Freescale Semiconductor
Application Note

Document Number: AN4057
Rev. 0, 3/2010

Resistive Touchscreen Controller Using the S08 Family

by: **Andy Juengst**
Applications Engineer

1 Introduction

Several materials available today have the unique property of being both transparent at optical frequencies and electrically conductive. Because of these properties, these materials can be placed over display panels and used to select options or interact with graphics displayed on the panel. Pressing or touching a point on the screen changes the electrical properties of the material that can be detected by a microcontroller. Several methods are available to detect this touch. Capacitive touch senses the change in charge rate caused by touching the panel. This method is out of scope for this application note and is not discussed. You can find more details about capacitive touch sensing at www.freescale.com/proximity.

The more common method is to use two layers of the resistive material separated by a small distance. When pressed, the two layers electrically short and form a voltage divider that can be measured. There are four-wire, five-wire, and eight-wire resistive touch panels. This application note describes all three methods. A software example is provided for the four and eight-wire methods. Modification to a five-wire panel requires only a minimal amount of software re-work. The sample code can be found on the same web page as this application note.

Contents

2 Resistive Touchscreen Fundamentals

The basic operation of the three different resistive touchscreen methods are described below.

2.1 Four-Wire Touchscreen Interface

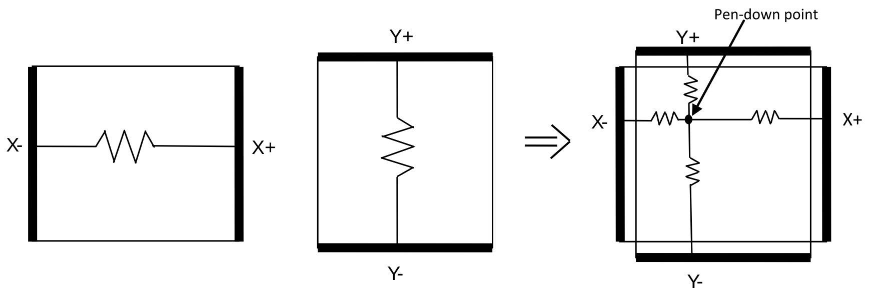

Figure 1. Four-wire touchscreen schematic

Figure 1 shows the basic make-up of a four-wire touchscreen interface. A piece of transparent conductive material has contacts placed on opposite ends. A second piece of the same material with contacts oriented 90° to the first piece is placed over the first piece with a small gap between the two. By pressing on the film, the two layers are made to come in contact with each other. The result is a simple voltage divider that can be sensed with a microcontroller. For example to measure the Y coordinate: Y+ is driven to V_{dd}. Y– is driven to V_{ss}. X– is set to an input with no pull-up or pull-down resistor, which effectively takes it out of the circuit. X+ is connected to an analog-to-digital input. A reading on X+ gives the voltage at the y– position of the pen-down point. The voltage ratio of this point to V_{dd} is the same as the A/D reading to the A/D range. This setup is shown in Figure 2 "(a)". The reading yields how far up the y– axis the point is by the following,

Equation 1:

$$\frac{A/D \quad reading}{A/D \quad resolution} = \frac{y \quad value}{height \quad of \quad panel}$$

The A/D resolution is the full-range of the A/D mode, depending on how many bits of resolution the A/D is set for (A/D resolution = 4096 for 12-bit mode). The height of panel can be any scale the system requires and is a known reference value to the host microprocessor. One common “ height ” is to use the full scale A/D resolution. The measured A/D reading then becomes the transmitted value to the host microprocessor, no multiplication or division is required. The host processor knows the ratio of pixels to A/D points and can then convert the reading into the appropriate pixel count from the edge of the panel. Another common method is to use the resolution of the LCD panel for the “ height. ” Consequently, the touchscreen controller sends the host processor the pixel data without any further calculations needed. A similar process is followed to determine the x coordinate.

Most systems do not need the uC continuously reading the control panel. This is an inefficient use of power because an active pen-down condition happens a small percentage of the time. The setup in Figure 1 also allows for the uC to go into a sleep mode and use an active button press to wake the uC causing it to start reading data points. Figure 2 "(b)" shows the configuration needed to sense a new pen-down condition. X+ changes from an A/D pin into a key board interrupt (KBI) with the weak internal pull-up resistor enabled. X– and Y+ are set-up as inputs with no pull-up or pull-down enabled. Y– is set as an output driven to V_{ss}. The uC is put into a low power sleep mode with the KBI enabled on the falling edge of X+. When a pen-down condition occurs, the resistance to V_{ss} through the Y– pin is low enough to overcome the weak pull-up on X+. As a result, X+ is pulled low and a KBI interrupt occurs, waking the uC and allowing readings to take place. A similar operation is to continuously read data points until the pen-up condition is sensed and then go back to sleep.

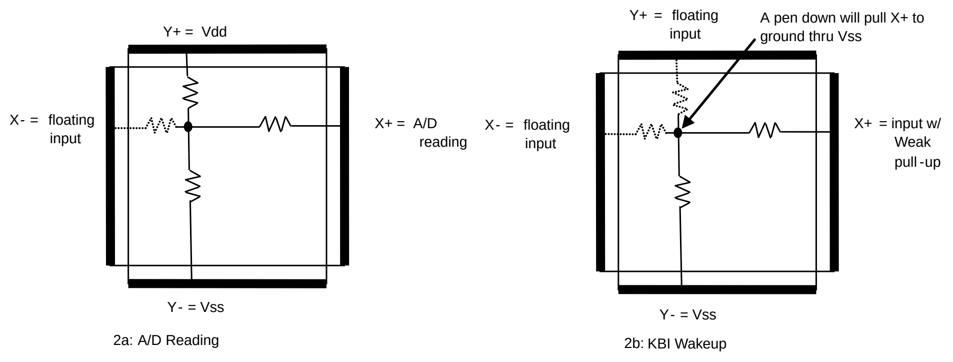

Figure 2. (a) A/D reading and (b) KBI wakeup

2.2 Eight-Wire Touchscreen Interface

There are inherent losses in the wires, connections, and inactive areas of the touchscreen. As a result, a value of less than V_{dd} and greater than V_{ss} is applied to the points noted above causing an error in the conversion. This error is calibrated out in an eight-wire touchscreen. Another sense pad is placed on each contact as shown in Figure 3. The V_{dd} is applied to Y+ and V_{ss} to Y–. The A/D readings are taken at both Y+ sense and Y– sense. The difference between these two readings is used for the A/D resolution value in Equation 1. Due to losses in the setup, the full range of A/D values that you might measure for a 12-bit setup is some value below 4096. A similar reading is taken for the X– axis.

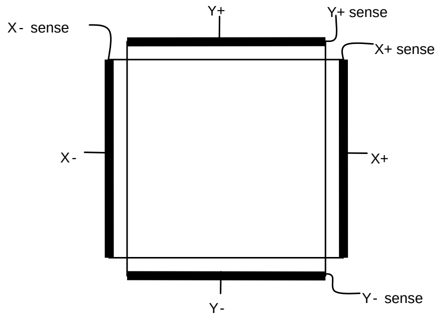

Figure 3. Eight-wire setup

2.3 Five-Wire Interface

One drawback of these resistive materials is that they can wear down over time. This wearing can affect the A/D readings as described. One method to overcome this degradation is used in a five-wire setup as shown in Figure 4 . The bottom layer is less susceptible to wear and has pads at each corner.

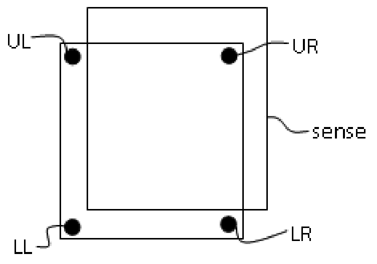

Figure 4. Five-wire setup

These pads are labeled UL, UR, LL, and LR. The entire top sheet is the sense pad. Scratches and wear have minimal impact. In a similar method as with the four-wire setup, the x– coordinate can be determined by applying V_{dd} on both UR and LR while applying V_{ss} on both UL and LL. Taking an A/D reading on the sense pad yields the x coordinate. Similarly, the y coordinate is obtained by putting V_{dd} on UL and UR while putting V_{ss} on LL and LR.

In a similar manner to the four-wire setup, the device can be put in a low-power sleep mode and woken with a keyboard interrupt. Three of the pads on the bottom sheet are set-up as high-impedance floating inputs. The fourth is a KBI with a weak pull-up, the sense pad is set to V_{ss}. When a pen-down condition occurs, the pin set as a KBI is pulled low to ground through the sense pin forcing a KBI.

3 Resistive Touchscreen Application Considerations

How a touchscreen is implemented is affected by both the application and the environment the touchscreen will be used in. Several key parameters to pay attention to are described below.

3.1 Conversion Resolution

The resolution of the touchscreen is determined by the application. Most applications that use large selection buttons on the display can get by with an 8-bit resolution or less. Applications that use smaller selection buttons, or that have buttons displayed very close or adjacent to each other, may require a 10-bit resolution. For applications that require the ability to follow the pen as it moves around the screen a 12-bit resolution may be required. The software provided did not test for a moving pen. The principles used are the same for this type of application. Some modifications may need to be made to sampling rates, filtering, and so on. The sample code is set for a 10-bit resolution.

3.2 Environment Impact

The environment, a touchscreen is placed in and can adversely affect the proper operation of the touchscreen. Improper use can result in bad coordinate value reads, readings missing, or readings taken when a pen-down condition does not exist. All of these result in a bad user experience.

3.2.1 Noise

Anyone who has used analog-to-digital converters knows that these readings are very susceptible to electrical noise in a system. Any voltage bounce in the power rails, or on the A/D lines can cause significant error in the A/D reading. This results in the wrong location calculated for the pen-down condition. As a result, care must be taken to isolate the touchscreen system from any noise source. In most cases, following good design practices for EMI is enough to obtain good performance. In extreme environments, isolated power rails and shielded A/D signals may be required.

3.2.2 Coordinate Median Filtering

Even with aggressive noise abatement techniques, there is always the chance of a spurious noise bounce within a system. Consequently, many touchscreen applications choose to implement a software filter on the readings. This application implements a simple median filter. Five consecutive readings are taken and the median value is chosen as the pen-down value. Depending on the application, more (or less) readings, an average, running-average, or a more advanced algorithm can be implemented.

3.2.3 Sampling Delay

Due to the parasitic capacitance of the system, there can be a delay between configuring the system for a reading and seeing the correct voltage on the A/D pin. Again, the amount of time required depends on each implementation, but for most systems this delay is small and may be ignored. This application does include a small delay between configuring the ports and starting the reading. The delay can be increased or eliminated as needed.

Even with the delay, the first reading on a new pen-down condition often results in inaccurate coordinates. As a result, the sample code does not transmit the first data points and therefore results in a 20 ms delay between pen-down and the first data transmission. This is considered small enough for most applications.

3.2.4 Pen-Up, Pen-Down Debounce

A KBI is used to wake the system from its low-power sleep mode. This is done by setting a pad on the first layer to a KBI with a weak pull-up. A pad on the second layer is tied to ground. When a pen-down occurs, the KBI is pulled low to ground through the second layer. The KBI controller typically responds to an interrupting signal within a clock cycle or two. As a result any noise spike on this pin can wake the system. A method needs to be implemented to avoid sending false pen-down conditions to the main microprocessor.

The system needs to detect when a pen-up condition occurs. This is managed in a similar fashion, except that the first pad is set to an input with a weak pull-up, KBI is not required. If the input is low, a pen-down condition still exists. A debounce method needs to be implemented to ignore false pen-up conditions.

The sample code provided manages both of these functions with the same debounce routine. The code does not actually perform a debounce on a pen-down condition prior to entering the main control loop. When a KBI is sensed, the uC wakes and reads the coordinates. It then enters a loop waiting for the timer described in Chapter 3.4 Coordinate Refresh Rate to expire before transmitting the coordinates. During the loop, the code continuously performs a pen-up debounce. When the timer expires, it determines if it has a valid pen-down condition and reacts accordingly. If pen-up is detected on the first code pass through, no data is transmitted and the uC goes back to sleep (a false KBI trigger). When pen-down is detected, it transmits the measured coordinates and goes back to the top of the loop to measure the coordinates again. It stays in the loop until a pen-up occurs. At the end of the loop that detects pen-up, the last valid pen-down data is transmitted and sleep mode is entered.

There are many methods to perform a debounce on an input. The method that best fits your system requirements should be implemented. The system chosen in this application weighs heavily towards a pen-up condition. While in the debounce loop, the state of the Input pin is monitored. During this loop, any low value (pen-down condition) increments a counter. When a high value (pen-up condition) is read, the counter is reset. At the end of the transmit interval, when the code leaves this loop; pen-up, pen-down is determined based on the value of this counter.

3.3 Calibrations

There are several methods of calibration that can improve the accuracy of the coordinates transmitted. These methods are described below.

3.3.1 Resistive Losses

Chapter 2.2 Eight-Wire Touchscreen Interface describes how an eight-wire touchscreen can be used to calibrate out the losses in the wiring and pads of the ITO. The code provided implements an eight-wire touchscreen and performs these calculations every power-up.

3.3.2 Touchscreen and Display Misalignment

When the resistive layers that comprise the touchscreen are placed over the display they often are not aligned or squared to each other. This leads to an offset in touchscreen coordinates versus display coordinates. As a result, an alignment calibration is often needed. This is the “ first time ” calibration that many users of touchscreen panels are familiar with. At the first power-up, you must pen-down on three or four different points on the screen. These known LCD locations are compared with the coordinates from the touchscreen controller to generate a translation routine for all new coordinates coming from the touchscreen controller.

There are a variety of translation routines that can be used. The equations below are for a simple linear transformation.

Equation 2:

$$Xdisplay = A(Xts) + B(Yts) + C$$
$$Ydisplay = D(Xts) + E(Yts) + F$$

Xts and Yts are the touchscreen controller’s measured x,y coordinates. When multiplied by the coefficients, they yield the x, y coordinates of the display, X display and Y display. To solve for the six coefficients, A – F, three readings must be taken at known display and touchscreen coordinates, resulting in six equations and six unknowns. After solving for and determining the six coefficients, any coordinates measured by the touchscreen controller can easily be converted into display coordinates.

Many operating systems (for example Linux or WinCE) commonly found on the main system processor are already configured to handle this calibration. They solve for and apply the coefficients to the raw data that comes from the touchscreen controller.

The sample code provided does implement the translation equations. A simple protocol is implemented to receive the six coefficients (12 bytes of data) and store them in the NVM. When this area in flash is not empty, the code applies the six coefficients to Equation 2 and sends X display and Y display to the system microprocessor. A separate command is available to read and transmit the values stored for the coefficients. This provides a simple means to verify the touchscreen controller is using the correct coefficient values. A reset command is also provided to reverse the process and force the controller to erase the coefficients and send non-translated data, Xts and Yts. The protocol is defined in Chapter 4.2 Protocol.

The code provided does not have the algorithm to calculate the six coefficients because the main system processor typically does this. If you desire the touchscreen controller to perform this calculation, you need to modify the communication protocol to receive X display and Y display coordinates. You also need to add a function to perform the calculation. The time for the S08 to perform this calculation is in the tens of ms. Because this is a do-once calculation, your system impact is minimal

3.3.3 Normalization

When the controller is configured to send Xts and Yts, it is common to normalize the values. As described in Chapter 2.1 Four-Wire Touchscreen Interface, this is the process of multiplying by the “height of the panel” parameter in Equation 1. Equation 2 can be used to normalize the output data. By setting coefficients A and E to a normalized value and setting B, C, D, and F to zero, the transmitted data is normalized to whatever scale desired. For example, by setting A and E to 1024 the results are converted back to a full 10-bit A/D scale. You can normalize the output to match the resolution of your display. By setting A = 800 and E = 600, the results are normalized to an SVGA screen. The sample code defaults to 1024 x 1024.

3.4 Coordinate Refresh Rate

Under a pen-down condition, the touchscreen controller continuously reads and transmits the coordinates. How often the coordinate data is sent to the main system processor depends on system requirements. If too slow, the delayed response is noticed by the end user. If too fast the main processor spends too much time handling unnecessary interrupts impacting its performance. The application code provided sends data to the main processor every 10 ms. This timer also controls the main loop in a pen-down condition. When the timer elapses, data is transmitted to the main system processor and the timer is restarted. This control time is defined by the variable TX_interval, found in control_constants.h. The current setup takes 2.5 ms to measure and calculate the x, y coordinates. This time needs to be taken into account if a faster interval is required. The baud rate of the SCI may also need to be considered if a faster interval is chosen.

4 Communication

This section describes the method used to communicate to the main system processor.

4.1 Serial Communication Interface (SCI)

The sample application uses the SCI for communication. The default baud rate is 19.2 KBaud. Code is also in place for 9600 baud and 38.4 Kbaud. These values can be selected in the comm.h file. The 9S08QE family also has an SPI and I^{2}C port. The code is not set-up to support these protocols but can easily be added.

The 9S08QE family was chosen because it has the ability to be woken by the SCI. This allows for a full two-way communication protocol. The main system microprocessor can send commands at any time and the 9S08QE wakes and responds. If this capability is not required in your system, even lower cost 9S08s or 9RS08s may be suitable for your needs.

4.2 Protocol

This section describes the simple communication protocol implemented in the sample code. The protocol must be modified to match your specific system needs.

4.2.1 Transmitting

The uC sends four bytes to the main uP at specific intervals as described in Chapter 3.4 Coordinate Refresh Rate.

Table 1.

		Bit							
		7	6	5	4	3	2	1	0
Byte	0					Pen_up		Align_c	MinMax_c
	1	X7	X6	X5	X4	X3	X2	X1	X0
	2	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0
	3	Y11	Y10	Y9	Y8	X11	X10	X9	X8

Byte 0—This is a control byte that provides information to the main uP about the coordinates it is receiving.

Bit 0 – MinMax_c—This bit tells whether the control provides coordinates that have had losses due to wiring calibrated out.
0—Losses are not calculated and removed.
1—Losses are removed. (This is automatic in the sample code provided. Slight modifications need to be made to allow for no calibration.)

Bit 1 – Align_c—This bit tells whether the control provides coordinates that have calibrated out the misalignment between the touchscreen panel and LCD panel.
0—Misalignment is not accounted for
1—Misalignment is accounted for
Bit 3 – Pen_up—This bit indicates a Pen_up condition exists. It informs the main uP that this is the last data point being transmitted.
0—Pen-down condition still exists
1—A pen-up condition was detected. This is the last transmitted coordinate until a new pen-down event occurs.

Byte 1—The lower 8 bits of the X coordinate.

Byte 2—The lower 8 bits of the Y coordinate.

Byte 3—The lower nibble contains the upper 4-bits of the x coordinate in a 12-bit resolution system. Only the lower two bits are used in a 10-bit resolution system. This nibble is not used for an 8-bit resolution system. The same situation holds for the upper nibble and the y coordinate. The sample code uses a 10-bit resolution.

4.2.2 Receiving

If the system's main microprocessor is going to manage the translation calibration as defined in Chapter 3.3.2 Transmitting, you may not need to receive any data at all. SERIAL_RX_SUPPORTED defined in control_constants.h allows you to bypass the Receive functions in the main loop resulting in more time debouncing the pen-down condition. You can also move to a lower end S08 or RS08.

The clock source for the SCI is stopped when the 9S08QE is in stop mode. The clock takes time to restart when coming out of the stop mode. As a result, the first incoming byte from the main processor wakes the uC, but the data read may be inaccurate. Consequently, this protocol requires that a dummy byte be sent first by the system uP. The following byte should then be one of the control bytes as described below.

NOTE

After being woken, the sample code sits in the RX interrupt routine waiting for the second byte to arrive. The second byte comes immediately from the main microprocessor. If there is a significant delay, or you are manually inputting the data into a terminal window, the QE's internal watchdog timer might expire and reset the QE. The QE then restarts and functions as normal, but the transmitting of data from the host controller needs to start over.

The data is sent in the following order:

Byte 0—0x69— initiates the command

Byte 1—Coefficient A high byte

Byte 2—Coefficient A low byte

Byte 3—Coefficient B high byte

Byte 4—Coefficient B low byte

Byte 5—Coefficient C high byte

Byte 6—Coefficient C low byte

Byte 7—Coefficient D high byte

Byte 8—Coefficient D low byte

Byte 9—Coefficient E high byte

Byte 10—Coefficient E low byte

Byte 11—Coefficient F high Byte

Byte 12—Coefficient F low Byte

No checksum or error checking is done in the communication protocol. The " v " command is implemented instead. When the control sees " v ", it transmits the values stored in the flash. The data is sent in the order shown above; starting with Byte 1 up through Byte 12. This command verifies the communication protocol and also verifies the flash writing routines.

5 MC9S08QE8 Pin-outs

Figure 5 shows the connections between the 16-pin QE8 and an eight-wire touchscreen as used in the sample code. Pin 6, PTB6 is labeled as a GPIO. This pin is used in the sample code as an output and controlled by the PTBD_PTBD6 variable. An oscilloscope was attached to this pin and used for debugging and to measure the timing of the various functions in the code.

Pins 3 and 4 are power and ground. Pin 1 is for an external reset. It can be used as a general purpose input, if no external reset is needed. Pin 2 is for background debugging and programming. This pin can also be used as a general purpose output. Pin 5 is another unused GPIO.

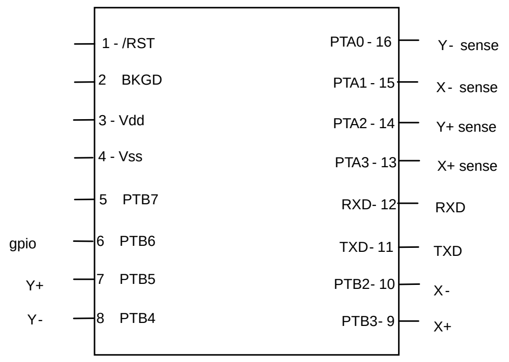

Figure 5. Touchscreen connections to the QE8

6 Sample Code Flowchart

This is the flowchart for the main loop in the sample code.

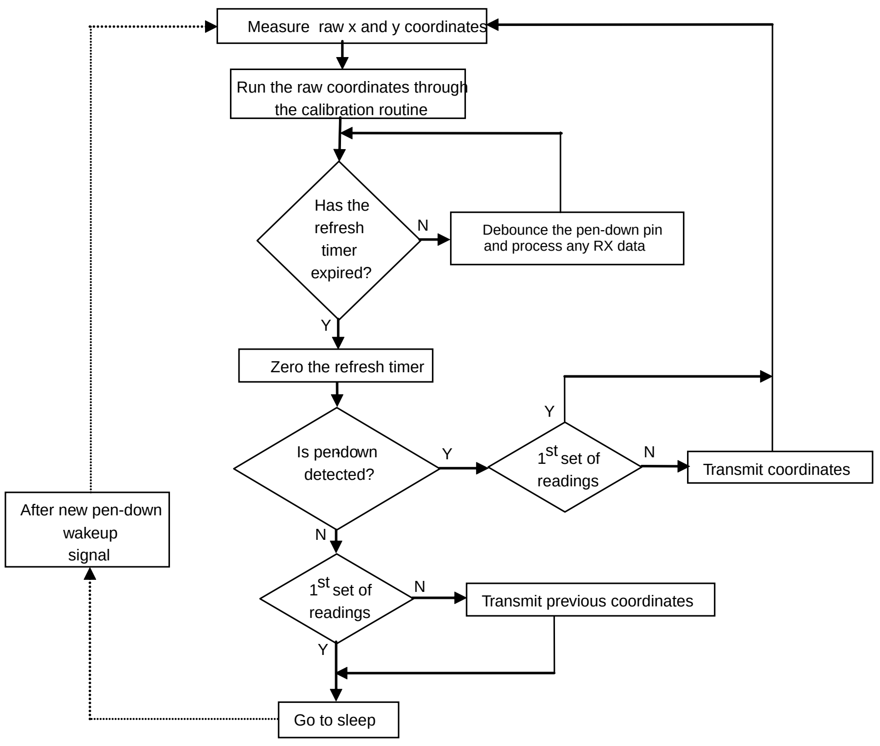
Measure raw x and y coordinates
Run the raw coordinates through the calibration routine
Has the refresh timer expired?
N
Debounce the pen-down pin and process any RX data
Y
Zero the refresh timer
Is pendown detected?
Y
1st set of readings
Y
N
Transmit coordinates
N
1st set of readings
N
Transmit previous coordinates
Y
Go to sleep
After new pen-down wakeup signal

Figure 6. Flowchart

7 Reference Material

MC9S08QE8 Datasheet—Freescale Website

MC9S08QE8 Reference Manual—Freescale Website

8 Acronyms

ADC—Analog digital converter

A/D—Analog digital converter

COP—Computer operating properly

GPIO—General purpose input/output

ISR—Interrupt service routine

KBI—Key board interrupt

NVM—Non-volatile memory (flash)

QG8— MC9S08QG8 microcontroller

SCI—Serial communication interface – aUART port

V_{dd}—Power rail – typically 3.3 Volts

V_{ss}—Ground rail

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

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or +1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN4057
Rev. 0, 3/2010