From dac90dc23211996286c3b934057d0df77cfef0b4 Mon Sep 17 00:00:00 2001

From: =?UTF-8?q?Lothar=20Wa=C3=9Fmann?= <LW@KARO-electronics.de>

Date: Fri, 28 Mar 2014 09:23:02 -0700

Subject: Input: edt-ft5x06 - add DT support

MIME-Version: 1.0

Content-Type: text/plain; charset=UTF-8

Content-Transfer-Encoding: 8bit

Signed-off-by: Lothar WaÃŸmann <LW@KARO-electronics.de>

Signed-off-by: Dmitry Torokhov <dmitry.torokhov@gmail.com>

---

.../bindings/input/touchscreen/edt-ft5x06.txt | 55 ++++++++

drivers/input/touchscreen/edt-ft5x06.c | 145 ++++++++++++++++-----

2 files changed, 169 insertions(+), 31 deletions(-)

create mode 100644 Documentation/devicetree/bindings/input/touchscreen/edt-ft5x06.txt

diff --git a/Documentation/devicetree/bindings/input/touchscreen/edt-ft5x06.txt b/Documentation/devicetree/bindings/input/touchscreen/edt-ft5x06.txt

new file mode 100644

index 0000000..76db967

--- /dev/null

+++ b/Documentation/devicetree/bindings/input/touchscreen/edt-ft5x06.txt

@@ -0,0 +1,55 @@

+FocalTech EDT-FT5x06 Polytouch driver

+=====================================

+

+There are 3 variants of the chip for various touch panel sizes

+FT5206GE1 2.8" .. 3.8"

+FT5306DE4 4.3" .. 7"

+FT5406EE8 7" .. 8.9"

+

+The software interface is identical for all those chips, so that

+currently there is no need for the driver to distinguish between the

+different chips. Nevertheless distinct compatible strings are used so

+that a distinction can be added if necessary without changing the DT

+bindings.

+

+

+Required properties:

+ - compatible: "edt,edt-ft5206"

+ or: "edt,edt-ft5306"

+ or: "edt,edt-ft5406"

+

+ - reg: I2C slave address of the chip (0x38)

+ - interrupt-parent: a phandle pointing to the interrupt controller

+ serving the interrupt for this chip

+ - interrupts: interrupt specification for the touchdetect

+ interrupt

+

+Optional properties:

+ - reset-gpios: GPIO specification for the RESET input

+ - wake-gpios: GPIO specification for the WAKE input

+

+ - pinctrl-names: should be "default"

+ - pinctrl-0: a phandle pointing to the pin settings for the

+ control gpios

+

+ - threshold: allows setting the "click"-threshold in the range

+ from 20 to 80.

+

+ - gain: allows setting the sensitivity in the range from 0 to

+ 31. Note that lower values indicate higher

+ sensitivity.

+

+ - offset: allows setting the edge compensation in the range from

+ 0 to 31.

+

+Example:

+ polytouch: edt-ft5x06@38 {

+ compatible = "edt,edt-ft5406", "edt,edt-ft5x06";

+ reg = <0x38>;

+ pinctrl-names = "default";

+ pinctrl-0 = <&edt\_ft5x06\_pins>;

+ interrupt-parent = <&gpio2>;

+ interrupts = <5 0>;

+ reset-gpios = <&gpio2 6 1>;

+ wake-gpios = <&gpio4 9 0>;

+ };

diff --git a/drivers/input/touchscreen/edt-ft5x06.c b/drivers/input/touchscreen/edt-ft5x06.c

index 412a85e..ee3434f 100644

--- a/drivers/input/touchscreen/edt-ft5x06.c

+++ b/drivers/input/touchscreen/edt-ft5x06.c

@@ -1,5 +1,7 @@

/\*

\* Copyright (C) 2012 Simon Budig, <simon.budig@kernelconcepts.de>

+ \* Daniel Wagener <daniel.wagener@kernelconcepts.de> (M09 firmware support)

+ \* Lothar WaÃŸmann <LW@KARO-electronics.de> (DT support)

\*

\* This software is licensed under the terms of the GNU General Public

\* License version 2, as published by the Free Software Foundation, and

@@ -33,6 +35,7 @@

#include <linux/debugfs.h>

#include <linux/slab.h>

#include <linux/gpio.h>

+#include <linux/of\_gpio.h>

#include <linux/input/mt.h>

#include <linux/input/edt-ft5x06.h>

@@ -45,6 +48,14 @@

#define WORK\_REGISTER\_NUM\_X 0x33

#define WORK\_REGISTER\_NUM\_Y 0x34

+#define M09\_REGISTER\_THRESHOLD 0x80

+#define M09\_REGISTER\_GAIN 0x92

+#define M09\_REGISTER\_OFFSET 0x93

+#define M09\_REGISTER\_NUM\_X 0x94

+#define M09\_REGISTER\_NUM\_Y 0x95

+

+#define NO\_REGISTER 0xff

+

#define WORK\_REGISTER\_OPMODE 0x3c

#define FACTORY\_REGISTER\_OPMODE 0x01

@@ -59,12 +70,30 @@

#define EDT\_RAW\_DATA\_RETRIES 100

#define EDT\_RAW\_DATA\_DELAY 1 /\* msec \*/

+enum edt\_ver {

+ M06,

+ M09,

+};

+

+struct edt\_reg\_addr {

+ int reg\_threshold;

+ int reg\_report\_rate;

+ int reg\_gain;

+ int reg\_offset;

+ int reg\_num\_x;

+ int reg\_num\_y;

+};

+

struct edt\_ft5x06\_ts\_data {

struct i2c\_client \*client;

struct input\_dev \*input;

u16 num\_x;

u16 num\_y;

+ int reset\_pin;

+ int irq\_pin;

+ int wake\_pin;

+

#if defined(CONFIG\_DEBUG\_FS)

struct dentry \*debug\_dir;

u8 \*raw\_buffer;

@@ -79,6 +108,9 @@ struct edt\_ft5x06\_ts\_data {

int report\_rate;

char name[EDT\_NAME\_LEN];

+

+ struct edt\_reg\_addr reg\_addr;

+ enum edt\_ver version;

};

static int edt\_ft5x06\_ts\_readwrite(struct i2c\_client \*client,

@@ -136,33 +168,58 @@ static irqreturn\_t edt\_ft5x06\_ts\_isr(int irq, void \*dev\_id)

{

struct edt\_ft5x06\_ts\_data \*tsdata = dev\_id;

struct device \*dev = &tsdata->client->dev;

- u8 cmd = 0xf9;

- u8 rdbuf[26];

+ u8 cmd;

+ u8 rdbuf[29];

int i, type, x, y, id;

+ int offset, tplen, datalen;

int error;

+ switch (tsdata->version) {

+ case M06:

+ cmd = 0xf9; /\* tell the controller to send touch data \*/

+ offset = 5; /\* where the actual touch data starts \*/

+ tplen = 4; /\* data comes in so called frames \*/

+ datalen = 26; /\* how much bytes to listen for \*/

+ break;

+

+ case M09:

+ cmd = 0x02;

+ offset = 1;

+ tplen = 6;

+ datalen = 29;

+ break;

+

+ default:

+ goto out;

+ }

+

memset(rdbuf, 0, sizeof(rdbuf));

error = edt\_ft5x06\_ts\_readwrite(tsdata->client,

sizeof(cmd), &cmd,

- sizeof(rdbuf), rdbuf);

+ datalen, rdbuf);

if (error) {

dev\_err\_ratelimited(dev, "Unable to fetch data, error: %d\n",

error);

goto out;

}

- if (rdbuf[0] != 0xaa || rdbuf[1] != 0xaa || rdbuf[2] != 26) {

- dev\_err\_ratelimited(dev, "Unexpected header: %02x%02x%02x!\n",

- rdbuf[0], rdbuf[1], rdbuf[2]);

- goto out;

- }

+ /\* M09 does not send header or CRC \*/

+ if (tsdata->version == M06) {

+ if (rdbuf[0] != 0xaa || rdbuf[1] != 0xaa ||

+ rdbuf[2] != datalen) {

+ dev\_err\_ratelimited(dev,

+ "Unexpected header: %02x%02x%02x!\n",

+ rdbuf[0], rdbuf[1], rdbuf[2]);

+ goto out;

+ }

- if (!edt\_ft5x06\_ts\_check\_crc(tsdata, rdbuf, 26))

- goto out;

+ if (!edt\_ft5x06\_ts\_check\_crc(tsdata, rdbuf, datalen))

+ goto out;

+ }

for (i = 0; i < MAX\_SUPPORT\_POINTS; i++) {

- u8 \*buf = &rdbuf[i \* 4 + 5];

+ u8 \*buf = &rdbuf[i \* tplen + offset];

bool down;

type = buf[0] >> 6;

@@ -170,10 +227,14 @@ static irqreturn\_t edt\_ft5x06\_ts\_isr(int irq, void \*dev\_id)

if (type == TOUCH\_EVENT\_RESERVED)

continue;

+ /\* M06 sometimes sends bogus coordinates in TOUCH\_DOWN \*/

+ if (tsdata->version == M06 && type == TOUCH\_EVENT\_DOWN)

+ continue;

+

x = ((buf[0] << 8) | buf[1]) & 0x0fff;

y = ((buf[2] << 8) | buf[3]) & 0x0fff;

id = (buf[2] >> 4) & 0x0f;

- down = (type != TOUCH\_EVENT\_UP);

+ down = type != TOUCH\_EVENT\_UP;

input\_mt\_slot(tsdata->input, id);

input\_mt\_report\_slot\_state(tsdata->input, MT\_TOOL\_FINGER, down);

@@ -197,12 +258,24 @@ static int edt\_ft5x06\_register\_write(struct edt\_ft5x06\_ts\_data \*tsdata,

{

u8 wrbuf[4];

- wrbuf[0] = tsdata->factory\_mode ? 0xf3 : 0xfc;

- wrbuf[1] = tsdata->factory\_mode ? addr & 0x7f : addr & 0x3f;

- wrbuf[2] = value;

- wrbuf[3] = wrbuf[0] ^ wrbuf[1] ^ wrbuf[2];

-

- return edt\_ft5x06\_ts\_readwrite(tsdata->client, 4, wrbuf, 0, NULL);

+ switch (tsdata->version) {

+ case M06:

+ wrbuf[0] = tsdata->factory\_mode ? 0xf3 : 0xfc;

+ wrbuf[1] = tsdata->factory\_mode ? addr & 0x7f : addr & 0x3f;

+ wrbuf[2] = value;

+ wrbuf[3] = wrbuf[0] ^ wrbuf[1] ^ wrbuf[2];

+ return edt\_ft5x06\_ts\_readwrite(tsdata->client, 4,

+ wrbuf, 0, NULL);

+ case M09:

+ wrbuf[0] = addr;

+ wrbuf[1] = value;

+

+ return edt\_ft5x06\_ts\_readwrite(tsdata->client, 2,

+ wrbuf, 0, NULL);

+

+ default:

+ return -EINVAL;

+ }

}

static int edt\_ft5x06\_register\_read(struct edt\_ft5x06\_ts\_data \*tsdata,

@@ -211,19 +284,36 @@ static int edt\_ft5x06\_register\_read(struct edt\_ft5x06\_ts\_data \*tsdata,

u8 wrbuf[2], rdbuf[2];

int error;

- wrbuf[0] = tsdata->factory\_mode ? 0xf3 : 0xfc;

- wrbuf[1] = tsdata->factory\_mode ? addr & 0x7f : addr & 0x3f;

- wrbuf[1] |= tsdata->factory\_mode ? 0x80 : 0x40;

+ switch (tsdata->version) {

+ case M06:

+ wrbuf[0] = tsdata->factory\_mode ? 0xf3 : 0xfc;

+ wrbuf[1] = tsdata->factory\_mode ? addr & 0x7f : addr & 0x3f;

+ wrbuf[1] |= tsdata->factory\_mode ? 0x80 : 0x40;

- error = edt\_ft5x06\_ts\_readwrite(tsdata->client, 2, wrbuf, 2, rdbuf);

- if (error)

- return error;

+ error = edt\_ft5x06\_ts\_readwrite(tsdata->client, 2, wrbuf, 2,

+ rdbuf);

+ if (error)

+ return error;

- if ((wrbuf[0] ^ wrbuf[1] ^ rdbuf[0]) != rdbuf[1]) {

- dev\_err(&tsdata->client->dev,

- "crc error: 0x%02x expected, got 0x%02x\n",

- wrbuf[0] ^ wrbuf[1] ^ rdbuf[0], rdbuf[1]);

- return -EIO;

+ if ((wrbuf[0] ^ wrbuf[1] ^ rdbuf[0]) != rdbuf[1]) {

+ dev\_err(&tsdata->client->dev,

+ "crc error: 0x%02x expected, got 0x%02x\n",

+ wrbuf[0] ^ wrbuf[1] ^ rdbuf[0],

+ rdbuf[1]);

+ return -EIO;

+ }

+ break;

+

+ case M09:

+ wrbuf[0] = addr;

+ error = edt\_ft5x06\_ts\_readwrite(tsdata->client, 1,

+ wrbuf, 1, rdbuf);

+ if (error)

+ return error;

+ break;

+

+ default:

+ return -EINVAL;

}

return rdbuf[0];

@@ -234,19 +324,21 @@ struct edt\_ft5x06\_attribute {

size\_t field\_offset;

u8 limit\_low;

u8 limit\_high;

- u8 addr;

+ u8 addr\_m06;

+ u8 addr\_m09;

};

-#define EDT\_ATTR(\_field, \_mode, \_addr, \_limit\_low, \_limit\_high) \

+#define EDT\_ATTR(\_field, \_mode, \_addr\_m06, \_addr\_m09, \

+ \_limit\_low, \_limit\_high) \

struct edt\_ft5x06\_attribute edt\_ft5x06\_attr\_##\_field = { \

.dattr = \_\_ATTR(\_field, \_mode, \

edt\_ft5x06\_setting\_show, \

edt\_ft5x06\_setting\_store), \

- .field\_offset = \

- offsetof(struct edt\_ft5x06\_ts\_data, \_field), \

+ .field\_offset = offsetof(struct edt\_ft5x06\_ts\_data, \_field), \

+ .addr\_m06 = \_addr\_m06, \

+ .addr\_m09 = \_addr\_m09, \

.limit\_low = \_limit\_low, \

.limit\_high = \_limit\_high, \

- .addr = \_addr, \

}

static ssize\_t edt\_ft5x06\_setting\_show(struct device \*dev,

@@ -257,10 +349,11 @@ static ssize\_t edt\_ft5x06\_setting\_show(struct device \*dev,

struct edt\_ft5x06\_ts\_data \*tsdata = i2c\_get\_clientdata(client);

struct edt\_ft5x06\_attribute \*attr =

container\_of(dattr, struct edt\_ft5x06\_attribute, dattr);

- u8 \*field = (u8 \*)((char \*)tsdata + attr->field\_offset);

+ u8 \*field = (u8 \*)tsdata + attr->field\_offset;

int val;

size\_t count = 0;

int error = 0;

+ u8 addr;

mutex\_lock(&tsdata->mutex);

@@ -269,15 +362,33 @@ static ssize\_t edt\_ft5x06\_setting\_show(struct device \*dev,

goto out;

}

- val = edt\_ft5x06\_register\_read(tsdata, attr->addr);

- if (val < 0) {

- error = val;

- dev\_err(&tsdata->client->dev,

- "Failed to fetch attribute %s, error %d\n",

- dattr->attr.name, error);

+ switch (tsdata->version) {

+ case M06:

+ addr = attr->addr\_m06;

+ break;

+

+ case M09:

+ addr = attr->addr\_m09;

+ break;

+

+ default:

+ error = -ENODEV;

goto out;

}

+ if (addr != NO\_REGISTER) {

+ val = edt\_ft5x06\_register\_read(tsdata, addr);

+ if (val < 0) {

+ error = val;

+ dev\_err(&tsdata->client->dev,

+ "Failed to fetch attribute %s, error %d\n",

+ dattr->attr.name, error);

+ goto out;

+ }

+ } else {

+ val = \*field;

+ }

+

if (val != \*field) {

dev\_warn(&tsdata->client->dev,

"%s: read (%d) and stored value (%d) differ\n",

@@ -299,9 +410,10 @@ static ssize\_t edt\_ft5x06\_setting\_store(struct device \*dev,

struct edt\_ft5x06\_ts\_data \*tsdata = i2c\_get\_clientdata(client);

struct edt\_ft5x06\_attribute \*attr =

container\_of(dattr, struct edt\_ft5x06\_attribute, dattr);

- u8 \*field = (u8 \*)((char \*)tsdata + attr->field\_offset);

+ u8 \*field = (u8 \*)tsdata + attr->field\_offset;

unsigned int val;

int error;

+ u8 addr;

mutex\_lock(&tsdata->mutex);

@@ -319,14 +431,29 @@ static ssize\_t edt\_ft5x06\_setting\_store(struct device \*dev,

goto out;

}

- error = edt\_ft5x06\_register\_write(tsdata, attr->addr, val);

- if (error) {

- dev\_err(&tsdata->client->dev,

- "Failed to update attribute %s, error: %d\n",

- dattr->attr.name, error);

+ switch (tsdata->version) {

+ case M06:

+ addr = attr->addr\_m06;

+ break;

+

+ case M09:

+ addr = attr->addr\_m09;

+ break;

+

+ default:

+ error = -ENODEV;

goto out;

}

+ if (addr != NO\_REGISTER) {

+ error = edt\_ft5x06\_register\_write(tsdata, addr, val);

+ if (error) {

+ dev\_err(&tsdata->client->dev,

+ "Failed to update attribute %s, error: %d\n",

+ dattr->attr.name, error);

+ goto out;

+ }

+ }

\*field = val;

out:

@@ -334,12 +461,14 @@ out:

return error ?: count;

}

-static EDT\_ATTR(gain, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_GAIN, 0, 31);

-static EDT\_ATTR(offset, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_OFFSET, 0, 31);

-static EDT\_ATTR(threshold, S\_IWUSR | S\_IRUGO,

- WORK\_REGISTER\_THRESHOLD, 20, 80);

-static EDT\_ATTR(report\_rate, S\_IWUSR | S\_IRUGO,

- WORK\_REGISTER\_REPORT\_RATE, 3, 14);

+static EDT\_ATTR(gain, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_GAIN,

+ M09\_REGISTER\_GAIN, 0, 31);

+static EDT\_ATTR(offset, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_OFFSET,

+ M09\_REGISTER\_OFFSET, 0, 31);

+static EDT\_ATTR(threshold, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_THRESHOLD,

+ M09\_REGISTER\_THRESHOLD, 20, 80);

+static EDT\_ATTR(report\_rate, S\_IWUSR | S\_IRUGO, WORK\_REGISTER\_REPORT\_RATE,

+ NO\_REGISTER, 3, 14);

static struct attribute \*edt\_ft5x06\_attrs[] = {

&edt\_ft5x06\_attr\_gain.dattr.attr,

@@ -374,6 +503,9 @@ static int edt\_ft5x06\_factory\_mode(struct edt\_ft5x06\_ts\_data \*tsdata)

}

/\* mode register is 0x3c when in the work mode \*/

+ if (tsdata->version == M09)

+ goto m09\_out;

+

error = edt\_ft5x06\_register\_write(tsdata, WORK\_REGISTER\_OPMODE, 0x03);

if (error) {

dev\_err(&client->dev,

@@ -406,12 +538,18 @@ err\_out:

enable\_irq(client->irq);

return error;

+

+m09\_out:

+ dev\_err(&client->dev, "No factory mode support for M09\n");

+ return -EINVAL;

+

}

static int edt\_ft5x06\_work\_mode(struct edt\_ft5x06\_ts\_data \*tsdata)

{

struct i2c\_client \*client = tsdata->client;

int retries = EDT\_SWITCH\_MODE\_RETRIES;

+ struct edt\_reg\_addr \*reg\_addr = &tsdata->reg\_addr;

int ret;

int error;

@@ -444,13 +582,14 @@ static int edt\_ft5x06\_work\_mode(struct edt\_ft5x06\_ts\_data \*tsdata)

tsdata->raw\_buffer = NULL;

/\* restore parameters \*/

- edt\_ft5x06\_register\_write(tsdata, WORK\_REGISTER\_THRESHOLD,

+ edt\_ft5x06\_register\_write(tsdata, reg\_addr->reg\_threshold,

tsdata->threshold);

- edt\_ft5x06\_register\_write(tsdata, WORK\_REGISTER\_GAIN,

+ edt\_ft5x06\_register\_write(tsdata, reg\_addr->reg\_gain,

tsdata->gain);

- edt\_ft5x06\_register\_write(tsdata, WORK\_REGISTER\_OFFSET,

+ edt\_ft5x06\_register\_write(tsdata, reg\_addr->reg\_offset,

tsdata->offset);

- edt\_ft5x06\_register\_write(tsdata, WORK\_REGISTER\_REPORT\_RATE,

+ if (reg\_addr->reg\_report\_rate)

+ edt\_ft5x06\_register\_write(tsdata, reg\_addr->reg\_report\_rate,

tsdata->report\_rate);

enable\_irq(client->irq);

@@ -479,7 +618,7 @@ static int edt\_ft5x06\_debugfs\_mode\_set(void \*data, u64 mode)

if (mode != tsdata->factory\_mode) {

retval = mode ? edt\_ft5x06\_factory\_mode(tsdata) :

- edt\_ft5x06\_work\_mode(tsdata);

+ edt\_ft5x06\_work\_mode(tsdata);

}

mutex\_unlock(&tsdata->mutex);

@@ -568,7 +707,6 @@ out:

return error ?: read;

};

-

static const struct file\_operations debugfs\_raw\_data\_fops = {

.open = simple\_open,

.read = edt\_ft5x06\_debugfs\_raw\_data\_read,

@@ -594,8 +732,7 @@ edt\_ft5x06\_ts\_prepare\_debugfs(struct edt\_ft5x06\_ts\_data \*tsdata,

static void

edt\_ft5x06\_ts\_teardown\_debugfs(struct edt\_ft5x06\_ts\_data \*tsdata)

{

- if (tsdata->debug\_dir)

- debugfs\_remove\_recursive(tsdata->debug\_dir);

+ debugfs\_remove\_recursive(tsdata->debug\_dir);

kfree(tsdata->raw\_buffer);

}

@@ -614,58 +751,100 @@ edt\_ft5x06\_ts\_teardown\_debugfs(struct edt\_ft5x06\_ts\_data \*tsdata)

#endif /\* CONFIG\_DEBUGFS \*/

-

-

static int edt\_ft5x06\_ts\_reset(struct i2c\_client \*client,

- int reset\_pin)

+ struct edt\_ft5x06\_ts\_data \*tsdata)

{

int error;

- if (gpio\_is\_valid(reset\_pin)) {

+ if (gpio\_is\_valid(tsdata->wake\_pin)) {

+ error = devm\_gpio\_request\_one(&client->dev,

+ tsdata->wake\_pin, GPIOF\_OUT\_INIT\_LOW,

+ "edt-ft5x06 wake");

+ if (error) {

+ dev\_err(&client->dev,

+ "Failed to request GPIO %d as wake pin, error %d\n",

+ tsdata->wake\_pin, error);

+ return error;

+ }

+

+ msleep(5);

+ gpio\_set\_value(tsdata->wake\_pin, 1);

+ }

+ if (gpio\_is\_valid(tsdata->reset\_pin)) {

/\* this pulls reset down, enabling the low active reset \*/

- error = devm\_gpio\_request\_one(&client->dev, reset\_pin,

- GPIOF\_OUT\_INIT\_LOW,

- "edt-ft5x06 reset");

+ error = devm\_gpio\_request\_one(&client->dev,

+ tsdata->reset\_pin, GPIOF\_OUT\_INIT\_LOW,

+ "edt-ft5x06 reset");

if (error) {

dev\_err(&client->dev,

"Failed to request GPIO %d as reset pin, error %d\n",

- reset\_pin, error);

+ tsdata->reset\_pin, error);

return error;

}

- mdelay(50);

- gpio\_set\_value(reset\_pin, 1);

- mdelay(100);

+ msleep(5);

+ gpio\_set\_value(tsdata->reset\_pin, 1);

+ msleep(300);

}

return 0;

}

static int edt\_ft5x06\_ts\_identify(struct i2c\_client \*client,

- char \*model\_name,

- char \*fw\_version)

+ struct edt\_ft5x06\_ts\_data \*tsdata,

+ char \*fw\_version)

{

u8 rdbuf[EDT\_NAME\_LEN];

char \*p;

int error;

+ char \*model\_name = tsdata->name;

+ /\* see what we find if we assume it is a M06 \*

+ \* if we get less than EDT\_NAME\_LEN, we don't want

+ \* to have garbage in there

+ \*/

+ memset(rdbuf, 0, sizeof(rdbuf));

error = edt\_ft5x06\_ts\_readwrite(client, 1, "\xbb",

EDT\_NAME\_LEN - 1, rdbuf);

if (error)

return error;

- /\* remove last '$' end marker \*/

- rdbuf[EDT\_NAME\_LEN - 1] = '\0';

- if (rdbuf[EDT\_NAME\_LEN - 2] == '$')

- rdbuf[EDT\_NAME\_LEN - 2] = '\0';

+ /\* if we find something consistent, stay with that assumption

+ \* at least M09 won't send 3 bytes here

+ \*/

+ if (!(strncasecmp(rdbuf + 1, "EP0", 3))) {

+ tsdata->version = M06;

+

+ /\* remove last '$' end marker \*/

+ rdbuf[EDT\_NAME\_LEN - 1] = '\0';

+ if (rdbuf[EDT\_NAME\_LEN - 2] == '$')

+ rdbuf[EDT\_NAME\_LEN - 2] = '\0';

+

+ /\* look for Model/Version separator \*/

+ p = strchr(rdbuf, '\*');

+ if (p)

+ \*p++ = '\0';

+ strlcpy(model\_name, rdbuf + 1, EDT\_NAME\_LEN);

+ strlcpy(fw\_version, p ? p : "", EDT\_NAME\_LEN);

+ } else {

+ /\* since there are only two versions around (M06, M09) \*/

+ tsdata->version = M09;

+

+ error = edt\_ft5x06\_ts\_readwrite(client, 1, "\xA6",

+ 2, rdbuf);

+ if (error)

+ return error;

+

+ strlcpy(fw\_version, rdbuf, 2);

- /\* look for Model/Version separator \*/

- p = strchr(rdbuf, '\*');

- if (p)

- \*p++ = '\0';

+ error = edt\_ft5x06\_ts\_readwrite(client, 1, "\xA8",

+ 1, rdbuf);

+ if (error)

+ return error;

- strlcpy(model\_name, rdbuf + 1, EDT\_NAME\_LEN);

- strlcpy(fw\_version, p ? p : "", EDT\_NAME\_LEN);

+ snprintf(model\_name, EDT\_NAME\_LEN, "EP0%i%i0M09",

+ rdbuf[0] >> 4, rdbuf[0] & 0x0F);

+ }

return 0;

}

@@ -675,33 +854,104 @@ static int edt\_ft5x06\_ts\_identify(struct i2c\_client \*client,

pdata->name <= edt\_ft5x06\_attr\_##name.limit\_high) \

edt\_ft5x06\_register\_write(tsdata, reg, pdata->name)

+#define EDT\_GET\_PROP(name, reg) { \

+ u32 val; \

+ if (of\_property\_read\_u32(np, #name, &val) == 0) \

+ edt\_ft5x06\_register\_write(tsdata, reg, val); \

+}

+

+static void edt\_ft5x06\_ts\_get\_dt\_defaults(struct device\_node \*np,

+ struct edt\_ft5x06\_ts\_data \*tsdata)

+{

+ struct edt\_reg\_addr \*reg\_addr = &tsdata->reg\_addr;

+

+ EDT\_GET\_PROP(threshold, reg\_addr->reg\_threshold);

+ EDT\_GET\_PROP(gain, reg\_addr->reg\_gain);

+ EDT\_GET\_PROP(offset, reg\_addr->reg\_offset);

+}

+

static void

edt\_ft5x06\_ts\_get\_defaults(struct edt\_ft5x06\_ts\_data \*tsdata,

const struct edt\_ft5x06\_platform\_data \*pdata)

{

+ struct edt\_reg\_addr \*reg\_addr = &tsdata->reg\_addr;

+

if (!pdata->use\_parameters)

return;

/\* pick up defaults from the platform data \*/

- EDT\_ATTR\_CHECKSET(threshold, WORK\_REGISTER\_THRESHOLD);

- EDT\_ATTR\_CHECKSET(gain, WORK\_REGISTER\_GAIN);

- EDT\_ATTR\_CHECKSET(offset, WORK\_REGISTER\_OFFSET);

- EDT\_ATTR\_CHECKSET(report\_rate, WORK\_REGISTER\_REPORT\_RATE);

+ EDT\_ATTR\_CHECKSET(threshold, reg\_addr->reg\_threshold);

+ EDT\_ATTR\_CHECKSET(gain, reg\_addr->reg\_gain);

+ EDT\_ATTR\_CHECKSET(offset, reg\_addr->reg\_offset);

+ if (reg\_addr->reg\_report\_rate != NO\_REGISTER)

+ EDT\_ATTR\_CHECKSET(report\_rate, reg\_addr->reg\_report\_rate);

}

static void

edt\_ft5x06\_ts\_get\_parameters(struct edt\_ft5x06\_ts\_data \*tsdata)

{

+ struct edt\_reg\_addr \*reg\_addr = &tsdata->reg\_addr;

+

tsdata->threshold = edt\_ft5x06\_register\_read(tsdata,

- WORK\_REGISTER\_THRESHOLD);

- tsdata->gain = edt\_ft5x06\_register\_read(tsdata, WORK\_REGISTER\_GAIN);

- tsdata->offset = edt\_ft5x06\_register\_read(tsdata, WORK\_REGISTER\_OFFSET);

- tsdata->report\_rate = edt\_ft5x06\_register\_read(tsdata,

- WORK\_REGISTER\_REPORT\_RATE);

- tsdata->num\_x = edt\_ft5x06\_register\_read(tsdata, WORK\_REGISTER\_NUM\_X);

- tsdata->num\_y = edt\_ft5x06\_register\_read(tsdata, WORK\_REGISTER\_NUM\_Y);

+ reg\_addr->reg\_threshold);

+ tsdata->gain = edt\_ft5x06\_register\_read(tsdata, reg\_addr->reg\_gain);

+ tsdata->offset = edt\_ft5x06\_register\_read(tsdata, reg\_addr->reg\_offset);

+ if (reg\_addr->reg\_report\_rate != NO\_REGISTER)

+ tsdata->report\_rate = edt\_ft5x06\_register\_read(tsdata,

+ reg\_addr->reg\_report\_rate);

+ tsdata->num\_x = edt\_ft5x06\_register\_read(tsdata, reg\_addr->reg\_num\_x);

+ tsdata->num\_y = edt\_ft5x06\_register\_read(tsdata, reg\_addr->reg\_num\_y);

+}

+

+static void

+edt\_ft5x06\_ts\_set\_regs(struct edt\_ft5x06\_ts\_data \*tsdata)

+{

+ struct edt\_reg\_addr \*reg\_addr = &tsdata->reg\_addr;

+

+ switch (tsdata->version) {

+ case M06:

+ reg\_addr->reg\_threshold = WORK\_REGISTER\_THRESHOLD;

+ reg\_addr->reg\_report\_rate = WORK\_REGISTER\_REPORT\_RATE;

+ reg\_addr->reg\_gain = WORK\_REGISTER\_GAIN;

+ reg\_addr->reg\_offset = WORK\_REGISTER\_OFFSET;

+ reg\_addr->reg\_num\_x = WORK\_REGISTER\_NUM\_X;

+ reg\_addr->reg\_num\_y = WORK\_REGISTER\_NUM\_Y;

+ break;

+

+ case M09:

+ reg\_addr->reg\_threshold = M09\_REGISTER\_THRESHOLD;

+ reg\_addr->reg\_gain = M09\_REGISTER\_GAIN;

+ reg\_addr->reg\_offset = M09\_REGISTER\_OFFSET;

+ reg\_addr->reg\_num\_x = M09\_REGISTER\_NUM\_X;

+ reg\_addr->reg\_num\_y = M09\_REGISTER\_NUM\_Y;

+ break;

+ }

}

+#ifdef CONFIG\_OF

+static int edt\_ft5x06\_i2c\_ts\_probe\_dt(struct device \*dev,

+ struct edt\_ft5x06\_ts\_data \*tsdata)

+{

+ struct device\_node \*np = dev->of\_node;

+

+ /\*

+ \* irq\_pin is not needed for DT setup.

+ \* irq is associated via 'interrupts' property in DT

+ \*/

+ tsdata->irq\_pin = -EINVAL;

+ tsdata->reset\_pin = of\_get\_named\_gpio(np, "reset-gpios", 0);

+ tsdata->wake\_pin = of\_get\_named\_gpio(np, "wake-gpios", 0);

+

+ return 0;

+}

+#else

+static inline int edt\_ft5x06\_i2c\_ts\_probe\_dt(struct device \*dev,

+ struct edt\_ft5x06\_ts\_data \*tsdata)

+{

+ return -ENODEV;

+}

+#endif

+

static int edt\_ft5x06\_ts\_probe(struct i2c\_client \*client,

const struct i2c\_device\_id \*id)

{

@@ -714,32 +964,40 @@ static int edt\_ft5x06\_ts\_probe(struct i2c\_client \*client,

dev\_dbg(&client->dev, "probing for EDT FT5x06 I2C\n");

+ tsdata = devm\_kzalloc(&client->dev, sizeof(\*tsdata), GFP\_KERNEL);

+ if (!tsdata) {

+ dev\_err(&client->dev, "failed to allocate driver data.\n");

+ return -ENOMEM;

+ }

+

if (!pdata) {

- dev\_err(&client->dev, "no platform data?\n");

- return -EINVAL;

+ error = edt\_ft5x06\_i2c\_ts\_probe\_dt(&client->dev, tsdata);

+ if (error) {

+ dev\_err(&client->dev,

+ "DT probe failed and no platform data present\n");

+ return error;

+ }

+ } else {

+ tsdata->reset\_pin = pdata->reset\_pin;

+ tsdata->irq\_pin = pdata->irq\_pin;

+ tsdata->wake\_pin = -EINVAL;

}

- error = edt\_ft5x06\_ts\_reset(client, pdata->reset\_pin);

+ error = edt\_ft5x06\_ts\_reset(client, tsdata);

if (error)

return error;

- if (gpio\_is\_valid(pdata->irq\_pin)) {

- error = devm\_gpio\_request\_one(&client->dev, pdata->irq\_pin,

- GPIOF\_IN, "edt-ft5x06 irq");

+ if (gpio\_is\_valid(tsdata->irq\_pin)) {

+ error = devm\_gpio\_request\_one(&client->dev, tsdata->irq\_pin,

+ GPIOF\_IN, "edt-ft5x06 irq");

if (error) {

dev\_err(&client->dev,

"Failed to request GPIO %d, error %d\n",

- pdata->irq\_pin, error);

+ tsdata->irq\_pin, error);

return error;

}

}

- tsdata = devm\_kzalloc(&client->dev, sizeof(\*tsdata), GFP\_KERNEL);

- if (!tsdata) {

- dev\_err(&client->dev, "failed to allocate driver data.\n");

- return -ENOMEM;

- }

-

input = devm\_input\_allocate\_device(&client->dev);

if (!input) {

dev\_err(&client->dev, "failed to allocate input device.\n");

@@ -751,13 +1009,19 @@ static int edt\_ft5x06\_ts\_probe(struct i2c\_client \*client,

tsdata->input = input;

tsdata->factory\_mode = false;

- error = edt\_ft5x06\_ts\_identify(client, tsdata->name, fw\_version);

+ error = edt\_ft5x06\_ts\_identify(client, tsdata, fw\_version);

if (error) {

dev\_err(&client->dev, "touchscreen probe failed\n");

return error;

}

- edt\_ft5x06\_ts\_get\_defaults(tsdata, pdata);

+ edt\_ft5x06\_ts\_set\_regs(tsdata);

+

+ if (!pdata)

+ edt\_ft5x06\_ts\_get\_dt\_defaults(client->dev.of\_node, tsdata);

+ else

+ edt\_ft5x06\_ts\_get\_defaults(tsdata, pdata);

+

edt\_ft5x06\_ts\_get\_parameters(tsdata);

dev\_dbg(&client->dev,

@@ -787,10 +1051,10 @@ static int edt\_ft5x06\_ts\_probe(struct i2c\_client \*client,

input\_set\_drvdata(input, tsdata);

i2c\_set\_clientdata(client, tsdata);

- error = devm\_request\_threaded\_irq(&client->dev, client->irq,

- NULL, edt\_ft5x06\_ts\_isr,

- IRQF\_TRIGGER\_FALLING | IRQF\_ONESHOT,

- client->name, tsdata);

+ error = devm\_request\_threaded\_irq(&client->dev, client->irq, NULL,

+ edt\_ft5x06\_ts\_isr,

+ IRQF\_TRIGGER\_FALLING | IRQF\_ONESHOT,

+ client->name, tsdata);

if (error) {

dev\_err(&client->dev, "Unable to request touchscreen IRQ.\n");

return error;

@@ -801,19 +1065,21 @@ static int edt\_ft5x06\_ts\_probe(struct i2c\_client \*client,

return error;

error = input\_register\_device(input);

- if (error) {

- sysfs\_remove\_group(&client->dev.kobj, &edt\_ft5x06\_attr\_group);

- return error;

- }

+ if (error)

+ goto err\_remove\_attrs;

edt\_ft5x06\_ts\_prepare\_debugfs(tsdata, dev\_driver\_string(&client->dev));

device\_init\_wakeup(&client->dev, 1);

dev\_dbg(&client->dev,

- "EDT FT5x06 initialized: IRQ pin %d, Reset pin %d.\n",

- pdata->irq\_pin, pdata->reset\_pin);

+ "EDT FT5x06 initialized: IRQ %d, WAKE pin %d, Reset pin %d.\n",

+ client->irq, tsdata->wake\_pin, tsdata->reset\_pin);

return 0;

+

+err\_remove\_attrs:

+ sysfs\_remove\_group(&client->dev.kobj, &edt\_ft5x06\_attr\_group);

+ return error;

}

static int edt\_ft5x06\_ts\_remove(struct i2c\_client \*client)

@@ -852,15 +1118,26 @@ static SIMPLE\_DEV\_PM\_OPS(edt\_ft5x06\_ts\_pm\_ops,

edt\_ft5x06\_ts\_suspend, edt\_ft5x06\_ts\_resume);

static const struct i2c\_device\_id edt\_ft5x06\_ts\_id[] = {

- { "edt-ft5x06", 0 },

- { }

+ { "edt-ft5x06", 0, },

+ { /\* sentinel \*/ }

};

MODULE\_DEVICE\_TABLE(i2c, edt\_ft5x06\_ts\_id);

+#ifdef CONFIG\_OF

+static const struct of\_device\_id edt\_ft5x06\_of\_match[] = {

+ { .compatible = "edt,edt-ft5206", },

+ { .compatible = "edt,edt-ft5306", },

+ { .compatible = "edt,edt-ft5406", },

+ { /\* sentinel \*/ }

+};

+MODULE\_DEVICE\_TABLE(of, edt\_ft5x06\_of\_match);

+#endif

+

static struct i2c\_driver edt\_ft5x06\_ts\_driver = {

.driver = {

.owner = THIS\_MODULE,

.name = "edt\_ft5x06",

+ .of\_match\_table = of\_match\_ptr(edt\_ft5x06\_of\_match),

.pm = &edt\_ft5x06\_ts\_pm\_ops,

},

.id\_table = edt\_ft5x06\_ts\_id,