

AN8650.02

SX8650 Linux® Touch Screen Driver

Table of Contents

1	Introduction	3
2	Connections.....	3
3	I²C interface.....	4
4	Touch Device Driver	5
4.1	<i>Interrupt rate considerations</i>	6
5	Software build process for AT91SAM9261 with SX8650	7
5.1	<i>Follow the instructions at http://wiki.openembedded.net/index.php/Getting_Started.....</i>	7
5.2	<i>Resulting binary files</i>	7
5.3	<i>Recompiling the kernel</i>	7
5.3.1	<i>Copy the kernel source from bitbake</i>	7
5.3.2	<i>adding the SX8650 driver to the kernel.....</i>	7
5.3.3	<i>Configuring the kernel</i>	8
5.3.4	<i>Compiling kernel.....</i>	8
5.3.5	<i>running the kernel.....</i>	9
6	References	10

1 Introduction

The SX8650 Linux® driver was developed to help users of the SX8650 touch screen controller to quickly use the device. The driver acts as a human input device in the Linux kernel. The driver was developed and tested using a SX8650EVK with an Atmel AT91SAM9261-EK evaluation board.

2 Connections

The SX8650 device must be connected to the host processor running the touch screen driver. This driver was developed and tested using the AT91SAM9261-EK board with SX8650EVK.

The following connections are required:

- Power +3.3v, and ground from the Atmel board prototype area.
- SDA for I²C™ data.
- SCL for I²C clock.
- Pen-down interrupt signal NIRQ.

On the SX8650EVK board, connections are made to JP1 to provide the I²C and interrupt signal. The four analog signals from the touch screen are made to J3 on the SX8650EVK. For details on JP1 and J3, refer to the SX8650EVK users guide.

On the AT91SAM9261-EK board, the included on-board touch controller ADS7843 is disconnected and replaced with the connections show in Figure 1. Refer to the AT91SAM9261-EK users guide for relevant documentation about connections to that board.

® Linux is a registered trademark of Linus Torvalds
™ I²C is trademark of NXP Semiconductor.

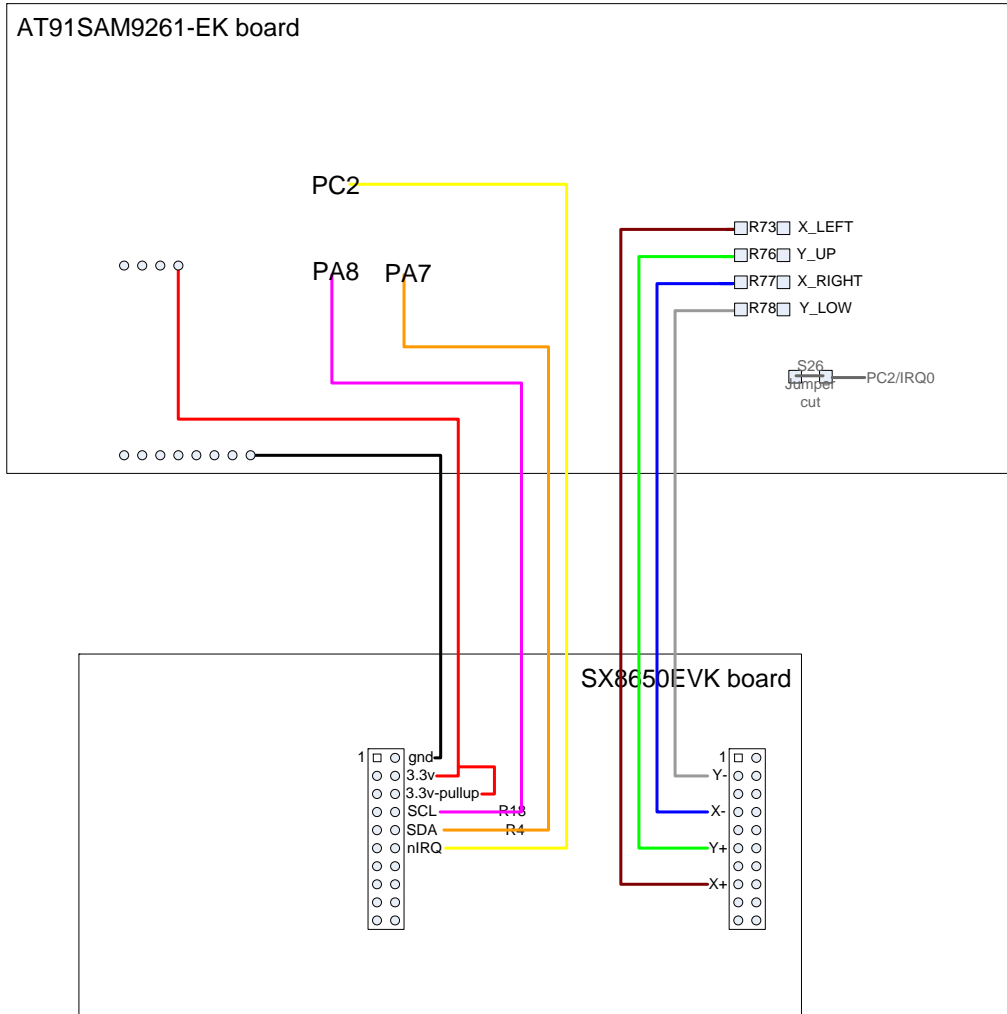


Figure 1 - SX8650 connection to AT91SAM9261-EK board

3 I²C interface

The I²C bus is the control and data bus through which the host processor sends commands and reads the analog values. The SX8650 driver uses the I²C bus using the standard Linux I²C API. The I²C API in the kernel provides all the necessary functions to write SX8650 registers and commands, and to read samples from the device.

For the hardware connection, the I²C signals are connected to PA8 = SCL and PA7 = SDA on the AT91SAM9261-EK. PA8 is available on J16-45, and PA7 is available on J16-44. Verify the pull-up resistor on the I²C bus: R18 and R4 on the SX8650EVK should be approximately 2.2kohm for sufficiently fast rise time on the I²C bus.

The SX8650-NIRQ signal is connected to PC2/IRQ0 pin available on J16-73 on the AT91 board. The on-board ADS7843 is disconnected by cutting the S26 jumper

4 Touch Device Driver

The device driver is implemented in the file `sx8650.c`.

The SX8650 is used in `PENTRG` mode, meaning that the NIRQ signal will be inactive when no pen is down on the touch screen. When the SX8650 device detects the pen is down on the screen, the device will first take samples, and then assert the NIRQ pin after the samples are available to be read by the driver software. After the samples have been read from the device, the NIRQ will de-assert. If the pen continues to be held down on the screen, the SX8650 device will again start a conversion cycle and assert the NIRQ when new samples are ready. The rate at which NIRQ will become asserted, is determined by the `POWDLY` setting multiplied by the number of channels selected to be converted in the channel mask register. The human interface input subsystem in the Linux kernel supports reporting pressure from the touch-screen in addition to the X, Y coordinates. The SX8650 is configured to sample X, Y, Z1, and Z2 channels.

To initialize the driver, the kernel calls the `sx8650_probe()` function. The SX8650 is reset and initialized for `PENTRG` operation, and the channel mask is set to convert X, Y, Z1, Z2 channels. In this probe function, the interrupt configuration is set up to cause `sx8650_irq()` to be called when the NIRQ pin is asserted. However, in the context that `sx8650_irq()` is called, no lengthy operations can be performed. The driver uses a work queue to read the samples from the SX8650 device. The `sx8650_irq()` function only calls `queue_work()`, which flags that samples need to be read. In the function `sx8650_pen_irq_worker()` the analog values are read from the SX8650 device and reported to the operating system. Normally samples will be immediately read from the touch screen device, yet if other kernel activities are busy in the work queue, some latency can be experienced in reading samples.

The SX8650 device does not uniquely report the pen-up condition. Instead, the software driver uses a high-resolution timer to detect the pen-up condition. If the pen has been lifted, a timeout will occur. This timeout condition can be detected when `sx8650_timer_handler()` is called. Every time the driver reads samples from the SX8650, the time-out is reset by calling `hrtimer_start()`, ensuring the timeout will only occur when interrupts from the device have stopped. The definition of `TS_TIMEOUT` must have a value which is appropriate for the interrupt rate from the SX8650 device.

The interrupt handler `sx8650_irq()` function is called twice for every interrupt from the SX8650. Therefore, state of the NIRQ pin from the device is checked by `get_pendown_state()` before continuing. The SX8650 will keep the NIRQ pin asserted until samples are read from the device, yet a de-asserted pin indicates samples are not ready.

No platform dependant code exists in `sx8650.c`. The actual hardware setup exists in `board-sam9261ek.c`. In this board setup file, the board-specific initialization is performed, and the physical pin used for NIRQ is defined. The I²C slave address of the SX8650 is defined in this file. The I²C master configuration is defined in `at91sam9261_devices.c`.

The SX8650 driver can be used on any hardware platform that Linux runs on, provided an I²C driver exists. The board-specific initialization and configuration must also be provided, using `board-sam9261ek.c` as an example. The board-specific initialization defines the slave address of the SX8650 as well as the interrupt pin on the target microprocessor. The struct `i2c_board_info` is used to pass this information.

For proper pressure measurement, the `y_plate_ohms` value is declared in the struct `sx8650_platform_data`. This structure also contains the functions for getting the pen-down state, clearing the pen interrupt, and any initialization required for the pen interrupt pin. These are all board-specific or microprocessor-specific definitions.

4.1 Interrupt rate considerations

The interrupt rate generated by the SX8650 is the `POWDLY` settings multiplied by the number of channels enabled in the channel mask register.

Although the primary intention of `POWDLY` is to accommodate the R/C time constant of the resistive touch screen, additional delay may be required according to the ability of the microprocessor to handle the interrupt rate while still providing adequate resources to application software. When increasing the interrupt rate of the SX8650, the developer must consider the CPU time used by the I²C driver, graphics subsystem and application software which must run while the pen is down on the touch screen.

The pen-trigger mode of the SX8650 permits faster interrupt rate due to the only I²C activity on each interrupt is reading the samples.

5 Software build process for AT91SAM9261 with SX8650

The OpenEmbedded/bitbake build system is one method of building an embedded Linux system. bitbake will retrieve and compile everything needed, including the cross compiler, the Linux kernel source, and all user applications. There is no need to manually download each individual item.

5.1 Follow the instructions at http://wiki.openembedded.net/index.php/Getting_Started

- In the local.conf, set MACHINE to "at91sam9261ek", and TMPDIR to <OETMP> to a place which has at least several gigabytes available
- A quick target recipe to build is minimal-image, but does not include X11

The build process can take several hours to complete.

5.2 Resulting binary files

After the bitbake build has finished, in your <OETMP> directory you will find binary files to burn into the evaluation board in the subdirectory `deploy/glibc/images/at91sam9261ek/`.

The root filesystem is ready to use, but the kernel will default to having the ADS7843 driver. The kernel must be re-built with the SX8650 driver instead.

5.3 Recompiling the kernel

5.3.1 Copy the kernel source from bitbake

Chose a place on your disk to build the kernel, then:

Into your working source directory, copy the Linux kernel source tree from:

```
<OETMP>/work/at91sam9261ek-angstrom-linux-gnueabi/linux-<version>
```

5.3.2 adding the SX8650 driver to the kernel

- Change current directory to your working source directory, where your copied kernel resides.
- Goto the sub-directory `drivers/input/touchscreen` in the kernel source
- Add SX8650 to the Kconfig file in this directory, after the TOUCHSCREEN_ADS7846 entry:

```
config TOUCHSCREEN_SX8650
    tristate "SX8650 based touchscreens"
    depends on I2C
    help
        Say Y here if you have a touchscreen interface using the
        SX8650 controller, and your board-specific
        setup code includes that in its table of I2C devices.

        If unsure, say N (but it's safe to say "Y").

        To compile this driver as a module, choose M here: the
        module will be called sx8650.
```

- Add SX8650 to the Makefile:

```
wm97xx-ts-y := wm97xx-core.o

obj-$(CONFIG_TOUCHSCREEN_ADS7846) += ads7846.o
obj-$(CONFIG_TOUCHSCREEN_SX8650) += sx8650.o      # <- added
obj-$(CONFIG_TOUCHSCREEN_ATMEL_TSADCC) += atmel_tsadcc.o
obj-$(CONFIG_TOUCHSCREEN_BITSYS) += h3600_ts_input.o
```

- copy the `sx8650.c` driver source file into `drivers/input/touchscreen`
- copy `sx8650.h` into `include/linux/i2c`
- copy the `board-sam9261ek.c` into the subdirectory `arch/arm/mach-at91`
This file contains hardware-specific definitions for the `sx8650` for this board.
- you may wish to change the `.udelay` setting in `at91sam9261_devices.c` for I²C bus speed.

5.3.3 Configuring the kernel

- in the toplevel directory of kernel source, configure kernel:

```
# make ARCH=arm menuconfig
```

- Enable I²C driver:
Device Drivers -> I2C support -> I2C Hardware Bus support -> GPIO-based bitbanging I2C
(or TWI driver if available)
- Disable the ADS7843 driver, and enable SX8650 driver:
Device Drivers -> Input device support -> Touchscreens
- exit and save

5.3.4 Compiling kernel

- Add to \$PATH the gnu compiler directory:

```
# export PATH=<OETMP>/cross/armv5te/bin/:$PATH
```

- build command:

```
# make ARCH=arm CROSS_COMPILE=arm-angstrom-linux-gnueabi-
```

The resulting kernel image will be in the subdirectory `arch/arm/boot`

U-boot requires a `ulmage`, it cannot boot a `zImage`.

- convert `zImage` to `ulmage`:

```
<OETMP>/staging/<hostarch>-linux/usr/bin/mkimage -A arm -O linux -C none -T kernel  
-a 20008000 -e 20008000 -n linux-2.6 -d zImage <output_uImage_name>
```

5.3.5 running the kernel

The kernel can be run two ways:

- For quick testing, uboot can tftp the kernel "tftp 0x21400000 <uImage>", if you already have programmed into the board at91bootstrap, uboot and root filesystem.
- Or for flash burning, the procedure for flashing demo can be used from <http://www.linux4sam.org/twiki/bin/view/Linux4SAM/GettingStarted>

You can edit the .tcl file for changing kernelFile

The procedure for using SAM-BA in linux is at <http://www.linux4sam.org/twiki/bin/view/Linux4SAM/SoftwareTools>

```
rmmod usbserial
modprobe usbserial vendor=0x03eb product=0x6124
lsusb -d 03eb:6124
```

6 References

- [1] SX8650 datasheet
- [2] AT91SAM9261-EK board
http://atmel.com/dyn/products/tools_card.asp?tool_id=3820
- [3] Linux on AT91
<http://www.linux4sam.org>

© Semtech 2009

All rights reserved. Reproduction in whole or in part is prohibited without the prior written consent of the copyright owner. The information presented in this document does not form part of any quotation or contract, is believed to be accurate and reliable and may be changed without notice. No liability will be accepted by the publisher for any consequence of its use. Publication thereof does not convey nor imply any license under patent or other industrial or intellectual property rights. Semtech assumes no responsibility or liability whatsoever for any failure or unexpected operation resulting from misuse, neglect improper installation, repair or improper handling or unusual physical or electrical stress including, but not limited to, exposure to parameters beyond the specified maximum ratings or operation outside the specified range.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

Contact Information

Semtech Corporation
Advanced Communications and Sensing Products Division
200 Flynn Road, Camarillo, CA 93012
Phone (805) 498-2111 Fax : (805) 498-3804