MC13892

1.0

Generated by Doxygen 1.5.8

Tue Aug 25 11:45:23 2009

Contents

1 Main Page

This driver uses a combination of interrupts and hardware polling. When the driver is started the MC13892 is put in interrupt mode. In this mode the MC13892 waits

for contact between the touch plates. When a contact (touch) is sensed the MC13982 generates an interrupt. The driver receives the interrupt

- Puts the MC13892 in position mode (touch mode in MC13892 docs)
- Requests the coordinate and resistance data from the ADC
- Does some simple processing and validation of the data
- Injected good coordinates into the input framework.
- Then switches into hardware polling mode until a release event.
- A release event is denoted by a contact resistance below a programmable threshold.

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

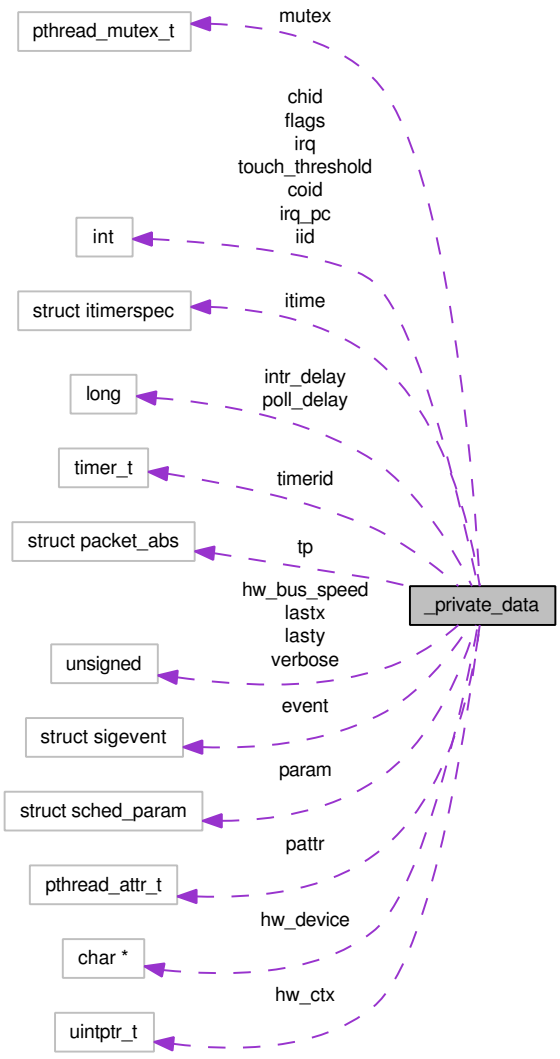
Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 _private_data Struct Reference

structure containing state of touch screen driver

Collaboration diagram for _private_data:



Data Fields

- int irq

  IRQ to attach to.

- int iid

  Interrupt ID.

- int irq_pc

  IRQ pulse code.

- int chid

  Interrupt channel ID.

- int coid

  Interrupt connection ID.

- pthread_attr_t pattr

  Interrupt thread attributes.

- struct sched_param param

  Scheduling parameter for interrupt thread.

- struct sigevent event

  Interrupt event.

- char * hw_device

  Hardware control I2C device name.

- unsigned hw_bus_speed

  I2C bus speed.

- uintptr_t hw_ctx

  Context data for control interface I2C.

- struct packet_abs tp

  touch event packet Packet sent to to input runtime for touch event.

- unsigned char verbose

  Verbose level set using multiple -v on command line (-vvv).

- int flags

  Driver state flags.

- unsigned lastx

  Last valid touch x coordinate.

- unsigned lasty

  Last valid touch y coordinate.

- pthread_mutex_t mutex
- timer_t timerid

  Mutex to manage concurrent access to hardware.

- struct itimerspec itime

  Touch relase time specification.

- long intr_delay

  Minimum interrupt delay.

- long poll_delay

  Poll delay.

- int touch_threshold

  Restance required to be considered a touch 0-1024.

4.1.1 Detailed Description

structure containing state of touch screen driver

Definition at line 80 of file touch.c.

4.1.2 Field Documentation

4.1.2.1 int irq

IRQ to attach to.

Definition at line 81 of file touch.c.

Referenced by intr_thread(), touch_init(), touch_parm(), touch_pulse(), and touch_reset().

4.1.2.2 int iid

Interrupt ID.

Definition at line 82 of file touch.c.

Referenced by intr_thread(), touch_pulse(), and touch_reset().

4.1.2.3 int irq_pc

IRQ pulse code.

Definition at line 83 of file touch.c.

Referenced by touch_init(), and touch_reset().

4.1.2.4 int chid

Interrupt channel ID.

Definition at line 85 of file touch.c.

Referenced by intr_thread(), and touch_reset().

4.1.2.5 int coid

Interrupt connection ID.

Definition at line 86 of file touch.c.

Referenced by touch_reset().

4.1.2.6 pthread_attr_t pattr

Interrupt thread attributes.

Definition at line 87 of file touch.c.

Referenced by touch_reset().

4.1.2.7 struct sched_param param [read]

Scheduling parameter for interrupt thread.

Definition at line 88 of file touch.c.

Referenced by touch_init(), touch_parm(), and touch_reset().

4.1.2.8 struct sigevent event [read]

Interrupt event.

Definition at line 89 of file touch.c.

Referenced by touch_init(), touch_parm(), and touch_reset().

4.1.2.9 char* hw_device

Hardware control I2C device name.

Definition at line 94 of file touch.c.

Referenced by touch_init(), touch_parm(), and touch_reset().

4.1.2.10 unsigned hw_bus_speed

I2C bus speed.

Definition at line 98 of file touch.c.

Referenced by touch_init(), touch_parm(), and touch_reset().

4.1.2.11 uintptr_t hw_ctx

Context data for control interface I2C.

Contains the file descriptor for the I2C device connection.

Definition at line 103 of file touch.c.

Referenced by intr_thread(), read_conversion(), set_touchscreen_mode(), start_conversion(), touch_pulse(), touch_reset(), and touch_shutdown().

4.1.2.12 struct packet_abs tp [read]

touch event packet Packet sent to to input runtime for touch event.

Definition at line 108 of file touch.c.

Referenced by intr_thread(), process_data(), and touch_pulse().

4.1.2.13 unsigned char verbose

Verbose level set using multiple -v on command line (-vvv).

Definition at line 110 of file touch.c.

Referenced by intr_thread(), process_data(), read_conversion(), set_touchscreen_mode(), touch_parm(), touch_pulse(), and touch_reset().

4.1.2.14 int flags

Driver state flags.

Definition at line 111 of file touch.c.

Referenced by touch_devctrl(), touch_init(), and touch_reset().

4.1.2.15 unsigned lastx

Last valid touch x coordinate.

Definition at line 113 of file touch.c.

Referenced by intr_thread(), process_data(), touch_init(), and touch_pulse().

4.1.2.16 unsigned lasty

Last valid touch y coordinate.

Definition at line 113 of file touch.c.

Referenced by intr_thread(), process_data(), touch_init(), and touch_pulse().

4.1.2.17 pthread_mutex_t mutex

Definition at line 116 of file touch.c.

Referenced by intr_thread(), touch_init(), and touch_pulse().

4.1.2.18 timer_t timerid

Mutex to manage concurrent access to hardware.

Touch release timer identifier

Definition at line 119 of file touch.c.

Referenced by intr_thread(), touch_pulse(), and touch_reset().

4.1.2.19 struct itimerspec itime [read]

Touch relase time specification.

Definition at line 120 of file touch.c.

Referenced by intr_thread(), and touch_pulse().

4.1.2.20 long intr_delay

Minimum interrupt delay.

Definition at line 122 of file touch.c.

Referenced by intr_thread(), touch_init(), and touch_parm().

4.1.2.21 long poll_delay

Poll delay.

The time between hardware polls, controls the event injection rate. Lager delays cause slower injection rates

Definition at line 123 of file touch.c.

Referenced by intr_thread(), touch_init(), touch_parm(), and touch_pulse().

4.1.2.22 int touch_threshold

Restance required to be considered a touch 0-1024.

Definition at line 124 of file touch.c.

Referenced by process_data(), touch_init(), and touch_parm().

The documentation for this struct was generated from the following file:

- touch.c

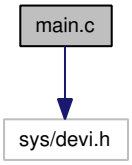
5 File Documentation

5.1 main.c File Reference

Starts MC13892 driver.

`#include <sys/devi.h>`

Include dependency graph for main.c:



Functions

- int main (int argc, char *argv[ ])
- __SRCVERSION ("$URL $Rev")

Variables

- input_module_t touch

  Touch screen input module.

- input_module_t * modules [ ]

5.1.1 Detailed Description

Starts MC13892 driver.

Definition in file main.c.

5.1.2 Function Documentation

5.1.2.1 int main (int *argc*, char * *argv*[ ])

Definition at line 37 of file main.c.

```
38 {
39     return begin(argc, argv);
40 }
```

5.1.2.2 __SRCVERSION ("$URL $Rev")

5.1.3 Variable Documentation

5.1.3.1 input_module_t touch

Touch screen input module.

We create one input_module_t structure to represent the touch screen.

Note:

If more than one are needed, i.e. in multiple bus lines; then the system will allocate a new module and copy the contents of the static one into it.

Definition at line 151 of file touch.c.

5.1.3.2 input_module_t* modules[ ]

Initial value:

```
 {
    &touch,
    NULL
}
```

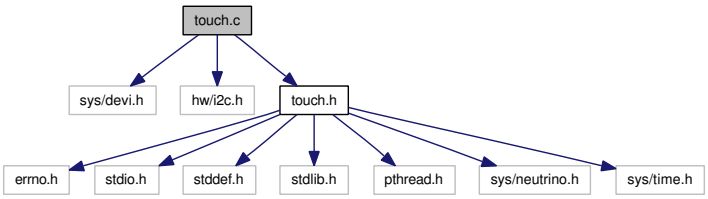
Definition at line 30 of file main.c.

5.2 touch.c File Reference

Touch screen driver for the MC13892 used on the i.MX35 PDK.

```
#include <sys/devi.h>
#include <hw/i2c.h>
#include "touch.h"
```

Include dependency graph for touch.c:



Data Structures

- struct _private_data

  structure containing state of touch screen driver

Defines

- #define CMD_PARAMETERS "i:a:b:p:D:d:t:v:"

  Command line parameters.

Typedefs

- typedef struct _private_data private_data_t

  structure containing state of touch screen driver

Functions

- static int touch_init (input_module_t *module)

  Initialization function.

- static int touch_devctrl (input_module_t *module, int event, void *ptr)

Informs input runtime about device capabilities.

- static int touch_reset (input_module_t *module)

  Reset the module.

- static int touch_pulse (message_context_t *ctp, int code, unsigned flags, void *data)

  Timer pulse handler gets called when the poll timer has expired.

- static int touch_parm (input_module_t *module,int opt,char *optarg)

  Parses the driver command line options.

- static int touch_shutdown (input_module_t *module, int delay)

  Shutdown touchscreen driver.

- static void * intr_thread (void *data)

  Interrupt handler function.

- static int i2c_read (uint32_t i2c_fd, int reg)

  I2C read register.

- static void i2c_write (uint32_t i2c_fd, int reg, int val)

  I2C write register.

- static void set_touchscreen_mode (void *data, int mode)

  Set the touch screen mode.

- static int process_data (uint16_t raw_x[2], uint16_t raw_y[2], uint16_t raw_r[2], int *x, int *y, void *data)

  Process the coordinate data.

- void start_conversion (void *data)

  Start coordinate conversion.

- void read_conversion (void *data, uint16_t x[2], uint16_t y[2], uint16_t r[2])

  Read coordinate information from MC13892 ADC.

Variables

- input_module_t touch

  Touch screen input module.

5.2.1 Detailed Description

Touch screen driver for the MC13892 used on the i.MX35 PDK.

Definition in file touch.c.

5.2.2 Define Documentation

5.2.2.1 #define CMD_PARAMETERS "i:a:b:p:D:d:t:v:"

Command line parameters.

-i irq

- IRQ for sample device (default 96).

-a device

- Sample device control interface default (/dev/i2c0).

-b speed

- Sample device control interface speed default (100000).

-p priority

- Pulse priority for the interrupt handling thread (default 21).

-D delay

- Millisecond minimum delay between interrupts (default 75)

-d delay

- Millisecond delay timer for injected events (default 100).

-t

- Touch threshold presure, resistance 0-1024 (default xx)

-v

- Verbosity, added v's means more verbosit

Definition at line 75 of file touch.c.

5.2.3 Typedef Documentation

5.2.3.1 typedef struct _private_data private_data_t

structure containing state of touch screen driver

5.2.4 Function Documentation

5.2.4.1 static int touch_init (input_module_t * *module*) [static]

Initialization function.

Called by the input driver to initialize this driver.

Parameters:

module Module to initialize, The actual touch module or a copy of it.

Returns:

0 always success

Note:

Callback specified in the input_module_t structure

Definition at line 195 of file touch.c.

References _private_data::event, FLAG_RESET, _private_data::flags, _private_data::hw_bus_speed, _private_data::hw_device, HW_POLL_TIME, INTR_DELAY, _private_data::intr_delay, _private_data::irq, _private_data::irq_pc, _private_data::lastx, _private_data::lasty, MC13892_I2C_BUS_SPEED, MC13892_I2C_DEVICE, MC13892_TOUCH_RESISTANCE_DEFAULT, _private_data::mutex, _private_data::param, _private_data::poll_delay, PULSE_PRIORITY, TOUCH_INT, _private_data::touch_threshold, and TRACE.

```
195                                                 {
196     private_data_t *dp = module->data;
197     TRACE;
198
199     if (!module->data) {
200         if (!(dp = module->data = scalloc(sizeof *dp))) {
201             return (-1);
202         }
203         ThreadCtl(_NTO_TCTL_IO, 0);
204
```

```
205         dp->flags = FLAG_RESET;
206         dp->irq = TOUCH_INT;
207         dp->irq_pc = DEVI_PULSE_ALLOC;
208         dp->hw_device = MC13892_I2C_DEVICE;
209         dp->hw_bus_speed = MC13892_I2C_BUS_SPEED;
210         dp->lastx = 0;
211         dp->lasty = 0;
212         dp->param.sched_priority = PULSE_PRIORITY;
213         dp->event.sigev_priority = dp->param.sched_priority;
214         dp->intr_delay = INTR_DELAY;
215         dp->poll_delay = HW_POLL_TIME;
216         dp->touch_threshold = MC13892_TOUCH_RESISTANCE_DEFAULT;
217         pthread_mutex_init(&dp->mutex, NULL);
218     }
219     return (0);
220 }
```

5.2.4.2 static int touch_devctrl (input_module_t * *module*, int *event*, void * *ptr*) [static]

Informs input runtime about device capabilities.

The number of buttons, number of coordinates and range of coordinates

Parameters:

module Module structure representing this instance of the driver

event Input event type

ptr Pointer to data structure to write ctrl value into.

Returns:

0 always returns success, calls exit(-1) on fail condition.

Note:

Called by input runtime.
Used by modules in an event bus line to send information further up the line to other modules (e.g. abs).

Definition at line 402 of file touch.c.

References _private_data::flags, TRACE, and TRACE_EXIT.

```
402                                                                 {
403     private_data_t *dp = module->data;
404     TRACE;
405
```

```
406     switch (event) {
407     case DEVCTL_GETDEVFLAGS:
408         *(unsigned short *) ptr = (dp->flags & FLAGS_GLOBAL);
409         break;
410     case DEVCTL_GETPTRBTNS:
411         *(unsigned long *) ptr = 1L;
412         break;
413     case DEVCTL_GETPTRCOORD:
414         *(unsigned char *) ptr = (unsigned char) 2;
415         break;
416     case DEVCTL_GETCOORDRNG: {
417         struct devctl_coord_range *range = ptr;
418 
419         range->min = 0;
420         range->max = 1024;
421         break;
422     }
423     default:
424         return (-1);
425     }
426 
427     TRACE_EXIT;
428     return (0);
429 }
```

5.2.4.3 static int touch_reset (input_module_t * *module*) [static]

Reset the module.

- Configure hardware control interface
- Create a timer to timeout touch events and inject the release events,
- Create a separate thread to handle the IRQ's from the touch controller.

Parameters:

module Module structure representing this instance of the driver

Returns:

0 always returns success, calls exit(-1) on fail condition.

Note:

called by input runtime.

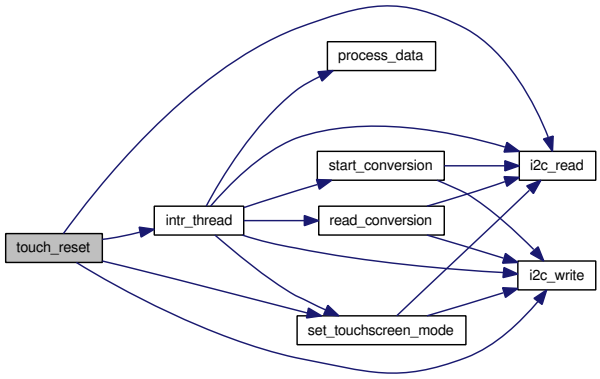
Definition at line 288 of file touch.c.

References _private_data::chid, _private_data::coid, _private_data::event, FLAG_INIT, _private_data::flags, _private_data::hw_bus_speed, _private_data::hw_ctx, _private_data::hw_device, i2c_read(), i2c_write(), _private_data::iid, INT_MASK_0_REG, INT_MASK_REG, INT_STATUS_0_REG, INTERRUPT_MODE, intr_thread(), _private_data::irq, _private_data::irq_pc, _private_data::param, _private_data::pattr, set_touchscreen_mode(), TIMER_REG, _private_data::timerid, TRACE, TRACE_EXIT, and _private_data::verbose.

```
288                                                 {
289     private_data_t *dp = module->data;
290     uint32_t buf = 0;
291     TRACE;
292 
293     if ((dp->flags & FLAG_INIT) == 0) {
294         int status;
295         // Enable IO capability.
296         if (ThreadCtl(_NTO_TCTL_IO, NULL) == -1) {
297             perror("ThreadCtl: ");
298             exit(EXIT_FAILURE);
299         }
300         dp->hw_ctx = open(dp->hw_device, O_RDWR);
301         if (dp->hw_ctx == -1) {
302             printf("Failed to open I2C device %s for MC13892", dp->hw_device);
303             exit(-1);
304         }
305 
306         if (status = devctl(dp->hw_ctx, DCMD_I2C_SET_BUS_SPEED,
307                 &(dp->hw_bus_speed), sizeof(dp->hw_bus_speed), NULL)) {
308             errno = status;
309             perror("devctl(BUS_SPEED)");
310         }
311 
312         /* Clear MC13892 interrupt status
313          * Bits 0 : ADCDONE
314          * Bits 2 : TSI touch screen wake upe
315          */
316         i2c_write(dp->hw_ctx, INT_STATUS_0_REG, (1 << 0) | (1 << 2));
317 
318         buf = i2c_read(dp->hw_ctx, INT_MASK_0_REG);
319         if (dp->verbose >= 3) {
320             fprintf(stderr, "Mask Reg: %x\n", buf);
321         }
322         /* clear TSI bit 4 in MC13892 mask register */
323         buf = ~0x4;
324         i2c_write(dp->hw_ctx, INT_MASK_REG, buf);
325 
326         /* Setup the PLLX timer
327          In switcher register 4 address 28
328          bit 18 in PLLEN set 1-enable PLL
329          Bits 19 PLL0  set 1
330          Bits 20 PLL0  set 1
331          Bits 21 PLL2  set 1
332          Frequency is 3440640Hz the maximum
333          */
334         buf = i2c_read(dp->hw_ctx, TIMER_REG);
335         buf &= ~((1<<18) | (1<<19) |  (1<<20) | (1<<21));
```

```
336         buf |= ((1<<18) | (1<<19) |  (1<<20) | (1<<21));
337         i2c_write(dp->hw_ctx, TIMER_REG, buf);
338         /* Put PMIC touch ADC into interrupt mode (required for pen down interrupt) */
339         set_touchscreen_mode(module, INTERRUPT_MODE);
340 
341         /* Create touch release timer */
342         dp->timerid = devi_register_timer(module, 15, &dp->irq_pc, NULL);
343 
344         /* Setup the interrupt handler thread */
345         if ((dp->chid = ChannelCreate(_NTO_CHF_DISCONNECT | _NTO_CHF_UNBLOCK))
346                 == -1) {
347             perror("Error: ChannelCreate");
348             exit(-1);
349         }
350 
351         if ((dp->coid = ConnectAttach(0, 0, dp->chid, _NTO_SIDE_CHANNEL, 0))
352                 == -1) {
353             perror("Error: ConnectAttach");
354             exit(-1);
355         }
356 
357         pthread_attr_init(&dp->pattr);
358         pthread_attr_setschedpolicy(&dp->pattr, SCHED_RR);
359         pthread_attr_setschedparam(&dp->pattr, &dp->param);
360         pthread_attr_setinheritsched(&dp->pattr, PTHREAD_EXPLICIT_SCHED);
361         pthread_attr_setdetachstate(&dp->pattr, PTHREAD_CREATE_DETACHED);
362         pthread_attr_setstacksize(&dp->pattr, 4096);
363 
364         dp->event.sigev_notify = SIGEV_PULSE;
365         dp->event.sigev_coid = dp->coid;
366         dp->event.sigev_code = 1;
367 
368         /* Attach interrupt. */
369         if (dp->verbose >= 3) {
370             fprintf(stderr, "Attaching to interrupt %d\n", dp->irq);
371         }
372         if ((dp->iid = InterruptAttachEvent(dp->irq, &dp->event,
373                 _NTO_INTR_FLAGS_TRK_MSK)) == -1) {
374             perror("Error: InterruptAttachEvent");
375             exit(-1);
376         }
377 
378         /* Create interrupt handler thread */
379         if (pthread_create(NULL, &dp->pattr, (void *) intr_thread, module)) {
380             perror("Error: pthread_create");
381             exit(-1);
382         }
383 
384         dp->flags |= FLAG_INIT;
385     }
386 
387     TRACE_EXIT;
388     return (0);
389 }
```

Here is the call graph for this function:



5.2.4.4 static int touch_pulse (message_context_t * *ctp*, int *code*, unsigned *flags*, void * *data*) [static]

Timer pulse handler gets called when the poll timer has expired.

Put MC13892 in Possition mode, reads, process, and injects coordinate data into input runtime.

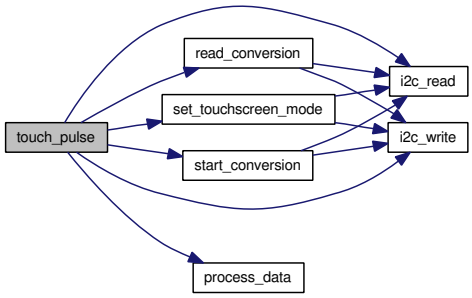
Definition at line 888 of file touch.c.

References _private_data::hw_ctx, i2c_read(), i2c_write(), _private_data::iid, INT_MASK_0_REG, INT_MASK_REG, INT_STATUS_0_REG, INTERRUPT_MODE, _private_data::irq, _private_data::itime, _private_data::lastx, _private_data::lasty, MC13892_ADCDONE_BIT, MC13892_TSI_BIT, _private_data::mutex, _private_data::poll_delay, POSITION_MODE, process_data(), read_conversion(), set_touchscreen_mode(), start_conversion(), _private_data::timerid, _private_data::tp, TRACE, and _private_data::verbose.

```
889                     {
890     input_module_t *module = (input_module_t *) data;
891     input_module_t *up = module->up;
892     private_data_t *dp = module->data;
893     uint16_t raw_x[2], raw_y[2], raw_r[2];
894     int x, y;
895     uint32_t buf = 0;
896     TRACE;
897 
898     InterruptMask(dp->irq, dp->iid);
899     pthread_mutex_lock(&dp->mutex);
900 
```

```
901     /* Put touchscreen into Position Mode */
902     set_touchscreen_mode(data, POSITION_MODE);
903
904     /* Start the conversion */
905     start_conversion(data);
906
907     /* Read the data from the controller */
908     read_conversion(data, raw_x, raw_y, raw_r);
909
910     /* Process Data */
911     process_data(raw_x, raw_y, raw_r, &x, &y, data);
912
913     if (dp->verbose >= 1) {
914         fprintf(stderr, "X:%d Y:%d State: %s\n", dp->tp.x, dp->tp.y,
915                 (dp->tp.buttons == 0L) ? "Released" : "Touched");
916     }
917
918     dp->lastx = dp->tp.x;
919     dp->lasty = dp->tp.y;
920     /* Emit the data to the upper layers */
921     clk_get(&dp->tp.timestamp);
922     (up->input)(up, 1, &dp->tp);
923
924     if (dp->tp.buttons != 0L) {
925         /* restart the hardware poll timer */
926         dp->itime.it_value.tv_sec = 0;
927         dp->itime.it_value.tv_nsec = dp->poll_delay;
928         dp->itime.it_interval.tv_sec = 0;
929         dp->itime.it_interval.tv_nsec = 0;
930
931         /* Set touch release timer */
932         timer_settime(dp->timerid, 0, &dp->itime, NULL);
933     }
934     /* clear status, just in case */
935     i2c_write(dp->hw_ctx, INT_STATUS_0_REG, MC13892_TSI_BIT | MC13892_ADCDONE_BIT);
936     /* re-enable touchscreen interrupt */
937     buf = i2c_read(dp->hw_ctx, INT_MASK_0_REG);
938     buf &= ~MC13892_TSI_BIT;
939     i2c_write(dp->hw_ctx, INT_MASK_REG, buf);
940     set_touchscreen_mode (module, INTERRUPT_MODE);
941     pthread_mutex_unlock(&dp->mutex);
942     InterruptUnmask(dp->irq, dp->iid);
943
944     return (0);
945 }
```

Here is the call graph for this function:



5.2.4.5 static int touch_parm (input_module_t * *module*, int *opt*, char * *optarg*) [static]

Parses the driver command line options.

Called once for each option.

Note:

called by input runtime.

Returns:

0 always successful

Parameters:

module Module structure representing this instance of the driver

opt Options to parse

optarg Argument for the option

Definition at line 227 of file touch.c.

References DEBUG_CMD, _private_data::event, _private_data::hw_bus_speed, _private_data::hw_device, _private_data::intr_delay, _private_data::irq, _private_data::param, _private_data::poll_delay, _private_data::touch_threshold, TRACE, and _private_data::verbose.

```
231       {
232     private_data_t *dp = module->data;
233     TRACE;
```

```
234
235     switch (opt) {
236        /* verbosity */
237     case 'v':
238         dp->verbose++;
239         break;
240         /* interrupt */
241     case 'i':
242         dp->irq = atoi(optarg);
243         break;
244         /* Hardware control device name */
245     case 'a':
246         dp->hw_device = optarg;
247         break;
248         /* Hardware control device speed */
249     case 'b':
250         dp->hw_bus_speed = atol(optarg);
251         break;
252         /* priority */
253     case 'p':
254         dp->param.sched_priority = atoi(optarg);
255         dp->event.sigev_priority = dp->param.sched_priority;
256         break;
257         /* interrupt delay, used to slow interrupts */
258     case 'D':
259         dp->intr_delay = atoi (optarg);
260         DEBUG_CMD(slogf(99,1,"Interrupt delay %d", dp->intr_delay);)
261         break;
262         /* delay time for hardware poll and therefor event injection */
263     case 'd':
264         dp->poll_delay = (atol(optarg)) * 100000; /* Convert to nsecs */
265         DEBUG_CMD(slogf(99,1,"Poll delay %d", dp->poll_delay);)
266         break;
267         /* touch resistance threshold */
268     case 't':
269         dp->touch_threshold = atoi(optarg);
270         break;
271     default:
272         fprintf(stderr, "Unknown option -%c", opt);
273         break;
274     }
275
276     return (0); /** @return  0 always successful */
277 }
```

5.2.4.6 static int touch_shutdown (input_module_t * *module*, int *delay*) [static]

Shutdown touchscreen driver.

Definition at line 1015 of file touch.c.

References _private_data::hw_ctx, and TRACE.

```
1015                                                                    {
1016     private_data_t *dp = module->data;
1017     TRACE;
1018
1019     close(dp->hw_ctx);
1020     free(module->data);
1021
1022     return (0);
1023 }
```

5.2.4.7 static void * intr_thread (void * *data*) [static]

Interrupt handler function.

This code is run by the interrupt handler thread. It waits on a pulse generated by the interrupt and then requests the X and Y coordinates from the touch controller (MC13892).

Parameters:

data device context (module_t)

Note:

Once the data has been fetched, a timer is started to poll for a more points or a release event.

Definition at line 656 of file touch.c.

References _private_data::chid, DEBUG_CMD, _private_data::hw_ctx, i2c_read(), i2c_write(), _private_data::iid, INT_MASK_0_REG, INT_MASK_REG, INT_STATUS_0_REG, INTERRUPT_MODE, _private_data::intr_delay, _private_data::irq, _private_data::itime, _private_data::lastx, _private_data::lasty, MC13892_ADCDONE_BIT, MC13892_TSI_BIT, _private_data::mutex, _private_data::poll_delay, POSITION_MODE, process_data(), PULSE_CODE, read_conversion(), set_touchscreen_mode(), start_conversion(), _private_data::timerid, _private_data::tp, TRACE, and _private_data::verbose.

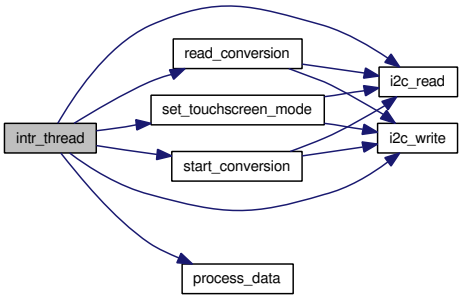
Referenced by touch_reset().

```
656                         {
657     input_module_t *module = (input_module_t *) data;
658     private_data_t *dp = module->data;
659     input_module_t *up = module->up;
660     struct _pulse pulse;
661     iov_t iov;
662     int rcvid;
663     uint32_t buf = 0;
```

```
664     uint16_t raw_x[2], raw_y[2], raw_r[2];
665     int x, y;
666     TRACE;
667 
668     SETIOV(&iov, &pulse, sizeof(pulse));
669 
670     while (1) {
671         if ((rcvid = MsgReceivev(dp->chid, &iov, 1, NULL)) == -1) {
672             if (errno == ESRCH) {
673                 pthread_exit(NULL);
674             }
675             continue;
676         }
677 
678         switch (pulse.code) {
679         case PULSE_CODE:
680             pthread_mutex_lock(&dp->mutex);
681             DEBUG_CMD(slogf(99,1,"Got interrupt IRQ status reg %x", i2c_read(dp->hw_ctx, INT_STA
682             if (dp->verbose >= 1) {
683                 printf("Got Interrupt\n");
684             }
685 
686             /* Stop timer */
687             dp->itime.it_value.tv_sec = 0;
688             dp->itime.it_value.tv_nsec = 0;
689             dp->itime.it_interval.tv_sec = 0;
690             dp->itime.it_interval.tv_nsec = 0;
691 
692             /* Set touch release timer */
693             timer_settime(dp->timerid, 0, &dp->itime, NULL);
694 
695             /* Clear interrupt and Unmask */
696             i2c_write(dp->hw_ctx, INT_STATUS_0_REG,
697                     MC13892_TSI_BIT);
698 
699             buf = i2c_read(dp->hw_ctx, INT_MASK_0_REG);
700             buf |= MC13892_TSI_BIT;
701             i2c_write(dp->hw_ctx, INT_MASK_0_REG, buf);
702 
703             /* Put touchscreen into Position Mode */
704             set_touchscreen_mode(data, POSITION_MODE);
705 
706             /* Start the conversion */
707             start_conversion(data);
708 
709             /* Read the data from the controller */
710             read_conversion(data, raw_x, raw_y, raw_r);
711 
712             /* Process Data */
713             process_data(raw_x, raw_y, raw_r, &x, &y, data);
714 
715             dp->tp.x = x;
716             dp->tp.y = y;
717 
718             if (dp->verbose >= 1) {
719                 fprintf(stderr, "X:%d Y:%d State: %s\n", dp->tp.x, dp->tp.y,
720                         (dp->tp.buttons == 0L) ? "Released" : "Touched");
```

```
721             }
722
723             /* Emit the data to the upper layers */
724             clk_get(&dp->tp.timestamp);
725             (up->input)(up, 1, &dp->tp);
726
727             dp->lastx = dp->tp.x;
728             dp->lasty = dp->tp.y;
729
730             if (dp->tp.buttons != 0L) {
731                 /* start the hardware poll timer */
732                 dp->itime.it_value.tv_sec = 0;
733                 dp->itime.it_value.tv_nsec = dp->poll_delay;
734                 dp->itime.it_interval.tv_sec = 0;
735                 dp->itime.it_interval.tv_nsec = 0;
736                 timer_settime(dp->timerid, 0, &dp->itime, NULL);
737             }
738
739             /* clear status, just in case */
740             i2c_write(dp->hw_ctx, INT_STATUS_0_REG, MC13892_TSI_BIT | MC13892_ADCDONE_BIT);
741             /* re-enable touchscreen interrupt */
742             buf = i2c_read(dp->hw_ctx, INT_MASK_0_REG);
743             buf &= ~MC13892_TSI_BIT;
744             i2c_write(dp->hw_ctx, INT_MASK_REG, buf);
745
746             set_touchscreen_mode(module, INTERRUPT_MODE);
747             InterruptUnmask(dp->irq, dp->iid);
748             pthread_mutex_unlock(&dp->mutex);
749             /* slow interrupts */
750             delay (dp->intr_delay);
751
752             break;
753         default:
754             if (rcvid) {
755                 MsgReplyv(rcvid, ENOTSUP, &iov, 1);
756             }
757             break;
758         }
759     }
760 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

5.2.4.8 static int i2c_read (uint32_t *i2c_fd*, int *reg*) [static]

I2C read register.

Read MC13892 register over I2C.

Parameters:

i2c_fd I2C file descriptor

reg Register number to read

Returns:

Value read from register

Definition at line 954 of file touch.c.

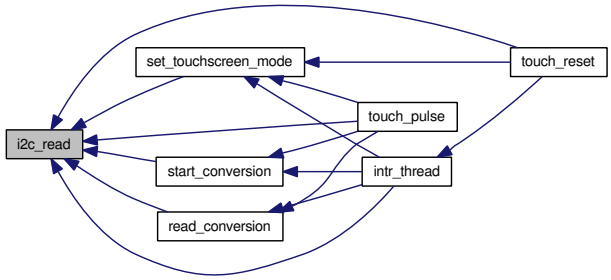
References DEBUG_CMD, and MC13892_I2C_ADDR.

Referenced by intr_thread(), read_conversion(), set_touchscreen_mode(), start_conversion(), touch_pulse(), and touch_reset().

```
954                                                 {
955     struct {
956         i2c_sendrecv_t hdr;
957         unsigned char bytes[3];
958     } msg;
959     int status;
960     int bytes;
961     int ret;
962     //TRACE;
963 
964     msg.hdr.slave.addr = MC13892_I2C_ADDR;
965     msg.hdr.slave.fmt = I2C_ADDRFMT_7BIT;
966     msg.hdr.send_len = 1;
967     msg.hdr.recv_len = 3;
968     msg.hdr.stop = 1;
969     msg.bytes[0] = (reg & 0x3f);
970     if (status = devctl(i2c_fd, DCMD_I2C_SENDRECV, &msg, sizeof(msg), &bytes)) {
971         errno = status;
972         perror("SENDRECV");
973 
974     }
```

```
975     ret = msg.bytes[0] << 16 | (msg.bytes[1] << 8) | (msg.bytes[2]);
976     DEBUG_CMD (printf ("read reg %d val %x\n", reg, ret));
977     return ret;
978 }
```

Here is the caller graph for this function:



5.2.4.9 static void i2c_write (uint32_t *i2c_fd*, int *reg*, int *val*) [static]

I2C write register.

Read MC13892 register over I2C.

Parameters:

i2c_fd I2C file descriptor

reg Register number to write

val Value to write

Definition at line 987 of file touch.c.

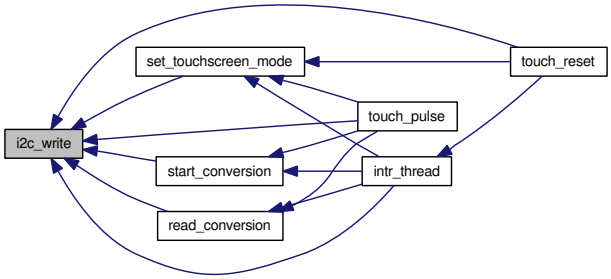
References DEBUG_CMD, and MC13892_I2C_ADDR.

Referenced by intr_thread(), read_conversion(), set_touchscreen_mode(), start_conversion(), touch_pulse(), and touch_reset().

```
987                                                     {
988     int status;
989     int bytes;
990     struct {
991         i2c_send_t hdr;
992         unsigned char bytes[4];
993     } msg;
994     //TRACE;
995     DEBUG_CMD (printf ("write reg %d val %x\n", reg, val));
```

```
996
997     msg.hdr.slave.addr = MC13892_I2C_ADDR;
998     msg.hdr.slave.fmt = I2C_ADDRFMT_7BIT;
999     msg.hdr.len = sizeof(msg.bytes);
1000     msg.hdr.stop = 1;
1001     msg.bytes[0] = (reg & 0x3F);
1002     msg.bytes[1] = ((val >> 16) & 0xFF);
1003     msg.bytes[2] = ((val >> 8) & 0xFF);
1004     msg.bytes[3] = ((val >> 0) & 0xFF);
1005
1006     if (status = devctl(i2c_fd, DCMD_I2C_SEND, &msg, sizeof(msg), &bytes)) {
1007         errno = status;
1008         perror("SEND");
1009     }
1010 }
```

Here is the caller graph for this function:



5.2.4.10 static void set_touchscreen_mode (void * *data*, int *mode*) [static]

Set the touch screen mode.

Sets the touch screen mode for the MC13892.

- Interrupt mode, the MC13892 waits for a touch event i.e. generates an interrupt when the plates make contact.
- Position mode, the MC13892 is reading positional data from ADC.
- Inactive mode, the MC13892 touch screen inactive touchscreen input can be used for general purpose ADC input.

Parameters:

data device context (module_t)
mode Mode to set (INTERRUPT/POSITION/INACTIVE)

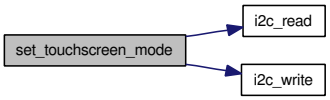
Definition at line 443 of file touch.c.

References ADC0_REG, DEBUG_MSG, _private_data::hw_ctx, i2c_read(), i2c_write(), INTERRUPT_MODE, MC13892_INACTIVE_MODE, MC13892_-INTERRUPT_MODE, MC13892_MODE_MASK, MC13892_TOUCHSCREEN_-MODE, POSITION_MODE, TRACE, and _private_data::verbose.

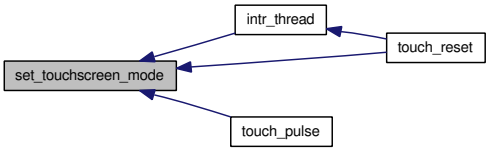
Referenced by intr_thread(), touch_pulse(), and touch_reset().

```
443                                                     {
444     input_module_t *module = (input_module_t *) data;
445     private_data_t *dp = module->data;
446     uint32_t buf;
447     TRACE;
448     buf = i2c_read(dp->hw_ctx, ADC0_REG);
449     buf &= ~MC13892_MODE_MASK;
450 
451     if (mode == POSITION_MODE) {
452         DEBUG_MSG("POSITION_MODE");
453         buf = MC13892_TOUCHSCREEN_MODE;
454     } else if (mode == INTERRUPT_MODE) {
455         DEBUG_MSG("INTERRUPT_MODE");
456         if (dp->verbose) {
457             fprintf(stderr, "INTERRUPT MODE!!!\n");
458         }
459         buf = MC13892_INTERRUPT_MODE;
460     } else {
461         DEBUG_MSG("INACTIVE_MODE");
462         buf = MC13892_INACTIVE_MODE;
463     }
464 
465     i2c_write(dp->hw_ctx, ADC0_REG, buf);
466 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.4.11 int process_data (uint16_t *raw_x*[2], uint16_t *raw_y*[2], uint16_t *raw_r*[2], int * *x*, int * *y*, void * *data*) [static]

Process the coordinate data.

- check for pen up event i.e. resistance below pen threshold
- Compare coordinate reject those with too great a spread
- Use Hysteresis to reject invalid data.

Parameters:

data device context (module_t)
raw_x Array of 2 for x coordinates
raw_y Array of 2 for y coordinates
raw_r Array of 2 for resistance
x Pointer to x coordinate for processed coordinate
y Pointer to y coordinate for processed coordinate

Note:

This code is called from the interrupt handler thread.

Returns:

0-success 1-release event sent -1 - invalid date ignore

Note:

Uses Hysteres analasys to validate data. According to Wikipedia: In a system with hysteresis, this is not possible; there is no way to predict the output without knowing the system's current state, and there is no way to know the system's state without looking at the history of the input. This means that it is necessary to know the path that the input followed before it reached its current value.

Definition at line 781 of file touch.c.

References ABS, DEBUG_MSG, DELTA_X_COORD_VARIANCE, DELTA_Y_COORD_VARIANCE, _private_data::lastx, _private_data::lasty, _private_data::touch_threshold, _private_data::tp, TRACE, and _private_data::verbose.

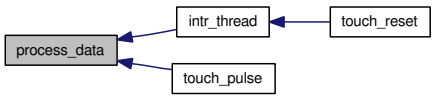
Referenced by intr_thread(), and touch_pulse().

```
782                                     {
783     input_module_t *module = (input_module_t *) data;
784     private_data_t *dp = module->data;
785     int i = 0;
```

```
786     uint16_t delta;
787     static uint16_t hys_x[2];
788     static uint16_t hys_y[2];
789     static uint32_t hysIndex;
790     TRACE;
791
792     // Check for pen up condition
793     for (; i < 2; i++) {
794           if (raw_r[i] > dp->touch_threshold) {
795             // Got a release
796             DEBUG_MSG("resistance below threshold");
797
798             *x = dp->lastx;
799             *y = dp->lasty;
800             dp->tp.buttons = 0L;
801
802             if (dp->verbose > 2) {
803                 fprintf(stderr,
804                         "resistance below threshold injecting a Release.\n");
805             }
806             return (1);
807         }
808     }
809
810     // Calculate absolute differences between x-coordinate samples
811     delta = ABS (raw_x[0] - raw_x[1]);
812
813     // Reject the samples if the spread is too large
814     if ((delta > DELTA_X_COORD_VARIANCE)) {
815         // Data is invalid
816         if (dp->verbose > 2) {
817             fprintf(stderr, "Data is invalid X Spread is too large.\n");
818         }
819         return (-1);
820     }
821
822     *x = raw_x[0] + raw_x[1];
823
824     // Calculate absolute differences between y-coordinate samples
825     delta = ABS (raw_y[0] - raw_y[1]);
826
827     if ((delta > DELTA_Y_COORD_VARIANCE)) {
828         // Data is invalid
829         if (dp->verbose > 2) {
830             fprintf(stderr, "Data is invalid Y spread is too large.\n");
831         }
832         return (-1);
833     }
834     *y = raw_y[0] + raw_y[1];
835
836     if (!dp->tp.buttons) {
837         // Prime the hysteresis buffers with average of two
838         // best samples from ADC
839         *x = *x >> 1;
840         *y = *y >> 1;
841         hys_x[0] = hys_x[1] = *x;
842         hys_y[0] = hys_y[1] = *y;
```

```
843     } else {
844         // Implement noise rejection since transition to pen up
845         // condition often gives us a spike in samples
846         if (ABS (*x - (hys_x[0] + hys_x[1])) > (DELTA_X_COORD_VARIANCE * 8)) {
847             // Data is invalid
848             if (dp->verbose > 2) {
849                 fprintf(stderr,
850                         "Data is invalid X Hystersis data is too great variance.\n");
851             }
852 
853             return (-1);
854         }
855 
856         if (ABS (*y - (hys_y[0] + hys_y[1])) > (DELTA_Y_COORD_VARIANCE * 8)) {
857             // Data is invalid
858             if (dp->verbose > 2) {
859                 fprintf(stderr,
860                         "Data is invalid Y Hystersis data is too great variance.\n");
861             }
862             return (-1);
863         }
864 
865         // Average two best samples from ADC with samples
866         // from hysteresis buffer
867         *x = (hys_x[0] + hys_x[1] + *x) >> 2;
868         *y = (hys_y[0] + hys_y[1] + *y) >> 2;
869 
870         // Replace an entry in hysteresis buffer
871         hys_x[hysIndex & 0x1] = *x;
872         hys_y[hysIndex & 0x1] = *y;
873 
874         hysIndex++;
875     }
876 
877     dp->tp.buttons = _POINTER_BUTTON_LEFT;
878 
879     return (0);
880 }
```

Here is the caller graph for this function:



5.2.4.12 void start_conversion (void * *data*)

Start coordinate conversion.

- Setup ADC for touchscreen conversion.
- Set auto increment mode
- Start ADC convertion
- Loop Waiting for conversion to finish (max 13ms)

Called in Position mode

Parameters:

data device context (module_t)

Note:

called by interrupt handler thread.

Definition at line 479 of file touch.c.

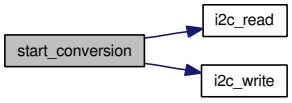
References ADC0_REG, ADC1_REG, ADCDONE1, ADCDONE1_MASK, _private_data::hw_ctx, i2c_read(), i2c_write(), INT_STATUS_0_REG, and TRACE.

Referenced by intr_thread(), and touch_pulse().

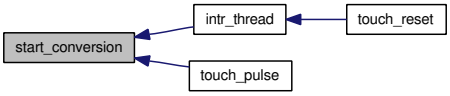
```
479                                     {
480     input_module_t *module = (input_module_t *) data;
481     private_data_t *dp = module->data;
482     uint32_t buf = 0;
483     int i;
484     TRACE;
485 
486     /*
487      * Setup ADC for touchsreen conversion.
488      * Reg ADC1
489      * bit 0 ADEN    - 1 enable ADC
490      * bit 1 RAND    - 0?
491      * bit 3 ADSEL   - 1 set to touch screen
492      * bit 11-18 AT0 - 0x01 delay conversion (default from 13783)
493      * bit 19 ATOX   - 0 delay before first only
494      * bit 21 ADTRIG - 0 Ignore ADTRIG
495      */
496     buf = i2c_read(dp->hw_ctx, ADC1_REG);
497     buf &= ~((1 << 21) | (0 << 19) | (0x0f << 11) | (1 << 3) | (0 << 1) | (1
498             << 0));
499     buf |= (0 << 21) | (0 << 19) | (0x01 << 11) | (1 << 3) | (0 << 1)
500             | (1 << 0);
501     i2c_write(dp->hw_ctx, ADC1_REG, buf);
502 
503     // increment mode.
504     /*
505      * Reg ADC0
506      * bit 10 TSREF   - 1 Enable touch screen reference
507      * bit 16 ADINC1  - 1 Enable read auto increment ADA1
```

```
508      * bit 17 ADINC2  - 1 Enable read auto increment ADA2
509      */
510     buf = i2c_read(dp->hw_ctx, ADC0_REG);
511     buf &= ~((1 << 10) | (1 << 16) | (1 << 17));
512     buf |= ((1 << 10) | (1 << 16) | (1 << 17));
513     i2c_write(dp->hw_ctx, ADC0_REG, buf);
514
515     /*
516      * Set auto increment start values
517      * Reg ADC1
518      * bit 5-7  ADA1 - 0x0  Start value
519      * bit 8-10 ADA2 - 0x0  Start value
520      */
521     buf = i2c_read(dp->hw_ctx, ADC1_REG);
522     buf &= ~((0x7 << 5) | (0x7 << 8));
523     buf |= ((0x0 << 5) | (0x0 << 8));
524     i2c_write(dp->hw_ctx, ADC1_REG, buf);
525
526     // Start the ADC conversion
527     /*
528      * Reg ADC1
529      * bit 20 ASC - 1 start conversion
530      */
531     buf = i2c_read(dp->hw_ctx, ADC1_REG);
532     buf &= ~(1 << 20);
533     buf |= (1 << 20);
534     i2c_write(dp->hw_ctx, ADC1_REG, buf);
535
536     for (i = 0; i < 13; i++) {
537         buf = i2c_read(dp->hw_ctx, INT_STATUS_0_REG);
538
539         if (buf & ADCDONE1) {
540             // Clear ADCDONE1, unmask and continue
541             buf |= ~ADCDONE1;
542             i2c_write(dp->hw_ctx, INT_STATUS_0_REG, buf);
543             buf = i2c_read(dp->hw_ctx, 1);
544             buf &= ADCDONE1_MASK;
545             break;
546         }
547         delay(1);
548     }
549
550     if (i == 13) {
551         fprintf(stderr, "Limit Reached conversion did not complete.\n");
552         fprintf(stderr, "Interrupt Reg: %x\n", i2c_read(dp->hw_ctx,
553                 INT_STATUS_0_REG));
554     }
555 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.4.13 void read_conversion (void * *data*, uint16_t *x*[2], uint16_t *y*[2], uint16_t *r*[2])

Read coordinate information from MC13892 ADC.

In order to reduce the interrupt rate and to allow for easier noise rejection, the touch screen readings are repeated in the readout sequence. ADC Conversion Signals sampled Readout Address (*) 0 X position 000 1 X position 001 2 Dummy 010 3 Y position 011 4 Y position 100 5 Dummy 101 6 Contact resistance 110 7 Contact resistance 111

Parameters:

data device context (module_t)

x Array of 2 for x coordinates

y Array of 2 for y coordinates

r Array of 2 for resisiance

Definition at line 577 of file touch.c.

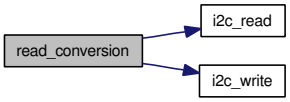
References ADC1_REG, ADC2_REG, CSP_BITFEXT, DEBUG_CMD, _private_data::hw_ctx, i2c_read(), i2c_write(), TRACE, and _private_data::verbose.

Referenced by intr_thread(), and touch_pulse().

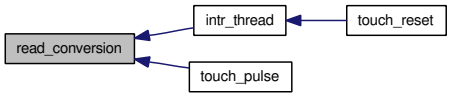
```
578                         {
579     input_module_t *module = (input_module_t *) data;
580     private_data_t *dp = module->data;
581     uint32_t buf = 0, raw_data = 0;
582     uint16_t raw_x[4];
583     uint16_t raw_y[4];
584     uint16_t raw_r[4];
585     TRACE;
586 
587     /* Read x twice */
588     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
589     raw_x[0] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
590     raw_x[1] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
591 
592     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
```

```
593     raw_x[2] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
594     raw_x[3] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
595
596     /* dummy read */
597     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
598
599     /* Read y twice */
600     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
601     raw_y[0] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
602     raw_y[1] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
603     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
604     raw_y[2] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
605     raw_y[3] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
606
607     /* dummy read */
608     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
609
610     /* Contact resistance twice */
611     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
612     raw_r[0] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
613     raw_r[1] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
614     raw_data = i2c_read(dp->hw_ctx, ADC2_REG);
615     raw_r[2] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD1);
616     raw_r[3] = CSP_BITFEXT (raw_data, MC13892_ADC2_ADD2);
617
618     /*
619      * Disable the ADC
620      * Reg ADC1
621      * bit 0 ADEN - 0 Disable ADC
622      */
623     buf = i2c_read(dp->hw_ctx, ADC1_REG);
624     buf &= ~(1 << 0);
625     buf |= (0 << 0);
626     i2c_write(dp->hw_ctx, ADC1_REG, buf);
627
628     DEBUG_CMD(slogf(99,1,"read_conversion x:%d x:%d x:%d x:%d y:%d y:%d y:%d y:%d r:%d r:%d r:%d
629     /* average values between ADC channels in the same sample, values mostly equal so this might
630     x[0] =(raw_x[0] + raw_x[1])/2;
631     x[1] =(raw_x[2] + raw_x[3])/2;
632     y[0] =(raw_y[0] + raw_y[1])/2;
633     y[1] =(raw_y[2] + raw_y[3])/2;
634     r[0] =(raw_r[0] + raw_r[1])/2;
635     r[1] =(raw_r[2] + raw_r[3])/2;
636     DEBUG_CMD(slogf(99,1,"read_conversion x:%d x:%d y:%d y:%d r:%d r:%d\n", x[0], x[1], y[0], y[
637
638     if (dp->verbose >= 1) {
639         printf("read_conversion x:%d x:%d y:%d y:%d r:%d r:%d\n", raw_x[0],
640                 raw_x[1], raw_y[0], raw_y[1], raw_r[0], raw_r[1]);
641     }
642
643 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



5.2.5 Variable Documentation

5.2.5.1 input_module_t touch

Initial value:

```
{
    NULL,
    NULL,
    NULL,
    0,
    DEVI_CLASS_ABS | DEVI_MODULE_TYPE_PROTO | DEVI_MODULE_TYPE_DEVICE,

    "touch",
    __DATE__,
    CMD_PARAMETERS,
    NULL,
    touch_init,
    touch_reset,
    NULL,
    NULL,
    touch_pulse,
    touch_parm,
    touch_devctrl,
    touch_shutdown
}
```

Touch screen input module.

We create one input_module_t structure to represent the touch screen.

Note:

If more than one are needed, i.e. in multiple bus lines; then the system will allocate a new module and copy the contents of the static one into it.

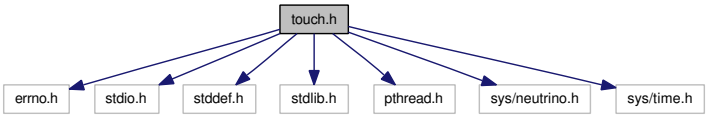
Definition at line 151 of file touch.c.

5.3 touch.h File Reference

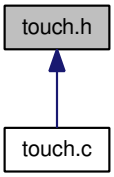
Touch screen driver for the MC13892 used on the i.MX35 PDK.

```
#include <errno.h>
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <pthread.h>
#include <sys/neutrino.h>
#include <sys/time.h>
```

Include dependency graph for touch.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define TRACE

  Trace function execution.

- #define TRACE_ENTER

  Trace function enter execution.

- #define TRACE_EXIT

  Trace function exit execution.

- #define DEBUG_MSG(x)

  Debug message.

- #define DEBUG_CMD(x)

  Debug command.

- #define TOUCH_INT 96

  Touch sreen interrupt.

- #define INACTIVE_MODE 0

  Touch sreen inactive mode MC13892 is neither waiting for a touch event or calculatinig a position.

- #define INTERRUPT_MODE 1

  Touch screen interrupt mode MC13892 is waiting for a touch event.

- #define POSITION_MODE 2

  Touch screen position mode.

- #define DELTA_X_COORD_VARIANCE 24

  Maximum allowed variance in the X coordinate samples.

- #define DELTA_Y_COORD_VARIANCE 24

  Maximum allowed variance in the X coordinate samples.

- #define ABS(x) ((x) >= 0 ? (x) : (-(x)))
- #define FLAG_INIT 0x1000

  Driver state initialize.

- #define FLAG_RESET 0x2000

  Driver state reset.

- #define INTR_DELAY 75

  Default minimum time between interrupt.

- #define HW_POLL_TIME 100

  Default time between hardware polls, this is default the rate of event injection.

- #define PULSE_PRIORITY 21

  Interrupt plus priority.

- #define PULSE_CODE 1

Interrupt pulse code.

- #define MC13892_I2C_DEVICE "/dev/i2c0"

  Default I2C address device name.

- #define MC13892_I2C_ADDR 8

  I2C address of MC13892.

- #define MC13892_I2C_BUS_SPEED 100000

  I2C Default bus speed.

- #define MC13892_TOUCH_RESISTANCE_DEFAULT 800

  Touch screen minimum resistance default.

- #define MC13892_MODE_MASK (1<<15|1<<14|1<<13|1<<12)

  Mask for MS13892 Mode.

- #define MC13892_INTERRUPT_MODE (1<<15|0<<14|0<<13|1<<12)

  Interrupt mode.

- #define MC13892_TOUCHSCREEN_MODE (1<<15| 0<<14|1<<13)

  Touch sceen mode.

- #define MC13892_INACTIVE_MODE (0<<13|0<<12)

  Inactive sceen mode.

- #define INT_STATUS_0_REG 0x0

  MC13892 Interrupt status register 0.

- #define INT_MASK_0_REG 0x1

  MC13892 Interrupt mask register 0.

- #define MC13892_ADCDONE_BIT (1<<0)

  bit 0 ADCDONEI ADC has finished requested conversions

- #define MC13892_TSI_BIT (1<<2)

  bit 2 TSI Touch screen wakeup

- #define ADC0_REG 43

  MC13892 offset ADC register 0.

- #define ADC1_REG 44

MC13892 offset ADC register 1.

- #define ADC2_REG 45

  MC13892 offset ADC register 2.

- #define ADCDONE1 0x000001

  MC13892 ADC conversion done bit.

- #define ADCDONE1_MASK 0xFFFFFE

  MC13892 ADC conversion done bit mask.

- #define INT_MASK_REG 1

  MC13892 interrupt mask register 0.

- #define TIMER_REG 28

  MC13892 Switcher 4 register For PLL control.

- #define MC13892_I2C_DEVICE "/dev/i2c0"

  Default I2C address device name.

- #define MC13892_I2C_ADDR 8

  I2C address of MC13892.

- #define MC13892_TOUCH_RESISTANCE_DEFAULT 500

  Touch screen minimum resistance default.

- #define MC13892_I2C_BUS_SPEED 100000

  I2C Default bus speed.

- #define PLL_TIMER 0x3C000
- #define MC13892_MODE_MASK (1<<15|1<<14|1<<13|1<<12)

  Mask for MS13892 Mode.

- #define MC13892_INTERRUPT_MODE (1<<15|0<<14|0<<13|1<<12)

  Interrupt mode.

- #define MC13892_TOUCHSCREEN_MODE (1<<15| 0<<14|1<<13)

  Touch sceen mode.

- #define MC13892_INACTIVE_MODE (0<<13|0<<12)

  Inactive sceen mode.

- #define MC13892_INTERRUPT_STATUS_REG0 0

MC13892 Interrupt status register 0.

- #define MC13892_INTERRUPT_MASK0 1

  MC13892 Interrupt mask register 0.

- #define MC13892_ADCDONE_BIT (1<<0)

  bit 0 ADCDONEI ADC has finished requested conversions

- #define MC13892_TSI_BIT (1<<2)

  bit 2 TSI Touch screen wakeup

- #define MC13892_ADC2_ADD1_LSH 2
- #define MC13892_ADC2_ADD1_WID 10
- #define MC13892_ADC2_ADD2_LSH 14
- #define MC13892_ADC2_ADD2_WID 10
- #define CSP_BITFMASK(bit) (((1U << (bit ## _WID)) - 1) << (bit ## _LSH))
- #define CSP_BITFVAL(bit, val) ((val) << (bit ## _LSH))
- #define CSP_BITFEXT(var, bit) ((var & CSP_BITFMASK(bit)) >> (bit ## _LSH))

5.3.1 Detailed Description

Touch screen driver for the MC13892 used on the i.MX35 PDK.

Definition in file touch.h.

5.3.2 Define Documentation

5.3.2.1 #define TRACE

Trace function execution.

Note:

Compile time switched debug

Definition at line 48 of file touch.h.

Referenced by intr_thread(), process_data(), read_conversion(), set_touchscreen_mode(), start_conversion(), touch_devctrl(), touch_init(), touch_parm(), touch_pulse(), touch_reset(), and touch_shutdown().

5.3.2.2 #define TRACE_ENTER

Trace function enter execution.

Note:

Compile time switched debug

Definition at line 52 of file touch.h.

5.3.2.3 #define TRACE_EXIT

Trace function exit execution.

Note:

Compile time switched debug

Definition at line 56 of file touch.h.

Referenced by touch_devctrl(), and touch_reset().

5.3.2.4 #define DEBUG_MSG(x)

Debug message.

Note:

Compile time switched debug

Definition at line 60 of file touch.h.

Referenced by process_data(), and set_touchscreen_mode().

5.3.2.5 #define DEBUG_CMD(x)

Debug command.

Note:

Compile time switched debug

Definition at line 64 of file touch.h.

Referenced by i2c_read(), i2c_write(), intr_thread(), read_conversion(), and touch_parm().

5.3.2.6 #define TOUCH_INT 96

Touch sreen interrupt.

Definition at line 68 of file touch.h.

Referenced by touch_init().

5.3.2.7 #define INACTIVE_MODE 0

Touch sreen inactive mode MC13892 is neither waiting for a touch event or calculatinig a position.

Note:

May never enter into this mode

Definition at line 75 of file touch.h.

5.3.2.8 #define INTERRUPT_MODE 1

Touch screen interrupt mode MC13892 is waiting for a touch event.

In this mode an interrupt is create when the touch screen planes touch, i.e. when the screen is touched

Definition at line 81 of file touch.h.

Referenced by intr_thread(), set_touchscreen_mode(), touch_pulse(), and touch_reset().

5.3.2.9 #define POSITION_MODE 2

Touch screen position mode.

MC13892 is calculating the position of a touch events.

Definition at line 86 of file touch.h.

Referenced by intr_thread(), set_touchscreen_mode(), and touch_pulse().

5.3.2.10 #define DELTA_X_COORD_VARIANCE 24

Maximum allowed variance in the X coordinate samples.

Definition at line 90 of file touch.h.

Referenced by process_data().

5.3.2.11 #define DELTA_Y_COORD_VARIANCE 24

Maximum allowed variance in the X coordinate samples.

Definition at line 93 of file touch.h.

Referenced by process_data().

5.3.2.12 #define ABS(x) ((x) >= 0 ? (x) : (-(x)))

Definition at line 95 of file touch.h.

Referenced by process_data().

5.3.2.13 #define FLAG_INIT 0x1000

Driver state initialize.

Definition at line 98 of file touch.h.

Referenced by touch_reset().

5.3.2.14 #define FLAG_RESET 0x2000

Driver state reset.

Definition at line 100 of file touch.h.

Referenced by touch_init().

5.3.2.15 #define INTR_DELAY 75

Default minimum time between interrupt.

Definition at line 103 of file touch.h.

Referenced by touch_init().

5.3.2.16 #define HW_POLL_TIME 100

Default time between hardware polls, this is default the rate of event injection.

Definition at line 105 of file touch.h.

Referenced by touch_init().

5.3.2.17 #define PULSE_PRIORITY 21

Interrupt plus priority.

Definition at line 108 of file touch.h.

Referenced by touch_init().

5.3.2.18 #define PULSE_CODE 1

Interrupt pulse code.

Definition at line 110 of file touch.h.

Referenced by intr_thread().

5.3.2.19 #define MC13892_I2C_DEVICE "/dev/i2c0"

Default I2C address device name.

Definition at line 198 of file touch.h.

Referenced by touch_init().

5.3.2.20 #define MC13892_I2C_ADDR 8

I2C address of MC13892.

Definition at line 203 of file touch.h.

Referenced by i2c_read(), and i2c_write().

5.3.2.21 #define MC13892_I2C_BUS_SPEED 100000

I2C Default bus speed.

Definition at line 214 of file touch.h.

Referenced by touch_init().

5.3.2.22 #define MC13892_TOUCH_RESISTANCE_DEFAULT 800

Touch screen minimum resistance default.

This values is used to denote the end of a touch event.

Definition at line 209 of file touch.h.

Referenced by touch_init().

5.3.2.23 #define MC13892_MODE_MASK (1<<15|1<<14|1<<13|1<<12)

Mask for MS13892 Mode.

Definition at line 231 of file touch.h.

Referenced by set_touchscreen_mode().

5.3.2.24 #define MC13892_INTERRUPT_MODE (1<<15|0<<14|0<<13|1<<12)

Interrupt mode.

Used for tocuh screen INTERRUPT_MODE

Definition at line 232 of file touch.h.

Referenced by set_touchscreen_mode().

5.3.2.25 #define MC13892_TOUCHSCREEN_MODE (1<<15| 0<<14|1<<13)

Touch sceen mode.

Used for positional mode, ADC read x,y, restance

Definition at line 233 of file touch.h.

Referenced by set_touchscreen_mode().

5.3.2.26 #define MC13892_INACTIVE_MODE (0<<13|0<<12)

Inactive sceen mode.

Definition at line 234 of file touch.h.

Referenced by set_touchscreen_mode().

5.3.2.27 #define INT_STATUS_0_REG 0x0

MC13892 Interrupt status register 0.

Definition at line 152 of file touch.h.

Referenced by intr_thread(), start_conversion(), touch_pulse(), and touch_reset().

5.3.2.28 #define INT_MASK_0_REG 0x1

MC13892 Interrupt mask register 0.

Definition at line 156 of file touch.h.

Referenced by intr_thread(), touch_pulse(), and touch_reset().

5.3.2.29 #define MC13892_ADCDONE_BIT (1<<0)

bit 0 ADCDONEI ADC has finished requested conversions

Definition at line 246 of file touch.h.

Referenced by intr_thread(), and touch_pulse().

5.3.2.30 #define MC13892_TSI_BIT (1<<2)

bit 2 TSI Touch screen wakeup

Definition at line 248 of file touch.h.

Referenced by intr_thread(), and touch_pulse().

5.3.2.31 #define ADC0_REG 43

MC13892 offset ADC register 0.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 166 of file touch.h.

Referenced by set_touchscreen_mode(), and start_conversion().

5.3.2.32 #define ADC1_REG 44

MC13892 offset ADC register 1.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 170 of file touch.h.

Referenced by read_conversion(), and start_conversion().

5.3.2.33 #define ADC2_REG 45

MC13892 offset ADC register 2.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 174 of file touch.h.

Referenced by read_conversion().

5.3.2.34 #define ADCDONE1 0x000001

MC13892 ADC conversion done bit.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 178 of file touch.h.

Referenced by start_conversion().

5.3.2.35 #define ADCDONE1_MASK 0xFFFFFE

MC13892 ADC conversion done bit mask.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 182 of file touch.h.

Referenced by start_conversion().

5.3.2.36 #define INT_MASK_REG 1

MC13892 interrupt mask register 0.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 186 of file touch.h.

Referenced by intr_thread(), touch_pulse(), and touch_reset().

5.3.2.37 #define TIMER_REG 28

MC13892 Switcher 4 register For PLL control.

Note:

Refer to MC13892 Users Guide 034.pdf for more information

Definition at line 191 of file touch.h.

Referenced by touch_reset().

5.3.2.38 #define MC13892_I2C_DEVICE "/dev/i2c0"

Default I2C address device name.

Definition at line 198 of file touch.h.

5.3.2.39 #define MC13892_I2C_ADDR 8

I2C address of MC13892.

Definition at line 203 of file touch.h.

5.3.2.40 #define MC13892_TOUCH_RESISTANCE_DEFAULT 500

Touch screen minimum resistance default.

This values is used to denote the end of a touch event.

Definition at line 209 of file touch.h.

5.3.2.41 #define MC13892_I2C_BUS_SPEED 100000

I2C Default bus speed.

Definition at line 214 of file touch.h.

5.3.2.42 #define PLL_TIMER 0x3C000

Definition at line 217 of file touch.h.

5.3.2.43 #define MC13892_MODE_MASK (1<<15|1<<14|1<<13|1<<12)

Mask for MS13892 Mode.

Definition at line 231 of file touch.h.

5.3.2.44 #define MC13892_INTERRUPT_MODE (1<<15|0<<14|0<<13|1<<12)

Interrupt mode.

Used for tocuh screen INTERRUPT_MODE

Definition at line 232 of file touch.h.

5.3.2.45 #define MC13892_TOUCHSCREEN_MODE (1<<15| 0<<14|1<<13)

Touch sceen mode.

Used for positional mode, ADC read x,y, restance

Definition at line 233 of file touch.h.

5.3.2.46 #define MC13892_INACTIVE_MODE (0<<13|0<<12)

Inactive sceen mode.

Definition at line 234 of file touch.h.

5.3.2.47 #define MC13892_INTERRUPT_STATUS_REG0 0

MC13892 Interrupt status register 0.
Definition at line 240 of file touch.h.

5.3.2.48 #define MC13892_INTERRUPT_MASK0 1

MC13892 Interrupt mask register 0.
Definition at line 244 of file touch.h.

5.3.2.49 #define MC13892_ADCDONE_BIT (1<<0)

bit 0 ADCDONEI ADC has finished requested conversions
Definition at line 246 of file touch.h.

5.3.2.50 #define MC13892_TSI_BIT (1<<2)

bit 2 TSI Touch screen wakeup
Definition at line 248 of file touch.h.

5.3.2.51 #define MC13892_ADC2_ADD1_LSH 2

Definition at line 251 of file touch.h.

5.3.2.52 #define MC13892_ADC2_ADD1_WID 10

Definition at line 252 of file touch.h.

5.3.2.53 #define MC13892_ADC2_ADD2_LSH 14

Definition at line 254 of file touch.h.

5.3.2.54 #define MC13892_ADC2_ADD2_WID 10

Definition at line 255 of file touch.h.

5.3.2.55 #define CSP_BITFMASK(bit) (((1U << (bit ## _WID)) - 1) << (bit ## _LSH))

Definition at line 258 of file touch.h.

5.3.2.56 #define CSP_BITFVAL(bit, val) ((val) << (bit ## _LSH))

Definition at line 259 of file touch.h.

5.3.2.57 #define CSP_BITFEXT(var, bit) ((var & CSP_BITFMASK(bit)) >> (bit ## _LSH))

Definition at line 260 of file touch.h.

Referenced by read_conversion().

Index