Freescale Semiconductor
Application Note

Document Number: AN4488
Rev. 0, 5/2012

CRTouch Android Software Driver for I.MX53 QSB

by: Humberto Vazquez
Application Engineering
Guadalajara, Mexico

Contents

1 Introduction

This application note describes the software device driver for the CRTouch device and how to use it on the I.MX53.

The driver allows communication with the CRTouch and provides these features:

- XY coordinates
- Zoom In and Out gestures
- Rotate clockwise and counterclockwise gestures
- Slide Up, Down, Left and Right event
- Electrode events in different configurations:
  - Keypad
  - Slider
  - Rotary
- Shutdown and Resume device

2 Hardware Connections

The driver uses five connections to support all the features:

- I2C SDA — Sends data
- I2C SCL — Synchronizes communication
- GPIO IRQ — Interruption to start communication
- GND — Ground
- GPIO LOW_POWER — Wakes the CRTouch device from shutdown

The used I2C SDA and I2C SCL pins are taken from the LVDS0 DisplayOutput and weld them to the resistors. See Figure 1.

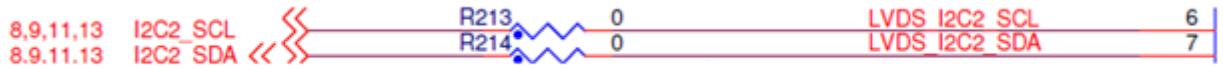

Figure 1. I2C SDA and I2C SCL pins

The used GPIO IRQ pin is the USER_LED_EN and is welded from the Gate input:

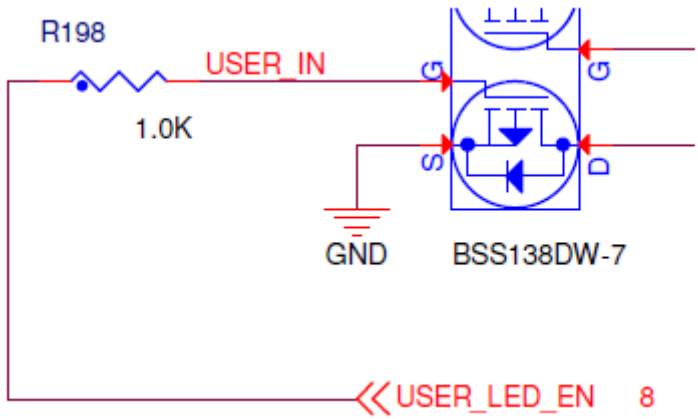

Figure 2. GPIO IRQ pin

The used GPIO LOW_POWER pin is the I2C3 SDA and must be configured as GPIO:

Figure 3. GPIO LOW_POWER pin

3 Driver Overview

The driver is composed of two *c* code files — crtouch_mt.c and crtouch_mt.h.

The driver is included in the next kernel path:kernel/drivers/input/touchscreen/

The driver register 0x49 address as a slave in the mx53_loco.c file The address is placed in the I2C bus 1.

NOTE

The device supports two different addresses, quit the jumper from the I2CADRSEL to assign address 0x49 for the CRTouch device.

Driver initialization

- The driver configures the trigger events register on the device. This allows it to reply to resistive basic events for the touch screen, those events are touch and release: Trigger Events register = 0x81
- Add resistive gestures from the Configuration register (Zoom, Rotate) and clean the slide event.

Driver behavior

When a falling edge is generated in the I.MX53 GPIO IRQ, an interruption is generated that launches a thread to process the information. That thread reads the status_register_1 to determine what occured on the CRTouch device, depending on the event generated is what the driver reports to the application. When the driver finishes reporting data to the application it goes to sleep and waits for another interruption.

Events reported

- A touched resistive screen generates absolute XY coordinates to the application
- Zoom In and Out events simulate a distance of two percent of the horizontal touch screen resolution.

  Zoom In example

  If a zoom in event is generated by the CRTouch device the coordinates simulated are the following:

  Initial points

  X1 — 45 percent horizontal resolution

  Y1 — 50 percent vertical resolution

  X2 — 55 percent horizontal resolution

  Y2 — 50 percent vertical resolution

  Final position

  X1 — 43 percent horizontal resolution

  Y1 — 50 percent vertical resolution

  X2 — 57 percent horizontal resolution

  Y2 — 50 percent vertical resolution

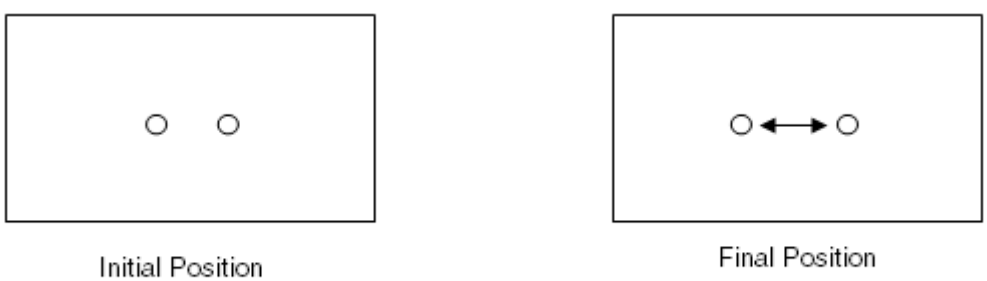

Figure 4. Zoom In

Zoom Out example

If a zoom out event is generated by the CRTouch device the coordinates simulated are the following:

Initial points

X1 — 43 percent horizontal resolution

Y1 — 50 percent vertical resolution

X2 — 57 percent horizontal resolution

Y2 — 50 percent vertical resolution

Final position

X1 — 45 percent horizontal resolution

Y1 — 50 percent vertical resolution

X2 — 55 percent horizontal resolution

Y2 — 50 percent vertical resolution

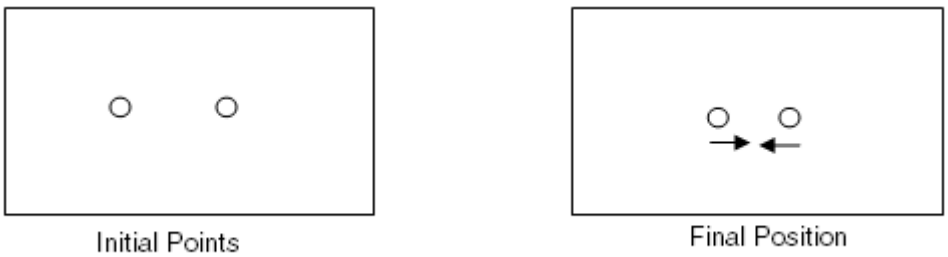

Figure 5. Zoom Out

- Rotate Clockwise and Counter clockwise events simulate the application in conjunction with a two touch coordinate rotary movement, reflecting an angle read from the device.

Rotate Clockwise example

If a Rotate Clockwise event is generated by the CRTouch device the coordinates simulated are the following:

Initial points

X1 — 50 percent horizontal resolution

Y1 — 50 percent vertical resolution

X2 — 50 percent horizontal resolution + horizontal resolution

Y2 — 50 percent vertical resolution

Final position

X1 — 50 percent horizontal resolution

Y1 — 50 percent vertical resolution

If the angle is less than or equal to 180

X2 — 50 percent horizontal resolution + (cos(angle) * horizontal resolution)

Y2 — 50 percent vertical resolution + (sin(angle) x vertical resolution)

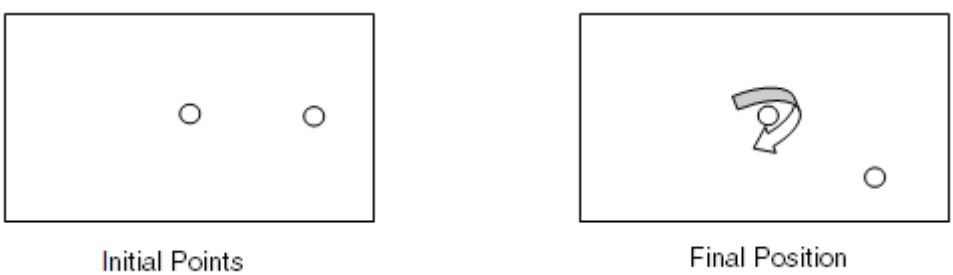

Figure 6. Rotate Clockwise

Rotate Counter Clockwise example

If a Rotate Counter Clockwise event is generated by the CRTouch device, the coordinates simulated are the following:

Initial points

X1 — 50 percent horizontal resolution

Y1 — 50 percent vertical resolution

X2 — 50 percent horizontal resolution + horizontal resolution

Y2 — 50 percent vertical resolution

Final position

X1 — 50 percent horizontal resolution

Y1 — 50 percent vertical resolution

If the angle is less than or equal to 180

X2 — 50 percent horizontal resolution + (cos(angle) * horizontal resolution)

Y2 — 50 percent vertical resolution + (~sin(angle) x vertical resolution)

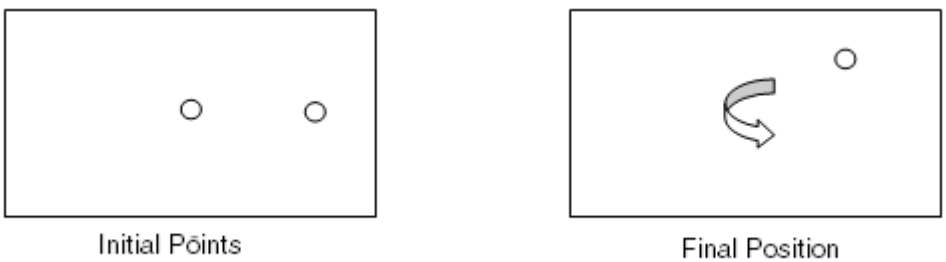

Figure 7. Rotate counter clockwise

CRTouch Resistive Event	Android Event
Slide down	KEY_H
Slide up	KEY_G
Slide left	KEY_F
Slide right	KEY_E

CRTouch Capacitive Event	Android Event
Slider • Incremental • Decremental	KEY_I KEY_J
Rotary • Incremental • Decremental	KEY_K KEY_L
Keypad • Button 0 • Button 1 • Button 2 • Button 3	KEY_A KEY_B KEY_C KEY_D

4 Communications

The driver communicates with the device via the I2C protocol through the I2C subsystem.

The driver responds to the interruption generated by CRTouch in the Pending events pin. The Pending events pin interrupts the driver with a falling edge, and launches a thread to read the status_register_1. This identifies the occurred event on the device. Depending on the occurred event on the device, the driver reports it to the application through the input device. When the thread finishes the process, it goes to sleep and waits for another interruption to launch the thread again.

5 Device Configurations

You can read and write CRTouch registers via the driver and configure behaviors. The driver creates a device file to read any register or configure any configurable register. It is registered under /dev where the device file is called *crtouch_dev*.

Configuration protocol

You can read and write a register byte per byte with the next API provided by the CRTouch driver:

Read

Send a variable char address with the address to read. The value read is stored in the same variable. A negative value is returned on error.

Example:

```
char address_to_read = 0x40;

error = read( fd , &address_to_read ,1)
```

Write

Send a variable char array address with the address and the data to write. The first byte is the address and the second byte is the data to write. The negative value is returned on error.

Example:

```
unsigned char data_to_write[2] = {0x40 , 0xFC};

error = write( fd , &data_to_write ,2)
```

- Read an address on a Linux application

```
#include <stdio.h>
#define DEVICE_FILE_NAME "/dev/crtouch_dev"
#define CONFIGURATION            0x40
#define RW                       0666
int main(int argc, char *argv[])
{
        char address_to_read = CONFIGURATION;
        int result = 0;
        int fd;

        if( ( fd = open (DEVICE_FILE_NAME , RW ) ) < 0 ){
        printf("Can't open device file: %s\n", DEVICE_FILE_NAME);
        exit(-1);
        }

        if( (result = read( fd , &address_to_read ,1) ) < 0){
        printf("Can't read register");
        exit(-1);
        }
        else{
        printf("Data readed: 0x%x", address_to_read);
        }

        return 0;
}
```

- Write an address on a Linux application

```
#include <stdio.h>
#define DEVICE_FILE_NAME "/dev/crtouch_dev"
#define ADDRESS            0x40
#define DATA               0xFC
#define RW                 0666

int main(int argc, char *argv[])
{
        unsigned char data_to_write[2] = {ADDRESS , DATA};
        int result = 0;
        int fd;

        if( ( fd = open (DEVICE_FILE_NAME , RW ) ) < 0 ){
        printf("Can't open device file: %s\n", DEVICE_FILE_NAME);
        exit(-1);
        }

        if( (result = write( fd , &data_to_write ,2) ) < 0){
        printf("Can't write register");
        exit(-1);
        }
        else{
        printf("Data written");
        }

        return 0;
}
```

NOTE

To configure the CRTouch device in the Android OS it is necessary to use a Java Native Interface (JNI) to call the native code. Device file calls are not supported in Java. You need to create a share object with NDK to communicate functions provided by the driver and Java application.

6 Low Power Modes

The CRTouch driver supports low power mode. When Android OS goes to a suspend state an I2C command is sent to the CRTouch device to go into shutdown mode. When the Android resumes, a low pulse is sent through the GPIO LOW_POWER to wake the CRTouch.

7 Touchscreen Calibrations

To calibrate the CRTouch device, you must add the crtouch_calibration parameter on bootargs in the i.MX53QSB.

An application runs at boot where the points indicated must be touched. For better precision you must use a stylus to calibrate the first three single points. If you use a four wire touchscreen the application shows you two points at the same time, your fingers must be used for better results. The application restarts if you do not touch the points in the indicated positions. The calibration information is stored in: /data/system/crtouch_calibrated. You can view the device resolution in which it was calibrated for the last time.

Example:

cat /data/system/crtouch_calibrated

If recalibration is wanted, the crtouch_calibrated file needs to be deleted.

Example:

rm /data/system/crtouch_calibrated

NOTE

A reboot is needed after CRTouch calibration.

8 Android Application

The CRTouch reports two touch coordinates to simulate a Gesture through the input device as all devices do. It reports the coordinates via the input device.

It is important to configure the device with the features needed when the application resumes and to clear those features when the application goes into Pause state.

This is an example code you can use to develop and obtain events reported by the CRTouch:

NOTE

You must create the SO with NDK Android Tool

```
package com.crtouch;

import android.app.Activity;
import android.os.Bundle;
import android.view.KeyEvent;
import android.view.MotionEvent;
import android.view.View;
import android.view.View.OnTouchListener;

public class CRTOUCHGallery extends Activity implements OnTouchListener
{

        /**
        * Load SO to configure cr-touch.
        * */
        static {
        System.loadLibrary("crtouch_config");

            }

        public static final int TRIGGER = 0x41;
        public static final int CONFIGURATION = 0x40;

        short setTrigger[] = {TRIGGER,0x83};
        short setTriggerBasic[] = {TRIGGER,0x81};
        short setConfiguration[] = {CONFIGURATION,0xFC};
        short setConfigurationBasic[] = {CONFIGURATION,0x84};
        short setEvents[] = {0x6D,0x03};


       /** Called when the activity is first created. */
       @Override
       public void onCreate(Bundle savedInstanceState) {

       }

       /**
        * Called when a key up event has occurred
        * */
       @Override
          public boolean onKeyUp(int keyCode, KeyEvent event) {

           switch(keyCode)

                 {

                           case KeyEvent.KEYCODE_A:/*Keypad 0*/
```

```
                                                        return true;

                    case KeyEvent.KEYCODE_B:/*Keypad 1*/
                                                        return true;

                    case KeyEvent.KEYCODE_C:/*Keypad 2*/
                                                        return true;

                    case KeyEvent.KEYCODE_D:/*Keypad 3*/
                                                        return true;

                    case KeyEvent.KEYCODE_E:/*Slide Right*/
                                                        return true;

                    case KeyEvent.KEYCODE_F:/*Slide Left*/
                                                        return true;

                    case KeyEvent.KEYCODE_G:/*Slide Up*/
                                                        return true;

                    case KeyEvent.KEYCODE_H:/*Slide Down*/
                                                        return true;

                    case KeyEvent.KEYCODE_I:/*Slider Incremental*/
                                                        return true;

                    case KeyEvent.KEYCODE_J:/*Slider Decremental*/
                                                        return true;

                    case KeyEvent.KEYCODE_K:/*Rotary Incremental*/
                                                        return true;

                    case KeyEvent.KEYCODE_L:/*Rotary Decremental*/
                                                        return true;

            }

         return false;

        }

    /**
     * Called when a key down event has occurred
    * */
         @Override
         public boolean onKeyDown(int keyCode, KeyEvent event) {

               switch(keyCode){

                       case KeyEvent.KEYCODE_C:
                                                        return true;

                       case KeyEvent.KEYCODE_A:
                                                        return true;

                       case KeyEvent.KEYCODE_D:
                                                        return true;
            }

         return false;

        }


         @Override
         public boolean onTouch(View v, MotionEvent event) {

               switch (event.getAction() & MotionEvent.ACTION_MASK) {
```

```
                        case MotionEvent.ACTION_DOWN:
        break;
                        case MotionEvent.ACTION_UP:
                        case MotionEvent.ACTION_POINTER_UP:
        break;

                        case MotionEvent.ACTION_POINTER_DOWN:
        break;


                        case MotionEvent.ACTION_MOVE:
        break;

                 }

                    return true; /*event handled */
             }

        /**
         * Called when the system is about to start resuming a previous activity
         * to configure CRTOUCH for only 1 touch report.
         */
        @Override
           protected void onPause(){

           int open;

                  super.onPause();
            open = openFile("/dev/crtouch_dev",666);
            writeFile(open,setConfigurationBasic,2);
            writeFile(open,setTriggerBasic,2);
            closeFile(open);

           /*Move Task to background*/
             moveTaskToBack(true);

           }

        /**
         * Called when the activity will start interacting with the user
         * to configure CRTOUCH for Gestures and Keypad Configuration.
         */
        @Override
           protected void onResume(){

           int open;

                  super.onResume();
            open = openFile("/dev/crtouch_dev",666);
            writeFile(open,setConfiguration,2);
            writeFile(open,setTrigger,2);
            writeFile(open,setEvents,2);

            closeFile(open);

           }

                 /**
                  * Invoke a native method to open CRTOUCH file.
                  *
                  * @param  fileName     Name of the file to open
                  * @param  permissions  How file should be opened
                  * @return              File Descriptor (fd) from the file opened
or                                          NULL if it can not be opened
                  */
                 public native int openFile(String fileName, int permissions);

                 /**
                  * Invoke a native method to close CRTOUCH file.
                  *
```

```
     * @param  fd  File descriptor to close
     * @return       0 If file was closed on succeed or -1 if error
     */
    public native int closeFile(int fd);

    /**
     * Invoke a native method to read CRTOUCH file (Registers).
     *
     * @param  fd  File descriptor from file opened
     * @param  data[char address]
     * @return       1 on succeed or -1 on error
     */
    public native int readFile(int fd, short data,int sizeofData);

    /**
     * Invoke a native method to write CRTOUCH  file (Registers).
     *
     * @param  fd  File descriptor from file opened
     * @param  data[char address][char data]
     * @return 1 on succeed or -1 on error
     */
    public native int writeFile(int fd, short[] data,int sizeofData);

}
```

9 Share Object Code to Configure CRTouch

```
#include <stdio.h>
#include <jni.h>
#include <android/log.h>

/*Header JNI for Android*/
/*Java_package_className_functionName*/

/*Open a device File*/
jint Java_com_crtouch_CRTOUCHGallery_openFile
(JNIEnv *env, jobject obj, jstring file, jint permissions)
{

  int result = 0;

      /*Convert jstring to C type*/
      const char *cFile = (*env)->GetStringUTFChars(env,file,0);

      result = open(cFile,(int)permissions);

      /*Release type*/
      (*env)->ReleaseStringUTFChars(env,file,cFile);

      return result;
}

/*Close a device File*/
jint Java_com_crtouch_CRTOUCHGallery_closeFile
  (JNIEnv *env, jobject obj, jint fd)
{
      return close((int)fd);
}

/*Read a device File*/
jint Java_com_crtouch_CRTOUCHGallery_readFile
  (JNIEnv *env, jobject obj, jint fd, jshort address, jint sizeofAddress)
{

      int result;
```

```
        /*Need to send a C variable*/
        char adduser = address;
        result = read((int)fd,&adduser,(int)sizeofAddress);

        if(result < 0)
                return result;
        else
                return adduser;

}

/*Write a device File*/
jint Java_com_crtouch_CRTOUCHGallery_writeFile
  (JNIEnv *env, jobject obj, jint fd, jshortArray data, jint sizeofData)
{
        int result = 0;
        char array[2];

        jshort* cData;

        /*Convert jshortArray to C type*/
           cData = (*env)->GetShortArrayElements(env,data,0);

        /*Change ShortArray to CharArray*/
           array[0]=(char)(*cData);
           cData++;
           array[1]=(char)(*cData);

        result = write((int)fd,&array,(int)sizeofData);

        /*Release type*/
        (*env)->ReleaseShortArrayElements(env,data,cData,JNI_ABORT);

  return result;

}
```

10 Conclusion

This driver gives you the opportunity to manage a resistive touch screen that supports multi-touch gestures and four electrodes with different settings. It also indicates where the user is touching with a single touch and gives you the opportunity to support two finger gestures like Zoom and Rotate.

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4488
Rev. 0, 5/2012