lock_init(&adap->q_lock);  
+ INIT_LIST_HEAD(&adap->q);  
    init_MUTEX(&adap->bus_lock);  
    init_MUTEX(&adap->clist_lock);  
    list_add_tail(&adap->list,&adapters);  
    INIT_LIST_HEAD(&adap->clients);  
  
+ adap->timer = kmalloc(sizeof(*adap->timer), GFP_KERNEL);  
+ if (!adap->timer) {  
+ res = -ENOMEM;  
+ goto out_unlock;  
+ }  
+  
+ init_timer(&adap->timer->timer);  
+ spin_lock_init(&adap->timer->lock);  
+ adap->timer->deleted = 0;  
+ adap->timer->running = 0;  
+ adap->timer->next_call_time = 0;  
+ adap->timer->adapter = adap;  
+  
    /* Add the adapter to the driver core.  
    * If the parent pointer is not set up,
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
    * we add this adapter to the host bus.
@@ -181,6 +202,7 @@
    struct i2c_driver *driver;
    struct i2c_client *client;
    int res = 0;
+ unsigned long flags;

    down(&core_lists);

@@ -233,6 +255,17 @@
    device_unregister(&adap->dev);
    list_del(&adap->list);

+ /* Stop the timer and free its memory */
+ spin_lock_irqsave(&adap->timer->lock, flags);
+ if (i2c_stop_timer(adap)) {
+ spin_unlock_irqrestore(&adap->timer->lock, flags);
+ kfree(adap->timer);
+ } else {
+ adap->timer->deleted = 1;
+ spin_unlock_irqrestore(&adap->timer->lock, flags);
+ }
+ adap->timer = NULL;
+
+ /* wait for sysfs to drop all references */
+ wait_for_completion(&adap->dev_released);
+ wait_for_completion(&adap->class_dev_released);
@@ -583,15 +616,283 @@
* -----
*/

-int i2c_transfer(struct i2c_adapter * adap, struct i2c_msg msgs[],int num)
+/* Must be called with the q_lock held. */
+static void i2c_start_entry(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ entry->started = 1;
+ switch (entry->xfer_type) {
+ case I2C_OP_I2C:
+ adap->algo->master_start(adap, entry);
+ break;
+ case I2C_OP_SMBUS:
+ adap->algo->smbus_start(adap, entry);
+ break;
+ default:
+ entry->result = -EINVAL;
+ i2c_op_done(adap, entry);
+ }
+
+ if (!entry->completed && entry->use_timer)
+ i2c_start_timer(adap, entry);
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+}
+
+/* Must be called with q lock held. */
+static void i2c_entry_inc(struct i2c_adapter * adapter,
+ struct i2c_op_q_entry * entry)
+{
+ int ret;
+ atomic_inc(&entry->usecount);
+}
+
+/* Get the first entry off the head of the queue and lock it there.
+ The entry is guaranteed to remain first in the list and the handler
+ not be called until i2c_entry_put() is called. */
+static struct i2c_op_q_entry *_i2c_entry_get(struct i2c_adapter * adap)
+{
+ struct i2c_op_q_entry * entry = NULL;
+
+ if (adap->algo->master_xfer) {
+ dev_dbg(&adap->dev, "master_xfer: with %d msgs.\n", num);
+ if (!list_empty(&adap->q)) {
+ struct list_head * link = adap->q.next;
+ entry = list_entry(link, struct i2c_op_q_entry, link);
+ if (entry->completed)
+ entry = NULL;
+ else
+ i2c_entry_inc(adap, entry);
+ }
+ pr_debug("_i2c_entry_get %p %p\n", adap, entry);
+ return entry;
+}
+
+struct i2c_op_q_entry *_i2c_entry_get(struct i2c_adapter * adap)
+{
+ unsigned long flags;
+ struct i2c_op_q_entry * entry;
+
+ spin_lock_irqsave(&adap->q_lock, flags);
+ entry = _i2c_entry_get(adap);
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+ return entry;
+}
+
+void i2c_entry_put(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ unsigned long flags;
+ struct i2c_op_q_entry * new_entry = NULL;
+
+ restart:
+ pr_debug("i2c_put %p %p\n", adap, entry);
+ if (atomic_dec_and_test(&entry->usecount)) {
```

[PATCH] Add a non-blocking I2C interface

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ spin_lock_irqsave(&adap->q_lock, flags);
+ list_del(&entry->link);
+
+ /* Get the next entry to start. */
+ new_entry = _i2c_entry_get(adap);
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+
+ entry->handler(entry);
+
+
+ if (new_entry) {
+ i2c_start_entry(adap, new_entry);
+ if (new_entry->start)
+ complete(new_entry->start);
+ /* Do tail recursion ourself. */
+ entry = new_entry;
+ goto restart;
+ }
+ }
+ }
+
+static void i2c_handle_timer(unsigned long data);
+
+static void i2c_start_timer(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ unsigned int wait_jiffies;
+ struct i2c_timer *t = adap->timer;
+ unsigned long flags;
+
+ wait_jiffies = ((entry->call_again_us + USEC_PER_JIFFIE - 1)
+ / USEC_PER_JIFFIE);
+ if (wait_jiffies == 0)
+ wait_jiffies = 1;
+ /* This won't be polled from the user code, so
+ start a timer to poll it. */
+ spin_lock_irqsave(&t->lock, flags);
+ if (! t->running) {
+ t->timer.expires = jiffies + wait_jiffies;
+ t->timer.data = (unsigned long) adap;
+ t->timer.function = i2c_handle_timer;
+ t->running = 1;
+ t->next_call_time = wait_jiffies * USEC_PER_JIFFIE;
+ add_timer(&t->timer);
+ t->sequence = adap->timer_sequence;
+ }
+ spin_unlock_irqrestore(&t->lock, flags);
+ }
+
+ /* Returns true if the timer is stopped (or was not running), false if
+ not. Must be called with the timer lock held. */
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+static int i2c_stop_timer(struct i2c_adapter * adap)
+{
+ return (!adap->timer->running || del_timer(&adap->timer->timer));
+}
+
+static void i2c_handle_timer(unsigned long data)
+{
+ struct i2c_timer * t = (void *) data;
+ struct i2c_adapter * adap;
+ unsigned long flags;
+ struct i2c_op_q_entry * entry;
+ unsigned int sequence_match;
+
+ spin_lock_irqsave(&t->lock, flags);
+ if (t->deleted) {
+ spin_unlock_irqrestore(&t->lock, flags);
+ kfree(t);
+ return;
+ }
+
+ adap = t->adapter;
+ t->running = 0;
+ sequence_match = adap->timer_sequence == t->sequence;
+ spin_unlock_irqrestore(&t->lock, flags);
+
+ entry = i2c_entry_get(adap);
+ pr_debug("i2c_handle_timer: %p %p\n", adap, entry);
+ if (!entry)
+ return;
+
+ if (sequence_match) {
+ /* This is the one we expected, call the poll routine. */
+ adap->algo->poll(adap, entry, t->next_call_time);
+
+ if (!entry->completed)
+ i2c_start_timer(adap, entry);
+ } else if (entry->use_timer)
+ /* We raced in timer deletion, just restart the
+ timer if necessary. */
+ i2c_start_timer(adap, entry);
+
+ i2c_entry_put(adap, entry);
+}
+
+void i2c_op_done(struct i2c_adapter *adap, struct i2c_op_q_entry *e)
+{
+ unsigned long flags;
+ int did_complete = 0;
+
+ pr_debug("i2c_op_done: %p %p\n", adap, e);
+ spin_lock_irqsave(&adap->q_lock, flags);
```

[PATCH] Add a non-blocking I2C interface

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ if (! e->completed) {
+ e->completed = 1;
+ did_complete = 1;
+ }
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+
+ if (did_complete) {
+ if (e->use_timer) {
+ struct i2c_timer *t = adap->timer;
+ spin_lock_irqsave(&t->lock, flags);
+ if (!i2c_stop_timer(adap))
+ /* If we are unable to stop the timer, that
+ means the timer has gone off but has not
+ yet run the first part of the handler call.
+ Increment the sequence so the timer handler
+ can detect this. */
+ adap->timer_sequence++;
+ spin_unlock_irqrestore(&t->lock, flags);
+ }
+ if (e->complete)
+ e->complete(adap, e);
+ }
+
+ i2c_entry_put(adap, e);
+}
+
+static void i2c_wait_complete(struct i2c_op_q_entry * entry)
+{
+ struct completion *done = entry->handler_data;
+ pr_debug("i2c_wait_complete %p\n", entry);
+ complete(done);
+}
+
+static void i2c_perform_op_wait(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ struct completion done;
+ unsigned long flags;
+ struct i2c_algorithm *algo = adap->algo;
+
+ pr_debug("i2c_perform_op_wait %p %p\n", adap, entry);
+ init_completion(&done);
+ entry->start = NULL;
+ entry->handler = i2c_wait_complete;
+ entry->handler_data = &done;
+ entry->started = 0;
+ entry->completed = 0;
+ entry->result = 0;
+ entry->use_timer = 0; /* We poll it directly. */
+ entry->data = NULL;
+ atomic_set(&entry->usecount, 1);
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ spin_lock_irqsave(&adap->q_lock, flags);
+ list_add_tail(&entry->link, &adap->q);
+ if (adap->q.next == &entry->link) {
+ /* Added to the list head, start it */
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+ i2c_start_entry(adap, entry);
+ } else {
+ struct completion start;
+ init_completion(&start);
+ entry->start = &start;
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+
+ wait_for_completion_interruptible(&start);
+
+ spin_lock_irqsave(&adap->q_lock, flags);
+ if (!entry->started) {
+ /* Operation was interrupted. There
+ is a race, we can't use the
+ wait_for_completion return code. */
+ entry->result = -ERESTARTSYS;
+ entry->completed = 1;
+ list_del(&entry->link);
+ }
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+ }
+
+ /* Once the operation is started, we will not
+ interrupt it. */
+ while (!entry->completed) {
+ unsigned int timeout = entry->call_again_us;
+ timeout += (USEC_PER_JIFFIE - 1);
+ timeout /= USEC_PER_JIFFIE;
+ if (timeout == 0)
+ timeout = 1;
+ wait_for_completion_timeout(&done, timeout);
+ if (entry->completed)
+ break;
+ algo->poll(adap, entry, timeout * USEC_PER_JIFFIE);
+ }
+ }
+
+static int i2c_transfer_entry(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ entry->xfer_type = I2C_OP_I2C;
+ entry->complete = NULL;
+ if (adap->algo->master_start) {
+ i2c_perform_op_wait(adap, entry);
+ return entry->result;
+ } else if (adap->algo->master_xfer) {
+ int ret;
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ dev_dbg(&adap->dev, "master_xfer: with %d msgs.\n",
+ entry->i2c.num);

        down(&adap->bus_lock);
- ret = adap->algo->master_xfer(adap,msgs,num);
+ ret = adap->algo->master_xfer(adap, entry->i2c.msgs,
+ entry->i2c.num);
        up(&adap->bus_lock);

        return ret;
@@ -601,33 +902,45 @@
    }
}

+int i2c_transfer(struct i2c_adapter * adap, struct i2c_msg msgs[],int num)
+{
+ struct i2c_op_q_entry *entry;
+ int rv;
+
+ entry = kmalloc(sizeof(*entry), GFP_KERNEL);
+ if (!entry)
+ return -ENOMEM;
+
+ entry->i2c.msgs = msgs;
+ entry->i2c.num = num;
+
+ rv = i2c_transfer_entry(adap, entry);
+ kfree(entry);
+ return rv;
+}
+
int i2c_master_send(struct i2c_client *client,const char *buf ,int count)
{
    int ret;
    struct i2c_adapter *adap=client->adapter;
    struct i2c_msg msg;

- if (client->adapter->algo->master_xfer) {
- msg.addr = client->addr;
- msg.flags = client->flags & I2C_M_TEN;
- msg.len = count;
- msg.buf = (char *)buf;
-
- dev_dbg(&client->adapter->dev, "master_send: writing %d bytes.\n",
- count);
+ msg.addr = client->addr;
+ msg.flags = client->flags & I2C_M_TEN;
+ msg.len = count;
+ msg.buf = (char *)buf;

- down(&adap->bus_lock);
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
- ret = adap->algo->master_xfer(adap,&msg,1);
- up(&adap->bus_lock);
+ dev_dbg(&client->adapter->dev, "master_send: writing %d bytes.\n",
+ count);

- /* if everything went ok (i.e. 1 msg transmitted), return #bytes
- * transmitted, else error code.
- */
- return (ret == 1)? count : ret;
- } else {
- dev_err(&client->adapter->dev, "I2C level transfers not supported\n");
- return -ENOSYS;
- }
+ ret = i2c_transfer(adap, &msg, 1);
+ if (ret < 0)
+ return ret;
+
+ /* if everything went ok (i.e. 1 msg transmitted), return #bytes
+ * transmitted, else error code.
+ */
+ return (ret == 1)? count : ret;
}

int i2c_master_recv(struct i2c_client *client, char *buf ,int count)
@@ -635,31 +948,27 @@
    struct i2c_adapter *adap=client->adapter;
    struct i2c_msg msg;
    int ret;
- if (client->adapter->algo->master_xfer) {
- msg.addr = client->addr;
- msg.flags = client->flags & I2C_M_TEN;
- msg.flags |= I2C_M_RD;
- msg.len = count;
- msg.buf = buf;

- dev_dbg(&client->adapter->dev, "master_recv: reading %d bytes.\n",
- count);
-
- down(&adap->bus_lock);
- ret = adap->algo->master_xfer(adap,&msg,1);
- up(&adap->bus_lock);
+ msg.addr = client->addr;
+ msg.flags = client->flags & I2C_M_TEN;
+ msg.flags |= I2C_M_RD;
+ msg.len = count;
+ msg.buf = buf;
+
+ dev_dbg(&client->adapter->dev, "master_recv: reading %d bytes.\n",
+ count);

- dev_dbg(&client->adapter->dev, "master_recv: return:%d (count:%d, addr:0x%02x)\n",
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
- ret, count, client->addr);
+ ret = i2c_transfer(adap, &msg, 1);
+ if (ret < 0)
+ return ret;
+
+ dev_dbg(&client->adapter->dev, "master_recv: return:%d (count:%d, addr:0x%02x)\n",
+ ret, count, client->addr);

- /* if everything went ok (i.e. 1 msg transmitted), return #bytes
- * transmitted, else error code.
- */
- return (ret == 1)? count : ret;
- } else {
- dev_err(&client->adapter->dev, "I2C level transfers not supported\n");
- return -ENOSYS;
- }
+ /* if everything went ok (i.e. 1 msg transmitted), return #bytes
+ * transmitted, else error code.
+ */
+ return (ret == 1)? count : ret;
}

@@ -1037,7 +1346,8 @@
}

/* Returns the number of read bytes */
-s32 i2c_smbus_read_i2c_block_data(struct i2c_client *client, u8 command, u8 *values)
+s32 i2c_smbus_read_i2c_block_data(struct i2c_client *client, u8 command,
+ u8 *values)
{
    union i2c_smbus_data data;
    int i;
@@ -1052,184 +1362,333 @@
}

-/* Simulate a SMBus command using the i2c protocol
- No checking of parameters is done! */
-static s32 i2c_smbus_xfer_emulated(struct i2c_adapter * adapter, u16 addr,
- unsigned short flags,
- char read_write, u8 command, int size,
- union i2c_smbus_data * data)
-{
- /* So we need to generate a series of msgs. In the case of writing, we
- need to use only one message; when reading, we need two. We initialize
- most things with sane defaults, to keep the code below somewhat
- simpler. */
- unsigned char msgbuf0[34];
- unsigned char msgbuf1[34];
- int num = read_write == I2C_SMBUS_READ?2:1;
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
- struct i2c_msg msg[2] = { { addr, flags, 1, msgbuf0 },
- { addr, flags | I2C_M_RD, 0, msgbuf1 }
- };
+
+static void i2c_smbus_complete_entry(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ if (entry->result < 0)
+ return;
+
+ if(entry->result >= 0 && entry->swpec &&
+ entry->smbus.size != I2C_SMBUS_QUICK &&
+ entry->smbus.size != I2C_SMBUS_I2C_BLOCK_DATA &&
+ (entry->smbus.read_write == I2C_SMBUS_READ ||
+ entry->smbus.size == I2C_SMBUS_PROC_CALL_PEC ||
+ entry->smbus.size == I2C_SMBUS_BLOCK_PROC_CALL_PEC)) {
+ if(i2c_smbus_check_pec(entry->smbus.addr,
+ entry->smbus.command,
+ entry->smbus.size,
+ entry->partial,
+ entry->smbus.data))
+ entry->result = -EINVAL;
+ }
+ }
+
+static void i2c_smbus_format_entry(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+{
+ entry->swpec = 0;
+ entry->partial = 0;
+ entry->smbus.flags &= I2C_M_TEN | I2C_CLIENT_PEC;
+ if((entry->smbus.flags & I2C_CLIENT_PEC) &&
+ !(i2c_check_functionality(adap, I2C_FUNC_SMBUS_HWPEC_CALC))) {
+ entry->swpec = 1;
+ if(entry->smbus.read_write == I2C_SMBUS_READ &&
+ entry->smbus.size == I2C_SMBUS_BLOCK_DATA)
+ entry->smbus.size = I2C_SMBUS_BLOCK_DATA_PEC;
+ else if(entry->smbus.size == I2C_SMBUS_PROC_CALL)
+ entry->smbus.size = I2C_SMBUS_PROC_CALL_PEC;
+ else if(entry->smbus.size == I2C_SMBUS_BLOCK_PROC_CALL) {
+ unsigned char *data = entry->smbus.data->block;
+ i2c_smbus_add_pec(entry->smbus.addr,
+ entry->smbus.command,
+ I2C_SMBUS_BLOCK_DATA,
+ entry->smbus.data);
+ entry->partial = data[data[0] + 1];
+ entry->smbus.size = I2C_SMBUS_BLOCK_PROC_CALL_PEC;
+ } else if(entry->smbus.read_write == I2C_SMBUS_WRITE &&
+ entry->smbus.size != I2C_SMBUS_QUICK &&
+ entry->smbus.size != I2C_SMBUS_I2C_BLOCK_DATA)
+ entry->smbus.size =
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ i2c_smbus_add_pec(entry->smbus.addr,  
+ entry->smbus.command,  
+ entry->smbus.size,  
+ entry->smbus.data);  
+ }  
+  
+ entry->complete = i2c_smbus_complete_entry;  
+}  
+  
+static void i2c_smbus_emu_complete(struct i2c_adapter * adap,  
+ struct i2c_op_q_entry * entry)  
+{  
+ unsigned char *msgbuf0 = entry->i2c.msgs[0].buf;  
+ unsigned char *msgbuf1 = entry->i2c.msgs[1].buf;  
+     int i;  
  
- msgbuf0[0] = command;  
- switch(size) {  
+ if (entry->smbus.read_write != I2C_SMBUS_READ)  
+ return;  
+  
+ switch(entry->smbus.size) {  
+ case I2C_SMBUS_BYTE:  
+ entry->smbus.data->byte = msgbuf0[0];  
+ break;  
+ case I2C_SMBUS_BYTE_DATA:  
+ entry->smbus.data->byte = msgbuf1[0];  
+ break;  
+ case I2C_SMBUS_WORD_DATA:  
+ case I2C_SMBUS_PROC_CALL:  
+ entry->smbus.data->word = msgbuf1[0]|(msgbuf1[1] << 8);  
+ break;  
+ case I2C_SMBUS_I2C_BLOCK_DATA:  
+ /* fixed at 32 for now */  
+ entry->smbus.data->block[0] = I2C_SMBUS_I2C_BLOCK_MAX;  
+ for (i = 0; i < I2C_SMBUS_I2C_BLOCK_MAX; i++)  
+ entry->smbus.data->block[i+1] = msgbuf1[i];  
+ break;  
+ }  
+  
+ entry->xfer_type = I2C_OP_SMBUS;  
+ i2c_smbus_complete_entry(adap, entry);  
+}  
+  
+static int i2c_smbus_emu_format(struct i2c_adapter *adap,  
+ struct i2c_op_q_entry * entry)  
+{  
+ /* So we need to generate a series of msgs. In the case of  
+ writing, we need to use only one message; when reading, we  
+ need two. We initialize most things with sane defaults, to  
+ keep the code below somewhat simpler. */
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```

+ unsigned char *msgbuf0 = entry->msgbuf0;
+ unsigned char *msgbuf1 = entry->msgbuf1;
+ int num = entry->smbus.read_write == I2C_SMBUS_READ?2:1;
+ struct i2c_msg *msg = entry->msg;
+ int i;
+
+ entry->i2c.msgs = msg;
+ entry->i2c.msgs[0].buf = msgbuf0;
+ entry->i2c.msgs[1].buf = msgbuf1;
+
+ msg[0].addr = entry->smbus.addr;
+ msg[0].flags = entry->smbus.flags;
+ msg[0].len = 1;
+ msg[1].addr = entry->smbus.addr;
+ msg[1].flags = entry->smbus.flags | I2C_M_RD;
+ msg[1].len = 1;
+
+ msgbuf0[0] = entry->smbus.command;
+ switch(entry->smbus.size) {
+     case I2C_SMBUS_QUICK:
+         msg[0].len = 0;
+         /* Special case: The read/write field is used as data */
+         - msg[0].flags = flags | (read_write==I2C_SMBUS_READ)?I2C_M_RD:0;
+         + msg[0].flags = (entry->smbus.flags |
+         + ((entry->smbus.read_write==I2C_SMBUS_READ)
+         + ? I2C_M_RD : 0));
+         num = 1;
+         break;
+     case I2C_SMBUS_BYTE:
+         - if (read_write == I2C_SMBUS_READ) {
+         + if (entry->smbus.read_write == I2C_SMBUS_READ) {
+             /* Special case: only a read! */
+         - msg[0].flags = I2C_M_RD | flags;
+         + msg[0].flags = I2C_M_RD | entry->smbus.flags;
+             num = 1;
+         }
+         break;
+     case I2C_SMBUS_BYTE_DATA:
+         - if (read_write == I2C_SMBUS_READ)
+         + if (entry->smbus.read_write == I2C_SMBUS_READ)
+             msg[1].len = 1;
+         else {
+             msg[0].len = 2;
+         - msgbuf0[1] = data->byte;
+         + msgbuf0[1] = entry->smbus.data->byte;
+         }
+         break;
+     case I2C_SMBUS_WORD_DATA:
+         - if (read_write == I2C_SMBUS_READ)
+         + if (entry->smbus.read_write == I2C_SMBUS_READ)
+             msg[1].len = 2;

```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```

        else {
            msg[0].len=3;
- msgbuf0[1] = data->word & 0xff;
- msgbuf0[2] = (data->word >> 8) & 0xff;
+ msgbuf0[1] = entry->smbus.data->word & 0xff;
+ msgbuf0[2] = (entry->smbus.data->word >> 8) & 0xff;
        }
        break;
    case I2C_SMBUS_PROC_CALL:
        num = 2; /* Special case */
- read_write = I2C_SMBUS_READ;
+ entry->smbus.read_write = I2C_SMBUS_READ;
        msg[0].len = 3;
        msg[1].len = 2;
- msgbuf0[1] = data->word & 0xff;
- msgbuf0[2] = (data->word >> 8) & 0xff;
+ msgbuf0[1] = entry->smbus.data->word & 0xff;
+ msgbuf0[2] = (entry->smbus.data->word >> 8) & 0xff;
        break;
    case I2C_SMBUS_BLOCK_DATA:
    case I2C_SMBUS_BLOCK_DATA_PEC:
- if (read_write == I2C_SMBUS_READ) {
- dev_err(&adapter->dev, "Block read not supported "
+ if (entry->smbus.read_write == I2C_SMBUS_READ) {
+ dev_err(&adap->dev, "Block read not supported "
            "under I2C emulation!\n");
            return -1;
        } else {
- msg[0].len = data->block[0] + 2;
+ msg[0].len = entry->smbus.data->block[0] + 2;
            if (msg[0].len > I2C_SMBUS_BLOCK_MAX + 2) {
- dev_err(&adapter->dev, "smbus_access called with "
+ dev_err(&adap->dev,
+ "smbus_access called with "
                "invalid block write size (%d)\n",
- data->block[0]);
+ entry->smbus.data->block[0]);
            return -1;
        }
- if(size == I2C_SMBUS_BLOCK_DATA_PEC)
+ if(entry->smbus.size == I2C_SMBUS_BLOCK_DATA_PEC)
            (msg[0].len)++;
            for (i = 1; i <= msg[0].len; i++)
- msgbuf0[i] = data->block[i-1];
+ msgbuf0[i] = entry->smbus.data->block[i-1];
        }
        break;
    case I2C_SMBUS_BLOCK_PROC_CALL:
    case I2C_SMBUS_BLOCK_PROC_CALL_PEC:
- dev_dbg(&adapter->dev, "Block process call not supported "
+ dev_dbg(&adap->dev, "Block process call not supported "

```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
        "under I2C emulation!\n");
    return -1;
    case I2C_SMBUS_I2C_BLOCK_DATA:
- if (read_write == I2C_SMBUS_READ) {
+ if (entry->smbus.read_write == I2C_SMBUS_READ) {
        msg[1].len = I2C_SMBUS_I2C_BLOCK_MAX;
    } else {
- msg[0].len = data->block[0] + 1;
+ msg[0].len = entry->smbus.data->block[0] + 1;
        if (msg[0].len > I2C_SMBUS_I2C_BLOCK_MAX + 1) {
- dev_err(&adapter->dev, "i2c_smbus_xfer_emulated called with "
+ dev_err(&adap->dev,
+ "i2c_smbus_xfer_emulated called with "
            "invalid block write size (%d)\n",
- data->block[0]);
+ entry->smbus.data->block[0]);
        return -1;
    }
- for (i = 1; i <= data->block[0]; i++)
- msgbuf0[i] = data->block[i];
+ for (i = 1; i <= entry->smbus.data->block[0]; i++)
+ msgbuf0[i] = entry->smbus.data->block[i];
    }
    break;
    default:
- dev_err(&adapter->dev, "smbus_access called with invalid size (%d)\n",
- size);
+ dev_err(&adap->dev,
+ "smbus_access called with invalid size (%d)\n",
+ entry->smbus.size);
    return -1;
}

- if (i2c_transfer(adapter, msg, num) < 0)
- return -1;
-
- if (read_write == I2C_SMBUS_READ)
- switch(size) {
- case I2C_SMBUS_BYTE:
- data->byte = msgbuf0[0];
- break;
- case I2C_SMBUS_BYTE_DATA:
- data->byte = msgbuf1[0];
- break;
- case I2C_SMBUS_WORD_DATA:
- case I2C_SMBUS_PROC_CALL:
- data->word = msgbuf1[0] | (msgbuf1[1] << 8);
- break;
- case I2C_SMBUS_I2C_BLOCK_DATA:
- /* fixed at 32 for now */
- data->block[0] = I2C_SMBUS_I2C_BLOCK_MAX;
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
- for (i = 0; i < I2C_SMBUS_I2C_BLOCK_MAX; i++)
- data->block[i+1] = msgbuf1[i];
- break;
- }
+ entry->xfer_type = I2C_OP_I2C;
+ entry->i2c.msgs = msg;
+ entry->i2c.num = num;
+ entry->complete = i2c_smbus_emu_complete;
    return 0;
}

+/* Simulate a SMBus command using the i2c protocol
+ No checking of parameters is done! */
+static s32 i2c_smbus_xfer_emulated(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry)
+
+{
+ if (i2c_smbus_emu_format(adap, entry))
+ return -EINVAL;
+
+ if (i2c_transfer_entry(adap, entry) < 0)
+ return -EINVAL;
+
-s32 i2c_smbus_xfer(struct i2c_adapter * adapter, u16 addr, unsigned short flags,
+ return entry->result;
+}
+
+s32 i2c_smbus_xfer(struct i2c_adapter * adap, u16 addr, unsigned short flags,
    char read_write, u8 command, int size,
    union i2c_smbus_data * data)
{
- s32 res;
- int swpec = 0;
- u8 partial = 0;
-
- flags &= I2C_M_TEN | I2C_CLIENT_PEC;
- if((flags & I2C_CLIENT_PEC) &&
- !(i2c_check_functionality(adapter, I2C_FUNC_SMBUS_HWPEC_CALC))) {
- swpec = 1;
- if(read_write == I2C_SMBUS_READ &&
- size == I2C_SMBUS_BLOCK_DATA)
- size = I2C_SMBUS_BLOCK_DATA_PEC;
- else if(size == I2C_SMBUS_PROC_CALL)
- size = I2C_SMBUS_PROC_CALL_PEC;
- else if(size == I2C_SMBUS_BLOCK_PROC_CALL) {
- i2c_smbus_add_pec(addr, command,
- I2C_SMBUS_BLOCK_DATA, data);
- partial = data->block[data->block[0] + 1];
- size = I2C_SMBUS_BLOCK_PROC_CALL_PEC;
- } else if(read_write == I2C_SMBUS_WRITE &&
- size != I2C_SMBUS_QUICK &&
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
- size != I2C_SMBUS_I2C_BLOCK_DATA)
- size = i2c_smbus_add_pec(addr, command, size, data);
- }
-
- if (adapter->algo->smbus_xfer) {
- down(&adapter->bus_lock);
- res = adapter->algo->smbus_xfer(adapter,addr,flags,read_write,
- command,size,data);
- up(&adapter->bus_lock);
- } else
- res = i2c_smbus_xfer_emulated(adapter,addr,flags,read_write,
- command,size,data);
-
- if(res >= 0 && swpec &&
- size != I2C_SMBUS_QUICK && size != I2C_SMBUS_I2C_BLOCK_DATA &&
- (read_write == I2C_SMBUS_READ || size == I2C_SMBUS_PROC_CALL_PEC ||
- size == I2C_SMBUS_BLOCK_PROC_CALL_PEC)) {
- if(i2c_smbus_check_pec(addr, command, size, partial, data))
- return -1;
+ struct i2c_op_q_entry *entry;
+ struct i2c_algorithm *algo = adap->algo;
+ int result;
+
+
+ entry = kmalloc(sizeof(*entry), GFP_KERNEL);
+ if (!entry)
+ return -ENOMEM;
+
+ entry->xfer_type = I2C_OP_SMBUS;
+ entry->smbus.addr = addr;
+ entry->smbus.flags = flags;
+ entry->smbus.read_write = read_write;
+ entry->smbus.command = command;
+ entry->smbus.size = size;
+ entry->smbus.data = data;
+
+ i2c_smbus_format_entry(adap, entry);
+
+ if (algo->smbus_start) {
+ i2c_perform_op_wait(adap, entry);
+ } else if (algo->smbus_xfer) {
+ down(&adap->bus_lock);
+ entry->result = adap->algo->smbus_xfer(adap,
+ entry->smbus.addr,
+ entry->smbus.flags,
+ entry->smbus.read_write,
+ entry->smbus.command,
+ entry->smbus.size,
+ entry->smbus.data);
+ up(&adap->bus_lock);
+ } else {
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ i2c_smbus_xfer_emulated(adap, entry);
    }
- return res;
+
+ result = entry->result;
+ kfree(entry);
+ return result;
}

+int i2c_non_blocking_capable(struct i2c_adapter *adap)
+{
+ return adap->algo->poll != NULL;
+}
+
+void i2c_poll(struct i2c_client *client,
+ unsigned int us_since_last_call)
+{
+ struct i2c_adapter *adap = client->adapter;
+ struct i2c_op_q_entry *entry;
+
+ entry = i2c_entry_get(adap);
+ if (!entry)
+ return;
+ adap->algo->poll(adap, entry, us_since_last_call);
+ i2c_entry_put(adap, entry);
+}
+
+int i2c_non_blocking_op(struct i2c_client *client,
+ struct i2c_op_q_entry *entry)
+{
+ unsigned long flags;
+ struct i2c_adapter *adap = client->adapter;
+
+ if (!i2c_non_blocking_capable(adap))
+ return -ENOSYS;
+
+ entry->smbus.addr = client->addr;
+ entry->smbus.flags = client->flags;
+
+ if (entry->xfer_type == I2C_OP_SMBUS) {
+ i2c_smbus_format_entry(adap, entry);
+ if (!adap->algo->smbus_start) {
+ if (i2c_smbus_emu_format(adap, entry))
+ return -EINVAL;
+ }
+ }
+
+ entry->start = NULL;
+ entry->started = 0;
+ entry->completed = 0;
+ entry->result = 0;
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ entry->use_timer = 1; /* Let the timer code poll it. */
+ entry->data = NULL;
+ atomic_set(&entry->usecount, 1);
+
+ spin_lock_irqsave(&adap->q_lock, flags);
+ list_add_tail(&entry->link, &adap->q);
+ if (adap->q.next == &entry->link) {
+ /* Added to the list head, start it */
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+ i2c_start_entry(adap, entry);
+ } else
+ spin_unlock_irqrestore(&adap->q_lock, flags);
+ return 0;
+}

/* You should always define `functionality'; the 'else' is just for
   backward compatibility. */
@@ -1258,6 +1717,7 @@
EXPORT_SYMBOL(i2c_clients_command);
EXPORT_SYMBOL(i2c_check_addr);

+EXPORT_SYMBOL(i2c_op_done);
EXPORT_SYMBOL(i2c_master_send);
EXPORT_SYMBOL(i2c_master_recv);
EXPORT_SYMBOL(i2c_control);
@@ -1278,6 +1738,10 @@
EXPORT_SYMBOL(i2c_smbus_write_block_data);
EXPORT_SYMBOL(i2c_smbus_read_i2c_block_data);

+EXPORT_SYMBOL(i2c_non_blocking_capable);
+EXPORT_SYMBOL(i2c_poll);
+EXPORT_SYMBOL(i2c_non_blocking_op);
+
EXPORT_SYMBOL(i2c_get_functionality);
EXPORT_SYMBOL(i2c_check_functionality);

Index: linux-2.6.11-rc2/include/linux/i2c.h
=====
--- linux-2.6.11-rc2.orig/include/linux/i2c.h 2005-01-26 15:59:53.000000000 -0600
+++ linux-2.6.11-rc2/include/linux/i2c.h 2005-01-28 08:53:31.000000000 -0600
@@ -32,7 +32,11 @@
#include <linux/types.h>
#include <linux/i2c-id.h>
#include <linux/device.h> /* for struct device */
+#include <linux/list.h>
+#include <linux/spinlock.h>
+#include <linux/timer.h>
#include <asm/semaphore.h>
+#include <asm/atomic.h>

/* --- General options ----- */
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
@@ -43,6 +47,7 @@
struct i2c_driver;
struct i2c_client_address_data;
union i2c_smbus_data;
+struct i2c_op_q_entry;

/*
 * The master routines are the ones normally used to transmit data to devices
@@ -55,7 +60,7 @@

/* Transfer num messages.
 */
-extern int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg msg[],int num);
+extern int i2c_transfer(struct i2c_adapter *adap, struct i2c_msg *msg,int num);

/*
 * Some adapter types (i.e. PCF 8584 based ones) may support slave behaviuor.
@@ -95,6 +100,25 @@
extern s32 i2c_smbus_read_i2c_block_data(struct i2c_client * client,
                                       u8 command, u8 *values);

+/* Non-blocking interface. The user should fill out the public
+ portions of the entry structure. All data in the entry structure
+ should be guaranteed to be available until the handler callback is
+ called with the entry. */
+typedef void (*i2c_op_done_cb)(struct i2c_op_q_entry *entry);
+
+extern int i2c_non_blocking_op(struct i2c_client *client,
+ struct i2c_op_q_entry *entry);
+
+/* Can the adapter do non-blocking operations? */
+extern int i2c_non_blocking_capable(struct i2c_adapter *adap);
+
+/* Poll the i2c interface. This should only be called in a situation
+ where scheduling and interrupts are off. You should put the amount
+ of microseconds between calls in us_since_last_call. */
+extern void i2c_poll(struct i2c_client *client,
+ unsigned int us_since_last_call);
+
+
+/*
 * A driver is capable of handling one or more physical devices present on
 * I2C adapters. This information is used to inform the driver of adapter
@@ -181,6 +205,33 @@
}

/*
+ * About locking and the non-blocking interface.
+ *
+ * The poll operations are called single-threaded (along with the
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ * xxx_start operations), so if the driver is only polled then there
+ * is no need to do any locking. If you are using interrupts, then
+ * the timer operations and interrupts can race and you need to lock
+ * appropriately.
+ *
+ * i2c_op_done() can be called multiple times on the same entry (as
+ * long as each one has a get operation). This handles poll and
+ * interrupt races calling i2c_op_done(). It will do the right thing.
+ */
+
+/* Called from a non-blocking interface to get the current working
+ entry. Returns NULL if there is none. This is primarily for
+ interrupt handlers to determine what they should be working on.
+ Note that if you call i2c_entry_get() and get a non-null entry, you
+ must call i2c_entry_put() on it. */
+struct i2c_op_q_entry *i2c_entry_get(struct i2c_adapter * adap);
+void i2c_entry_put(struct i2c_adapter * adap,
+ struct i2c_op_q_entry * entry);
+
+/* Called from a non-blocking interface to report that an operation
+ has completed. Can be called from interrupt context. */
+void i2c_op_done(struct i2c_adapter *adap, struct i2c_op_q_entry *entry);
+
+/*
+ * The following structs are for those who like to implement new bus drivers:
+ * i2c_algorithm is the interface to a class of hardware solutions which can
+ * be addressed using the same bus algorithms – i.e. bit-banging or the PCF8584
+ @@ -190,15 +241,40 @@
+     char name[32]; /* textual description */
+     unsigned int id;
+
+ - /* If an adapter algorithm can't do I2C-level access, set master_xfer
+ + /* If an adapter algorithm can't do I2C-level access, set master_xfer
+     to NULL. If an adapter algorithm can do SMBus access, set
+     smbus_xfer. If set to NULL, the SMBus protocol is simulated
+     using common I2C messages */
+ - int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg msgs[],
+ - int num);
+ - int (*smbus_xfer) (struct i2c_adapter *adap, u16 addr,
+ - unsigned short flags, char read_write,
+ - u8 command, int size, union i2c_smbus_data * data);
+ + int (*master_xfer)(struct i2c_adapter *adap, struct i2c_msg *msgs,
+ + int num);
+ + int (*smbus_xfer)(struct i2c_adapter *adap, u16 addr,
+ + unsigned short flags, char read_write,
+ + u8 command, int size, union i2c_smbus_data * data);
+
+ + /* These are like the previous calls, but they will only start
+ + the operation. The poll call will be called periodically
+ + to drive the operation of the bus. Each of these calls
+ + should set the result on an error, and set the timeout as
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ necessary. Note that even interrupt driven drivers need to
+ poll so they can time out operations. When the operation
+ is complete, these should call i2c_op_done(). Note that
+ all the data structures passed in are guaranteed to be kept
+ around until the operation completes. These may be called
+ from interrupt context. If the start operation fails, this
+ should set the result and call i2c_op_done(). */
+ void (*master_start)(struct i2c_adapter *adap,
+ struct i2c_op_q_entry *entry);
+ void (*smbus_start)(struct i2c_adapter *adap,
+ struct i2c_op_q_entry *entry);
+ /* us_since_last_poll is the amount of time since the last
+ time poll was called. Note that this may be *less* than the
+ time you requested, so always use this number and don't
+ assume it's the one you gave it. This time is approximate
+ and is only guaranteed to be >= the time since the last
+ poll. The value may be zero. */
+ void (*poll)(struct i2c_adapter *adap,
+ struct i2c_op_q_entry *entry,
+ unsigned int us_since_last_poll);

        /* ---- these optional/future use for some adapter types.*/
        int (*slave_send)(struct i2c_adapter *,char*,int);
@@ -212,6 +288,21 @@
};

/*
+ * The timer has it's own separately allocated data structure because
+ * it needs to be able to exist even if the adapter is deleted (due to
+ * timer cancellation races).
+ */
+struct i2c_timer {
+ spinlock_t lock;
+ int deleted;
+ struct timer_list timer;
+ int running;
+ unsigned int next_call_time;
+ struct i2c_adapter *adapter;
+ unsigned int sequence;
+};
+
+/*
+ * i2c_adapter is the structure used to identify a physical i2c bus along
+ * with the access algorithms necessary to access it.
+ */
@@ -228,8 +319,15 @@
        int (*client_unregister)(struct i2c_client *);

        /* data fields that are valid for all devices */
+ struct list_head q;
+ spinlock_t q_lock;
```


Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ /* Note that this is not a union because an smbus operation
+ may be converted into an i2c operation (thus both
+ structures will be used). The data in these may be changed
+ by the driver. */
+ struct {
+ struct i2c_msg *msgs;
+ int num;
+ } i2c;
+ struct {
+ /* Addr and flags are filled in by the non-blocking
+ send routine that takes a client. */
+ u16 addr;
+ unsigned short flags;
+
+ char read_write;
+ u8 command;
+
+ /* Note that the size is *not* the length of the data.
+ It is the transaction type, like I2C_SMBUS_QUICK
+ and the ones after that below. If this is a block
+ transaction, the length of the rest of the data is
+ in the first byte of the data, for both transmit
+ and receive. */
+ int size;
+ union i2c_smbus_data *data;
+ } smbus;
+
+ /*****
+ /* For use by the bus interface. The bus interface sets the
+ timeout in microseconds until the next poll operation.
+ This *must* be set in the start operation. The time_left
+ and data can be used for anything the bus interface likes.
+ data will be set to NULL before being started so the bus
+ interface can use that to tell if it has been set up
+ yet. */
+ unsigned int call_again_us;
+ long time_left;
+ void *data;
+
+ /*****
+ /* Internals */
+ struct list_head link;
+ struct completion *start;
+ unsigned int started : 1;
+ unsigned int completed : 1;
+ unsigned int use_timer : 1;
+ u8 swpec;
+ u8 partial;
+ void (*complete)(struct i2c_adapter *adap,
+ struct i2c_op_q_entry *entry);
+
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ /* It's wierd, but we use a usecount to track if an q entry is
+ in use and when it should be reported back to the user. */
+ atomic_t usecount;
+
+ /* These are here for SMBus emulation over I2C. I don't like
+ them taking this much room in the data structure, but they
+ need to be available in this case. */
+ unsigned char msgbuf0[34];
+ unsigned char msgbuf1[34];
+ struct i2c_msg msg[2];
+};
+
+ /* To determine what functionality is present */

#define I2C_FUNC_I2C 0x00000001
@@ -423,22 +610,22 @@
#define I2C_FUNC_SMBUS_READ_BLOCK_DATA_PEC 0x40000000 /* SMBus 2.0 */
#define I2C_FUNC_SMBUS_WRITE_BLOCK_DATA_PEC 0x80000000 /* SMBus 2.0 */

-#define I2C_FUNC_SMBUS_BYTE I2C_FUNC_SMBUS_READ_BYTE | \
- I2C_FUNC_SMBUS_WRITE_BYTE
-#define I2C_FUNC_SMBUS_BYTE_DATA I2C_FUNC_SMBUS_READ_BYTE_DATA | \
- I2C_FUNC_SMBUS_WRITE_BYTE_DATA
-#define I2C_FUNC_SMBUS_WORD_DATA I2C_FUNC_SMBUS_READ_WORD_DATA | \
- I2C_FUNC_SMBUS_WRITE_WORD_DATA
-#define I2C_FUNC_SMBUS_BLOCK_DATA I2C_FUNC_SMBUS_READ_BLOCK_DATA | \
- I2C_FUNC_SMBUS_WRITE_BLOCK_DATA
-#define I2C_FUNC_SMBUS_I2C_BLOCK I2C_FUNC_SMBUS_READ_I2C_BLOCK | \
- I2C_FUNC_SMBUS_WRITE_I2C_BLOCK
-#define I2C_FUNC_SMBUS_I2C_BLOCK_2 I2C_FUNC_SMBUS_READ_I2C_BLOCK_2 | \
- I2C_FUNC_SMBUS_WRITE_I2C_BLOCK_2
-#define I2C_FUNC_SMBUS_BLOCK_DATA_PEC I2C_FUNC_SMBUS_READ_BLOCK_DATA_PEC | \
- I2C_FUNC_SMBUS_WRITE_BLOCK_DATA_PEC
-#define I2C_FUNC_SMBUS_WORD_DATA_PEC I2C_FUNC_SMBUS_READ_WORD_DATA_PEC | \
- I2C_FUNC_SMBUS_WRITE_WORD_DATA_PEC
+#define I2C_FUNC_SMBUS_BYTE (I2C_FUNC_SMBUS_READ_BYTE | \
+ I2C_FUNC_SMBUS_WRITE_BYTE)
+#define I2C_FUNC_SMBUS_BYTE_DATA (I2C_FUNC_SMBUS_READ_BYTE_DATA | \
+ I2C_FUNC_SMBUS_WRITE_BYTE_DATA)
+#define I2C_FUNC_SMBUS_WORD_DATA (I2C_FUNC_SMBUS_READ_WORD_DATA | \
+ I2C_FUNC_SMBUS_WRITE_WORD_DATA)
+#define I2C_FUNC_SMBUS_BLOCK_DATA (I2C_FUNC_SMBUS_READ_BLOCK_DATA | \
+ I2C_FUNC_SMBUS_WRITE_BLOCK_DATA)
+#define I2C_FUNC_SMBUS_I2C_BLOCK (I2C_FUNC_SMBUS_READ_I2C_BLOCK | \
+ I2C_FUNC_SMBUS_WRITE_I2C_BLOCK)
+#define I2C_FUNC_SMBUS_I2C_BLOCK_2 (I2C_FUNC_SMBUS_READ_I2C_BLOCK_2 | \
+ I2C_FUNC_SMBUS_WRITE_I2C_BLOCK_2)
+#define I2C_FUNC_SMBUS_BLOCK_DATA_PEC (I2C_FUNC_SMBUS_READ_BLOCK_DATA_PEC | \
+ I2C_FUNC_SMBUS_WRITE_BLOCK_DATA_PEC)
```

Linux-Kernel: [PATCH] Add a non-blocking I2C interface

```
+ #define I2C_FUNC_SMBUS_WORD_DATA_PEC (I2C_FUNC_SMBUS_READ_WORD_DATA_PEC |\
+ I2C_FUNC_SMBUS_WRITE_WORD_DATA_PEC)

# define I2C_FUNC_SMBUS_READ_BYTE_PEC I2C_FUNC_SMBUS_READ_BYTE_DATA
# define I2C_FUNC_SMBUS_WRITE_BYTE_PEC I2C_FUNC_SMBUS_WRITE_BYTE_DATA
@@ -447,14 +634,14 @@
# define I2C_FUNC_SMBUS_BYTE_PEC I2C_FUNC_SMBUS_BYTE_DATA
# define I2C_FUNC_SMBUS_BYTE_DATA_PEC I2C_FUNC_SMBUS_WORD_DATA

- #define I2C_FUNC_SMBUS_EMUL I2C_FUNC_SMBUS_QUICK |\
- I2C_FUNC_SMBUS_BYTE |\
- I2C_FUNC_SMBUS_BYTE_DATA |\
- I2C_FUNC_SMBUS_WORD_DATA |\
- I2C_FUNC_SMBUS_PROC_CALL |\
- I2C_FUNC_SMBUS_WRITE_BLOCK_DATA |\
- I2C_FUNC_SMBUS_WRITE_BLOCK_DATA_PEC |\
- I2C_FUNC_SMBUS_I2C_BLOCK
+ #define I2C_FUNC_SMBUS_EMUL (I2C_FUNC_SMBUS_QUICK |\
+ I2C_FUNC_SMBUS_BYTE |\
+ I2C_FUNC_SMBUS_BYTE_DATA |\
+ I2C_FUNC_SMBUS_WORD_DATA |\
+ I2C_FUNC_SMBUS_PROC_CALL |\
+ I2C_FUNC_SMBUS_WRITE_BLOCK_DATA |\
+ I2C_FUNC_SMBUS_WRITE_BLOCK_DATA_PEC |\
+ I2C_FUNC_SMBUS_I2C_BLOCK)

/*
 * Data for SMBus Messages
```

—
To unsubscribe from this list: send the line "unsubscribe linux-kernel" in
the body of a message to majordomo@vger.kernel.org
More majordomo info at <http://vger.kernel.org/majordomo-info.html>
Please read the FAQ at <http://www.tux.org/lkml/>