Freescale Semiconductor
Application Note

Document Number:AN4875
Rev 0, 02/2014

Getting Started with S08RN60

by: Osvaldo Romero

Contents

1 Overview

This application note is an introduction to the S08RN MCU family for automotive solutions. This document gives an explanation of the main modules of the S08RN family so that a project can be started.

The application note is divided into following sections; Introduction, Watchdog (WDOG), Internal Clock Source (ICS), Serial Communications Interface (SCI), FlexTimer Module (FTM), Analog-to-digital converter (ADC), Keyboards Interrupts (KBI), and Touch Sense Input (TSI). Each section contains a brief description of each module together with initialization steps and example code. Also a software project for each module is included.

The software included in this application note was developed in TWR-S08RN60 board available at freescale.com/s08rn.

After reading this application note, you will be familiar with the new 8-bit MCU for automotive applications. Also you will be able to start a project with a S08RN60 MCU.

2 Introduction

This section gives an introduction to the main features and advantages of the new family of 5 V 8-bit S08RN MCU.

The S08RN MCU family is a low-cost, high-performance 8-bit microcontroller unit based on a HCS08 central processor unit. The S08RN offers an easy migration from existing 8-bit S08 product because of its pin compatibility and wide range of memory/package options. The family ranges from 16 pins to 64 pins and from 16 KB to 60 KB of flash memory.

The S08RN family has the following features.

- Based on a HCS08 central processor unit
- Up to 20 Mhz of bus frequency
- 16-bit Watchdog (WDOG)
- Cyclic redundancy check for error detection
- Up to 60 KB of Flash memory
- Up to 4 KB of Ram memory
- 256 bytes of EEPROM
- Up to three LIN/SCI channels
- One IIC channel
- Up to two SPI channels
- One Analog Comparator (ACMP)
- Touch sensing input capability with up to 16 channels
- Up to 16 ADC channels with 8, 10 and 12-bit resolution
- Two 8- bit modulo timers
- Up to eight keyboard interrupt pins
- 16-bit Real Time Counter (RTC)
- Three 16-bit FlexTimer Module (FTM)

The S08RN MCU family is fully AEC-Q100 qualified suitable for any automotive application with ambient temperatures up to 125 °C. This new 8-bit family of MCUs is suitable for any kind of sensor, actuator, or user interface module in an automotive LIN network.

The example software that is included has been developed on the TWR-S08RN60 board using Codewarrior v10.5. The TWR-S08RN60 and reference manual of S08RN60 MCU are available in freescale.com/s08rn.

3 Watchdog (WDOG)

The purpose of this section is to explain how to disable the watchdog timer. In all labs of this application note, the watchdog timer is disabled. Steps and example code to disable the watchdog are given in the following sections.

3.1 Description

Before starting to work on the modules of the MCU, the first thing to know is that the watchdog must be configured or disabled before 128 bus clock cycles after reset. If the watchdog is not disabled during this cycle, it will continuously reset the MCU, unless the watchdog counter is refreshed. By default, the watchdog is enabled after reset. Another consideration for the watchdog is that its registers are write-once. This means that after a write occurs, the registers cannot be modified, unless a reset occurs.

3.2 Disable steps

Perform the following steps to disable the watchdog.

- Unlock the watchdog register to be able to write on it. The unlock sequence is to write the WDOG_CNTH:L register with 0xC520 value followed by 0xD928 value.

- After the watchdog has been unlocked, the next step is to disable the watchdog. The watchdog is disabled by writing a value of 0 to the watchdog control and status registers 1 and 2 (WDGO_CS1 = 0 and WDOG_CS2 = 0).
- Now that the watchdog is disabled, it is necessary to ensure that the watchdog will not generate a reset. Out of reset, the watchdog timeout value register (WDOG_TOVAL) has a value of 0; if this value is not changed, the watchdog will always generate a reset. To avoid a reset from watchdog, write the value of 0xFFFF in the WDOG_TOVAL register.

With these three steps, the watchdog is completely disabled. Remember to do these steps before 128 bus clock cycles after reset.

3.3 Example

The following code is an example for disabling the watchdog.

```
/* First unlock the Watchdog so that we can write to registers */
WDOG_CNT = 0xC520;
WDOG_CNT = 0xD928;

/* Write all 6 registers once within 128 bus cycles after unlocking */
WDOG_CS1 = 0;
WDOG_CS2 = 0;
WDOG_TOVAL = 0xFFFF;
WDOG_WIN = 0x0000;
```

For the software labs that are in the zip file, the function COP_Disable() disables the watchdog using this code. The function can be found in the COP.c file.

4 Internal Clock Source (ICS)

This section describes how to configure the clocks of the S08RN MCU family. At the end of the section, you will be capable to configure the Internal Clock Source module (ICS) to frequency-locked loop engaged external (FEE) operation mode and set a bus speed of 16 MHz. All labs for this application note include a function to configure the bus frequency to 16 MHz.

In order to select the 8 MHz crystal included in TWR-S08RN60, jumpers J17 and J18 must be connected to pin number 2 and 3 each one.

4.1 Description

Before explaining the ICS module, a brief description of the system clock distribution is given. The S08RN MCU family contains the following three on-chip clock sources.

- Internal clock source module (ICS) - The main clock source generator providing bus clock and other references clocks to peripherals.
- External oscillator module (XOSC) - Provides reference clock to ICS, the real time counter (RTC), and other MCU sub-systems.
- Low-power oscillator module (LPO) - Provides 1 kHz reference clock to RTC and watchdog (WDOG).

The ICS module provides a clock source option for the MCU. This module contains a frequency-lock loop (FLL) as a clock source that is controllable by an internal or external reference clock. The module can provide the internal reference clock or the FLL clock as a source for the MCU system clock, ICSCLK. Whichever clock source is chosen, ICSCLK is the output from a bus clock divider (BDIV), which allows a lower frequency to be derived.

The ICS has seven operation modes; FEI, FEE, FBI, FBILP, FBE, FBELP, and stop. The following diagram shows the seven states of the ICS. In this diagram, the arrows indicate the allowed transitions between the states and also show the conditions to enter in that mode.

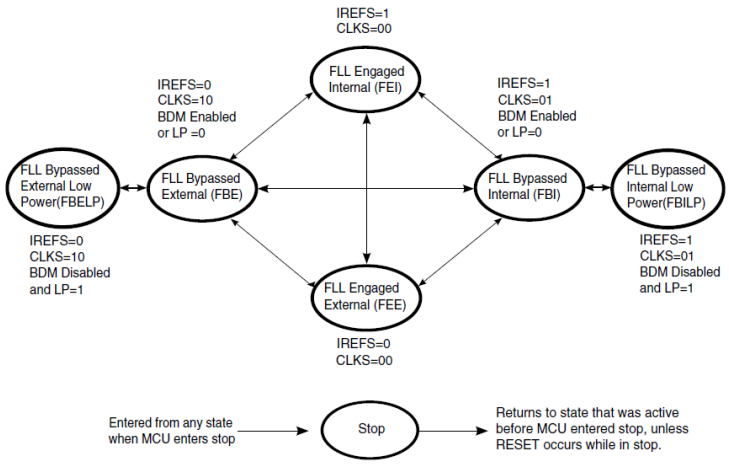

Figure 1. ICS clocking switching modes

For this application note, the ICS is configured to FEE operation mode. The bus is configured to 16 MHz using an external oscillator of 8 MHz.

The FEE operation mode is entered under the following conditions.

- When the clock source select is FLL on the ICS control register 1 (ICS_C1 [CLKS] = 0b00)
- When an external reference clock is selected in the ICS control register 1 (ICS_C1 [IREFS] = 0b0)
- When the reference divider bit is written in the ICS control register 1(ICS_C1 [RDIV]) to divide the external oscillator to be within the range of 31.25 kHz to 39.0625 kHz

The FLL loop locks the frequency to 512 times the external reference frequency, as selected by the reference divider bit.

4.2 Configuration steps

To configure the ICS to operate in FEE mode with a bus frequency of 16 MHz using an 8 MHz external oscillator, perform the following steps.

- Enable the external oscillator by setting the OSCEN bit in OSC status and control register (ISC_OSCSC [OSCEN] = 1).
- Choose the frequency range of the main oscillator. In this case, select high frequency range of 4-20 MHz by writing a 1 to RANGE bit in the OSC status and control register (ISC_OSCSC[RANGE] = 1).
- Write a 1 to the OSCOS bit in the OSC status and control register (ISC_OSCSC[OSCOS] = 1) to select the oscillator as the output clock of OSC module.
- Wait for the oscillator to initialize. The oscillator initialization completes until the OSCINT bit is set in OSC status and control register (ISC_OSCSC [OSCINIT] = 1).
- Select a frequency reference divider for the oscillator by writing a value to RDIV bit in ICS control and status register 1 (ICS_C1 [RDIV]).Choose a divider of 128, so the external oscillator will be divided by 128. This division gives a result of 31.25 KHz. Remember that the result of this division must be between 31.25 KHz and 39.0625 KHz.
- Change FLL reference clock to external clock by writing a 0 to IREFS bit in ICS control and status register 1(ICS_C1[IREFS] = 0).
- Wait for the synchronization of IREFST bit in ICS status register (ICS_S [IREFST]). This bit indicates if the current source for the reference clock is external or internal. In this case, wait to be external (ICS_S [IREFST] = 0).

- Wait until FLL is unlocked. A 1 in the LOCK bit at the ICS status register indicates that FLL is locked (ICS_S [LOCK]). The FLL loop locks the frequency to 512 times the external reference frequency. Now the clock is running at 16 MHz.
- Select a bus prescaler by writing a value to BDIV in ICS control and status register 2 (ICS_C2 [BDIV]). In this case, choose a prescaler of 1 by writing 0 to the bit.
- Clear the loss of lock sticky bit. Write a 1 to the LOLS bit in ICS status register (ISC_S [LOLS] = 1).

4.3 Example

The following code is an example to configure the BUS at 16 MHz.

```
/* Enable external Oscillator */
ICS_OSCSC_OSCEN = 1;

/* Main oscillator between 4 and 20 MHz */
ICS_OSCSC_RANGE = 1;

/* Using oscillator output as clock */
ICS_OSCSC_OSCOS = 1;

/* Wait for oscillator to initialize */
while(!ICS_OSCSC_OSCINIT);

/* Divide the frequency over 256.  8000/256 = 31.25 KHz*/
ICS_C1_RDIV = 3;

/* Change FLL reference clock to external clock */
ICS_C1_IREFS = 0;

/* Wait for the IREFS write to synch */
while(ICS_S_IREFST);

/* Wait for PLL lock, now running at 16 MHz (512 * 31.25) */
while(!ICS_S_LOCK);

/* No prescaler, so clock is still 16 MHz*/
ICS_C2_BDIV = 0;

/* Clear Loss of lock sticky bit */
ICS_S_LOLS = 1;
```

For the software labs that are in the zip file, the function CLK_Init () initialize the clock to run at 16 MHz using the 8 MHz external oscillator. The function can be found in the CLK.c file.

5 Lab 1 : Serial Communications Interface (SCI)

This section demonstrates how to configure and use the serial communications interface (SCI) module to receive and send characters, using hardware interrupts and software polling. A software lab is included to demonstrate the SCI functionality, TWR-S08RN60_LAB1. This first lab uses the OSBDM port to communicate via serial communication interface 2 on the RN60 MCU. This lab waits for characters sent from a PC terminal and echoes the received data at a baud rate of 9600, any PC terminal can be used. At the end of the section, you will be able to configure the SCI module to receive and send data.

In order to use the OSBDM of the TWR-S08RN60 to send and receive data via SCI2, jumpers J1 and J2 must have connected to pin number 1 and 2.

5.1 Description

The S08RN60 includes three SCI channels. Some of the SCI module features are full-duplex, standard non-return-to-zero (NRZ) format, double-buffered transmitter and receiver with separate enable, interrupts for transmit data register empty and receive data register full, programmable baud rates, and programmable 8-bit or 9-bit character length.

The clock source for the SCI baud rate generator is the bus-rate clock. The following figure shows how to calculate the baud rate generation.

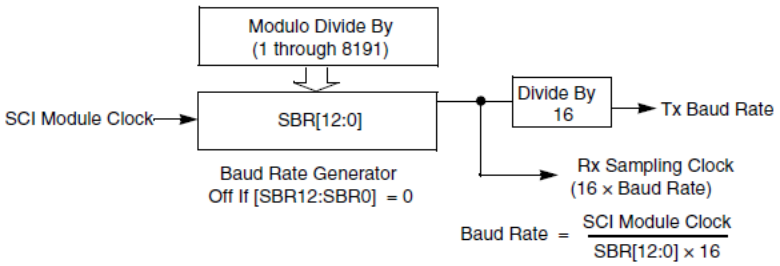

Figure 2. SCI baud rate generation

To enable the transmitter in SCI, the TE bit in SCI control register 2 (SCIx_C2 [TE]) must be set. The transmitter remains idle until data is available in the transmit buffer. Programs store data into the transmit data buffer by writing the SCI data register (SCI_D). Use hardware interrupt or software polling to send data through the SCI module.

To enable the transmit interrupt, set the TIE bit in SCI control register 2 (SCIx_C2 [TIE]). Transmit data register empty (SCI_S1 [TDRE]) indicates when there is room in the transmit data buffer to write another transmit character to SCI_D. Every time that SCI_S1 [TDRE] is set an interrupt will be requested.

Every time a data transmit is completed, a hardware interrupt can be requested. To enable this interrupt, the transmit complete interrupt enable bit (SCI_C2 [TCIE]) must be set. The transmit complete bit (SCI_S1 [TC]) indicates that the transmitter has finished transmitting all data, when this bit is set, a hardware interrupt will be generated.

Instead of hardware interrupts, software polling may be used to monitor the transmit data register empty SCI_S1 [TDRE] and the transmit complete SCI_S1 [TC] status flags if the corresponding interrupts flags, SCI_C2 [TIE] and SCI_C2 [TCIE] are clear.

To enable the receiver in SCI, the RE bit in SCI control register 2 (SCI_C2 [RE]) must be set. Character frames consist of a start bit of logic 0, eight (or nine) data bits (lsb first), and one (or two) stop bits of logic 1. When the receive data register is full (SCI_S1 [RDRF] = 1), it gets the data from the receive data register by reading SCI_D. There are two ways that user can manage received data, software polling and hardware interrupts.

To use the receive interrupt the enable RIE bit in SCI control register 2 (SCIx_C2 [RIE]) must be set. Every time that the receive data register (SCIx_S1 [RDRF] = 1) is full, a hardware interrupt will be requested. At the hardware interrupt service routine (ISR), the SCIx_S1 [RDRF] flag is cleared by reading SCIx_S1 register while SCIx_S1 [RDRF] is set and then reading SCI_D.

When using polling, this sequence is naturally satisfied in the normal course of the user program. At polling, the SCIx_C2 [RIE] bit must be clear.

5.2 Initialization steps

To initialize the SCI module follow the next steps.

- Select the number of stop bits for serial communication by setting /clearing the stop bit number select in SCI baud rate register: high (SCIx_BDH [SBNS]). Clearing the bit will select one stop bit, setting the bit will select two stop bits.
- Select the desired baud rate by writing a value in SCI baud rate register (SCIx_BD). Remember to use the equation in Figure 2.

- Select 8-bit/9-bit format by clearing /setting the mode select bit in SCI control register 1 (SCIx_C1 [M]). Clearing the bit will select 8-bit format and setting the bit will select 9-bit format.
- Enable/ disable parity bit in SCI control register 1 (SCIx_C1 [PE]).
- Enable transmitter and receiver by setting transmit enable bit and receive enable bit in SCI control register 2 (SCIx_C2 [TE] = 1 and SCIx_C2 [RE] = 1).
- If transmit and/or receive interrupts are desired, enable them by setting the transmit interrupt enable bit, transmission complete interrupt enable bit, and receiver interrupt enable bit in SCI control register 2 (SCIx_C2 [TIE] = 1,SCIx_C2 [TCIE] = 1 and SCIx_C2 [RIE] = 1).

The MCU interrupts must be enabled by calling the EnableInterrupts macro after configuring the SCI, only if interrupts will be used.

5.3 Example

The following code shows how to initialize the SCI module. In this code, a baud rate of 9600 is selected, one stop bit is used, 8-bit format is selected, parity is disabled, and receiver interrupt is enabled.

```
void SCI_Init()
{
/* One stop bit only */
SCI2_BDH_SBNS = 0;
/* Select the baud rate, BD_Divider = BUS_CLK/16/BAUDRATE = 16M/16/9600 = 104 */
/* 8bit mode, 1 stop bit, no parity */
SCI2_C1 = 0;
/* Enable TX */
SCI2_C2_TE  = 1;
/* Enable RX */
SCI2_C2_RE = 1;
/* Enable reception interrupt */
SCI2_C2_RIE = 1;
}
```

The code above shows how to send data using polling. In this case, the number 1 will be sent.

```
/* Wait for transmit buffer to be empty*/
while(!( SCI2_S1 & SCI2_S1_TDRE_MASK));
/* Read SCI2_S1 register*/
(void)SCI2_S1;
/* Write the data you want to send*/
SCI2_D ='1';
```

The following code shows how to handle received data at the interrupt service routine.

```
interrupt VectorNumber_Vsci2rx void SCI_RX_ISR()
{
UINT8 data=0;
/* Clear reception flag mechanism. */
SCI2_S1_RDRF = 1;
(void) SCI2_S1;
/* Receive data */
data = SCI2_D;
}
```

In the TWR-S08RN60_LAB1 project, the SCI.c and SCI.h files include the configuration, send, receive, and interrupt functions for the serial communication.

6 Lab 2: FlextTimer Module (FTM)

The purpose of this lab is that you learn how to configure the FlexTimer Module (FTM) as an output compare to generate periodically interrupts at desired frequencies. A software lab is included as an example to demonstrate the capabilities of the FTM. This second lab, TWR-S08RN60_LAB2, toggles the four LEDs included in the TWR-S08RN60 at a different frequencies, which is configurable using the FTM.

6.1 Description

The FlexTimer Module is a timer that supports input capture, output compare, and generation of PWM signals to control electric motors and power management applications. The FTM timer is a 16-bit counter that can be used as an unsigned or signed counter.

In output compare mode, the FTM can generate timed pulses with programmable position, polarity, duration, and frequency. When the counter matches the value in the channel value (CnVH and CnVL) registers of an output compare channel, an interrupt is generated.

6.2 Configuration steps

To configure the FTM as an output compare, perform the following steps.

- Select a prescaler for the FTM channel by writing a value to the prescale factor selection bit in FTM status and control register (FTMx_SC [PS]). Remember that this bit can be written only when FTMx_MODE[WPDIS] = 1. This value will determine the frequency at which the 16–bit FlexTimer counter will increase by one.
- Enable the interrupt for the FTM channel by setting the channel interrupt enable bit in FTM channel status and control register (FTMx_CnSC [CHIE] = 1).
- Select the channel operation mode. For output compare mode, write a 1 at the channel mode select MSA bit in FTM channel status and control register (FTMx_CnSC [MSA] = 1).
- To select the frequency of the interrupt, write a value in FTM channel value register (FTMx_CnV). Every time the FlexTimer counter reaches this value, an interrupt will be generated.
- Finally select the clock source selection in FTM status and control register (FTMx_SC [CLKS]). Remember that this bit is write protected, it can be written only when MODE [WPDIS] = 1.

After configuring the FTM, the interrupts of the MCU must be enable. To enable interrupts in the MCU, call the EnableInterrupts macro.

At the FTM interrupt service routine (ISR), the following steps must be done.

- Clear the interrupt channel flag by reading FTM channel status and control register (FTMx_CnSC) while channel flag is set and then writing a 0 to the channel flag bit in FTM channel status and control register (FTMx_CnSC [CHF] = 0).
- Refresh the frequency of the interrupt in the Channel value register (FTMx_CnV).
- Write desired code.

6.3 Example

The following code shows an example to configure the FTM2 channel 0 as an output compare to generate an interrupt every 131 ms.

```
/* Clock divided over 16. This means 16 MHz /16 = 1 MHz. The 1 MHz clock is divided by a
prescaler of 2  because MODE [FTMEN] = 0  out of reset, so our system clock 1 MHz /2 = 500
KHz, this means the counter will increase every 1/500 KHz = 2uS */
FTM2_SC_PS = 0b100;

/* Interrupt enable */
FTM2_C0SC_CHIE = 1;

/* Channel as Output Compare */
FTM2_C0SC_MSA = 1;

/*Configure FTM channel 0 to toggle each 65535x2uS = 131070 us=131 ms*/
FTM2_C0V = 0xFFFF;

/* Select the system clock */
FTM2_SC_CLKS = 1;
```

The following code shows how to handle the interrupt service routine of the FTM module. In this case, the interrupt flag is cleared, the new timeout is set, and the LED at port G0 is toggled.

```
interrupt VectorNumber_Vftm2ch0 void FTM2_CH0_ISR()
{
/* Clear flag mechanism */
(void)FTM2_C0SC;
FTM2_C0SC_CHF = 0;

/* Set new timeout*/
FTM2_C0V = FTM2_C0V + 0xFFFF;

/* Led Toggle at port G0*/
PORT_PTGD_PTGD0 ^= 1;
}
```

In the TWR-S08RN60_LAB2 project, the FTM.c and FTM.h files include the configuration and interrupt functions to toggle the 4 LEDs.

7 Lab 3: Analog-to-digital converter (ADC)

Through this third lab, you will learn how to initialize the Analog-to-digital converter (ADC) of the S08RN family. A software lab is included, TWR-S08RN60_LAB3, where the ADC is configured to read the voltage of the potentiometer included in the TWR-S08RN60. The ADC value is displayed in a binary form in the 4 LEDs that are included with the board.

In order to read the voltage of the potentiometer at ADC channel 0, jumper J7 must have pins 1 and 2 connected.

7.1 Description

The 12-bit analog-to-digital converter is a successive approximation ADC designed for operation within an integrated microcontroller system-on-chip. The ADC has up to 16 external analog inputs, external pin inputs, and five internal analog inputs including internal bandgap, temperature sensor, and references.

The ADC can perform an analog-to-digital conversion on any of the software selectable channels. In 12-, 10-, and 8-bit mode, the selected channel voltage is converted by a successive approximation algorithm into a 12-, 10-, or 8-bit digital result.

When a conversion is completed, the result is placed in the 8-bit data registers (ADC_RH and ADC_RL). The conversion complete flag (ADC_SC1 [COCO]) is then set and an interrupt is generated if the conversion complete interrupt has been enabled (ADC_SC1 [AIEN] = 1).

7.2 Initialization steps

To initialize the ADC, perform the following steps.

- Update ADC status and control register 3 (ADC_SC3) to select the input clock source and the divide ratio used to generate the internal clock, ADCK.
- Select hardware or software conversion trigger by setting or clearing the conversion trigger select bit in ADC status and control register 2 (ADC_SC2 [ADTRG]).
- Enable or disable compare function by setting or clearing its bit in ADC status and control register 2 (ADC_SC2 [ACFE]).
- Update ADC status and control register 1 (ADC_SC1) to select whether conversions will be continuous or completed only once, and to enable or disable conversion complete interrupts.
- Enable the ADC pin by writing a 1 to the desired ADC pin control bit in ADC pin control register 1 or 2 depending on the desired pin (ADC_APCTL1, ADC_APCTL2). The pin control registers disable the I/O port control of the pins used as analog inputs.
- Select the resolution of the ADC by writing a value to the MODE bit in ADC status and control register 3 (ADC_SC3 [MODE]). By default 8-bit resolution is selected.

To read the ADC values, perform the following steps.

- Set the channel to read, by writing a channel value to the input channel select bits in ADC status and control register 1 (ADC_SC1[ADCH]).
- Wait for conversion to complete. When the conversion complete flag bit is set at ADC status and control register 1 (ADC_SC1 [COCO]), user can read the ADC conversion result.
- Read ADC conversion result registers (ADC_R).

7.3 Example

The following example shows how to initialize the ADC and read the resulting value of the conversion.

```
void ADC_Init()
{
/* Bus clock selected*/
ADC_SC3_ADICLK = 0;
/* Divide ratio = 1, clock rate = Input clock */
ADC_SC3_ADIV = 0;
/* Software trigger*/
ADC_SC2_ADTRG = 0;
/* Disable compare function*/
ADC_SC2_ACFE = 0;
/* No continuous conversion*/
ADC_SC1_ADCO = 0;
/* Disable conversion complete interrupt*/
ADC_SC1_AIEN = 0;
/* Enable ADC pin 0 */
ADC_APCTL1_ADPC0 = 1;
/* 8-bit resolution*/
ADC_SC3_MODE =0x00;
}
UINT8 ADC_Read(UINT8 ch)
{
/* Set the channel to read, trigger ADC */
ADC_SC1_ADCH = ch;
/* While conversion has not been completed */
while(!ADC_SC1_COCO);
/* Return ADC read */
return ADC_RL;
}
```

In the TWR-S08RN60_LAB3 project, the ADC.c and ADC.h files include the initialization and read functions for the ADC.

8 Lab 4: Keyboard Interrupts (KBI)

This fourth lab demonstrates how to initialize and use the Keyboard Interrupts (KBI). A software lab is included, TWR-S08RN60_LAB4. In this lab, the two buttons, that are included in the TWR-S08RN60, are used to toggle two LEDs using KBI.

8.1 Description

This on-chip peripheral module is called a keyboard interrupt module because it was originally designed to simplify the connection and use of row-column matrices of keyboard switches. However, these inputs are also useful as extra external interrupt inputs and as an external means of waking the MCU from stop or wait low-power modes.

Some of the features of the KBI are up to eight keyboard interrupt pins with individual pin enable bits, one software-enabled keyboard interrupt, enable pullup resistor, independent edge sensitivity selection, and exit from low-power modes.

8.2 Initialization steps

To initialize the keyboard interrupts, perform the following steps.

- Mask keyboard interrupts by clearing the interrupt enable bit in KBI status and control register (KBIx_SC [KBIE] = 0).
- Choose the KBI polarity by writing the edge select bit in KBI edge select register (KBIx_ES [KBEDG]).
- Configure the associated bits in PORT_PTxPE. Remember that the pin selected must have the KBI functionality. Enable internal pull up resistor if necessary.
- Enable the Keyboard interrupt channel by setting the pin enable bit in KBI pin enable register (KBIx_PE [KBIPE] = 1).
- Clean all flags by setting the KBI acknowledge bit in KBI status and control register (KBIx_SC [KBACK] = 1).
- Enable Keyboard interrupts by setting the interrupt enable bit in KBI status and control register (KBIx_SC [KBIE] = 1).

After initializing the KBI, the interrupts of the MCU must be enabled. To enable interrupts in the MCU, call the EnableInterrupts macro.

For the interrupt, perform the following steps.

- Clear the interrupt flag and set the acknowledge bit in KBI status and control register (KBIx_SC [KBACK] = 1).
- Read the input associated to the KBI channel.

8.3 Example

The following code shows how to initialize the KBI interrupts in channel 4 with edge polarity 0. At the interrupt, a LED in port G1 is toggled.

```
void KBI_Init()
{
/* Mask interrupts*/
KBI1_SC_KBIE   = 0;
/*Falling edge*/
KBI1_ES_KBEDG4 = 0;
/* Button pins as inputs */
```

```
PORT_PTDIE_PTDIE4 = 1;
/* Enable KBI channel 4 (SW3) */
KBI1_PE_KBIPE4 = 1;
/* Erase previous flags */
KBI1_SC_KBACK = 1;
/* Enable KBI Interrupt */
KBI1_SC_KBIE = 1;
}
interrupt VectorNumber_Vkbi1 void KBI_ISR()
{
/* Clear flag*/
KBI1_SC_KBACK = 1;
/* Check if the input in channel 4 has been pressed*/
if(!PORT_PTDD_PTDD4)
{
/* Toogle led at port G1*/
PORT_PTGD_PTGD1 ^= 1;
}
}
```

In the TWR-S08RN60_LAB4 project, the KBI.c and KBI.h files include the initialization and interrupt functions for KBI.

9 Lab 5: Touch Sense Input (TSI)

In this lab, you will learn how to initialize the TSI and scan electrodes to determine if a touch event has occurred. The software for this lab is TWR-S08RN60_LAB5. This lab uses the 4 electrodes that are included in the TWR-S08RN60. Every time an electrode has been touched, its led will turn on.

9.1 Description

The TSI module provides capacitive touch sensing detection with high sensitivity and enhanced robustness. Each TSI pin implements the capacitive measurement by doing a current source scan and charging and discharging the electrode, once or multiple times. A reference oscillator ticks the scan time and stores the result in a 16-bit register when the scan completes.

The TSI provides a solid capacitive measurement module to the implementation of touch keyboard, rotaries and slider. It supports up to 16 external electrodes, is configurable up to 4096 scan times, and has the capability to wake MCU from stop3 mode.

The TSI fully support Freescale touch sensing software (TSS) library, see freescale.com/touchsensing. The TSS library is an innovative touch-sensing software library that offers differentiated features such as water tolerance, noise detection, and touch detection algorithm for reduced false touches under electrical noised.

The TSI module uses two oscillators, the electrode oscillator and the reference oscillator to detect a touch.

In the electrode oscillator, the electrode pin capacitance measurement uses a dual oscillator approach. The frequency of the TSI electrode oscillator depends on the external electrode capacitance and the TSI module configuration. This external electrode is an area of conductive material.

The TSI electrode oscillator circuit is illustrated in the following figure. A configurable constant current source is used to charge and discharge the external electrode capacitance. A buffer hysteresis defines the oscillator delta voltage. The delta voltage defines the margin of high and low voltage, which are the reference input of the comparator in different time.

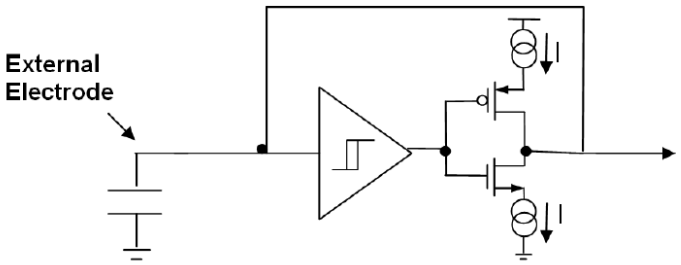

Figure 3. TSI electrode oscillator circuit

The hysteresis delta voltage is defined in the module electrical specifications present in the device data sheet. The following figure shows the voltage amplitude waveform of the electrode capacitance charging and discharging with a programmable current.

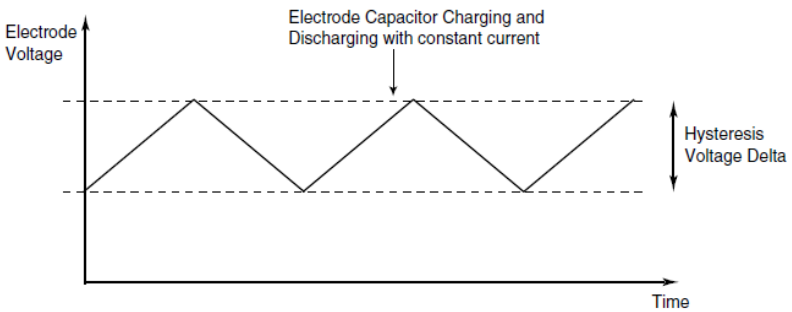

Figure 4. TSI electrode oscillator chart

The TSI reference oscillator has the same topology of the TSI electrode oscillator. The TSI reference oscillator has an internal reference capacitor instead of an external capacitor (electrode). This reference oscillator is configured to be faster than the external oscillator. The reference oscillator counts the number of oscillations per external reference scan.

Every time a touch is detected in the external electrode, the charging and discharging time will be slower, so the number of counts of the reference oscillator increases. By comparing the number counts of the reference oscillator, we can determine if a touch has occurred. See the following figure.

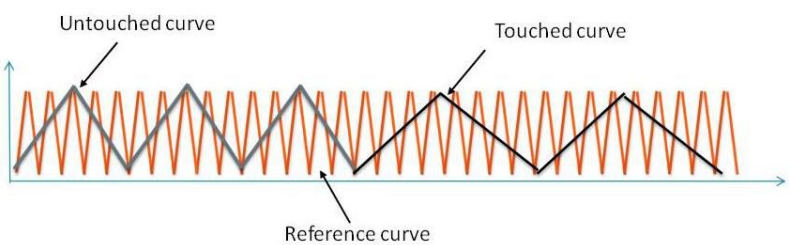

Figure 5. TSI electrode and reference oscillator comparison

9.2 Initialization steps

To initialize the TSI module, perform the following steps.

- Enable the TSI module by setting the TSIEN bit in the control and status register 0 (TSI_CS0 [TSIEN] = 1).
- Enable or disable the TSI interrupts. By setting or clearing the TSIIEN bit in TSI control and status register 0 (TSI_CS0 [TSIIEN]).

- Select the trigger mode by writing to the STM bit in TSI control and status register 0 (TSI_CS0 [STM]). When the bit is clear, the software trigger mode is applied. When this bit is set, hardware trigger mode is applied.
- Select a frequency prescaler by writing a value to PS bit in TSI control and status register 1 (TSI_CS1 [PS]).
- Select the number of samples by writing a value in the NSCN bit in TSI control and status register 1 (TSI_CS1 [NSCN]).
- Select the current source to charge and discharge the reference oscillator. Write a value to the REFCHRG bit in TSI control and status register 2 (TSI_CS2 [REFCHRG]).
- Select the oscillators voltage in DVOLT bit at Control and Status register 2 (TSI_CS2 [DVOLT]).
- Select the current source to charge and discharge the electrode oscillator. Write a value to the EXTCHRG bit in TSI control and status register 2 (TSI_CS2 [EXTCHRG]).
- Enable the pin of the board electrodes by setting the desired pin in TSI pin enable register 1 or 0 (TSI_PENx [PENn] = 1).

To initialize a scan request on an electrode via software trigger, perform the following steps.

- Set the channel to scan by selecting the channel to be measured in the TSICH bit at TSI control and status register 3 (TSI_CS3 [TSICH] = 1).
- Enable the scan by setting the SWTS bit at TSI control and status register 0 (TSI_CS0 [SWTS] = 1).
- Wait for conversion to complete. When the EOSF bit at TSI control and status register 0 is set (TSI_CS0 [ESOF]), the scan has been completed.
- Clear the scan complete flag. Set the ESOF bit (TSI_CS0 [ESOF]=1).
- Read the TSI counter to determine if the electrode has been touched or not (TSI_CNT).

9.3 Example

The following code initializes the TSI module with the four electrodes that are included in the TWR-S08RN60 without using the TSS library.

```
void TSI_Init()
{
/* TSI Enabled */
TSI_CS0_TSIEN = 1;
/* TSI Interrupt disabled */
TSI_CS0_TSIIEN = 0;
/* Software Trigger */
TSI_CS0_STM = 0;
/* Clock prescaler. clock/2^PS */
TSI_CS1_PS = 1;
/* Number of samples. Samples = NCSN+1 (up to 32 samples) */
TSI_CS1_NSCN = 4;
/* Charge current (1 microAmp) */
TSI_CS2_REFCHRG = 1;
/* Charge voltage 0.1-1.33 */
TSI_CS2_DVOLT = 0;
/* External charge current (1 microAmp) */
TSI_CS2_EXTCHRG = 1;
/* Enable the 4 pins of the board electrodes */
TSI_PEN0_PEN0 =  1;
TSI_PEN0_PEN1 =  1;
TSI_PEN1_PEN14 = 1;
TSI_PEN1_PEN15 = 1;
}
```

The following code shows how to initialize a scan request on channel 0.

```
/* Set channel to scan */
TSI_CS3_TSICH = 0;
/* Software trigger */
TSI_CS0_SWTS = 1;
```

```
/* Wait for conversion to finish */
while(!TSI_CS0_EOSF);
/* Clear Flag */
TSI_CS0_EOSF = 1;
/* Check if a touch has occurred*/
if (TSI_CNT >280)
PORT_PTGD_PTGD0 = 0;
else
PORT_PTGD_PTGD0 = 1;
```

In the TWR-S08RN60_LAB5 project, the TSI.c and TSI.h files include the TSI initialization and scan initialization functions.

10 References

The following references are available at freescale.com.

1. S9S08RN60 Reference Manual, Rev 1, 01/2014 available at freescale.com/s08rn
2. AN3041: Internal Clock Source (ICS) Module on the HCS08s in Depth
3. AN4431: TSI module application on the S08PT family

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4875
Revision 0, 02/2014