Freescale Semiconductor
Application Note

Document Number: AN4023
Rev. 0, 7/2010

Enabling the IRTC Module

**by: Alejandra Guzman
Guadalajara
Mexico**

1 Introduction

This document describes a driver for an independent real time clock (IRTC). This allows users the customization of all the possible configurations for this peripheral, like full clock functionalities, calendaring, and auto adjustment for daylight savings, a programmable alarm, and a minute countdown timer.

The driver was tested on the MCF51EM256 and the software architecture was designed to provide seamless migration between devices that have the same peripheral module.

This application note is intended to be used by all software development engineers, test engineers, and anyone else who has to use microcontrollers with IRTC.

To download the software for this application note go to www.freescale.com (AN4023SW)

Contents

2 IRTC Main Features

- Time keeping functions by second, minute, and hour counters
- Calendaring functions via date, day-of-week, month, and year counters
- Alarm set for specific hour, minute, and second.
- Countdown timer with minute resolution
- Daylight savings adjustment
- Leap year automatic adjustment

3 Independent Real Time Clock (IRTC)

The following sections describe the most relevant characteristics of the IRTC.

3.1 IRTC Power Supply

The IRTC power supply source depends on the MCU operation mode and the LVD configuration.

Run, wait mode, or stop4

- If (VDD > VLVDH), the IRTC is powered by VDD pin
- If (VDD < VLVDH) and (VDD > VLVDL), the IRTC is powered by VDD and VBAT pins.
- If (VDD < VLVDL), the IRTC is powered by the VBAT pin

LPRun, LPWait, stop3, or stop 2

- The IRTC always operates from Vbat

3.2 Write Protection Mechanism

A protection mechanism is built-in the IRTC to protect against spurious writes into the IRTC by runaway code.

The protection mechanism requires the CPU to write a specific sequence of codes to the RTC_CTRL [1:0] bits in the control register that allows write access to the registers. Figure 1 shows the sequence needed to disable the write protection.

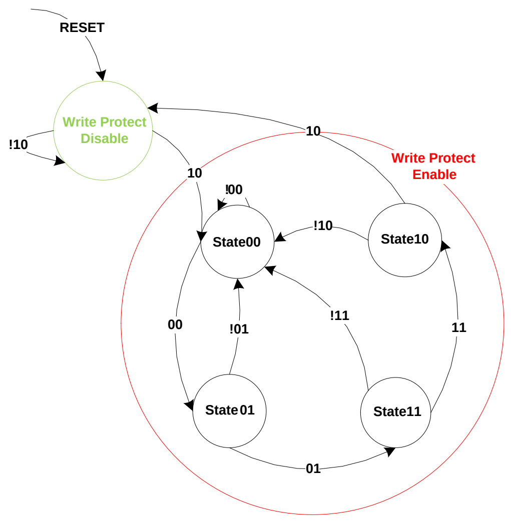

Figure 1. Write protection state machine

Writing the sequence 00 – 01 – 11 – 10 to the WE [1:0] bits at the RTC_CTRL register disables the write protection. To enable write protection, 10 is written on these bits.

After power-on reset, the write-protect mechanism is disabled allowing the user code to calibrate the RTC clock. Set the time in the clock registers, and set the date in the calendar registers.

After calibration of the time and the date settings are done, the user code must enable the write protection mode. If not, the registers are placed in the write protect mode, 15 seconds after power-on. If the write-protect mechanism is unlocked by the software, the write protection mode will be enabled two seconds after the unlock sequence.

3.3 Compensation Logic

A frequency compensation module is integrated into the RTC to correct any error in the 1 Hz clock due to any variations in the 32 kHz clock caused by crystal inaccuracy, board variations, or a change in temperature. The compensation value for both the crystal and temperature variation is set by the software. Corrections are executed in the hardware.

3.4 Tamper Detection

The RTC can detect any intrusion via its tamper detection mechanism which gets enabled automatically after the calibration of the RTC is complete. For this purpose, two pins have been provided and any change of state on these pins indicate a tamper that gets stored in the RTC registers.

4 Software

The software for this application note implements a Real Time Clock in a real environment.

4.1 Application Overview

Figure 2 and Figure 3 provide a high-level overview of how the software works. To start, the MCU initializes peripherals that are to be used through the code like the LCD, IRTC, KBI, and so on.

After the initialization has finished, the application enters an infinite loop that runs a state machine in charge of the display, refreshing each time an IRTC event has been detected, managing the configuration menu, and resetting the independent real-time clock parameters.

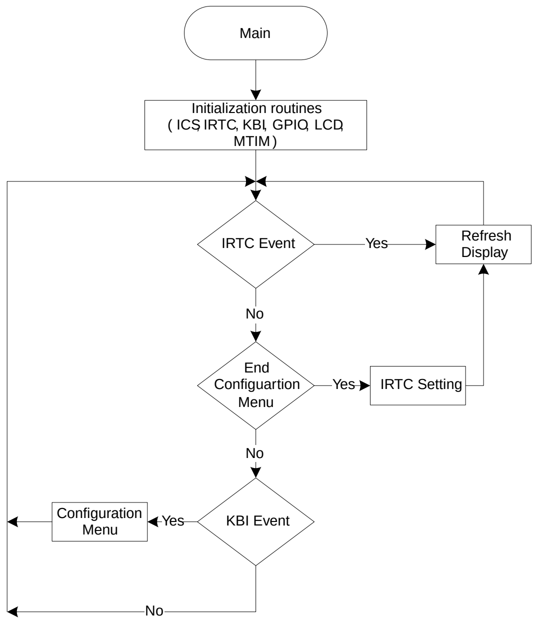

Figure 2. Application flowchart

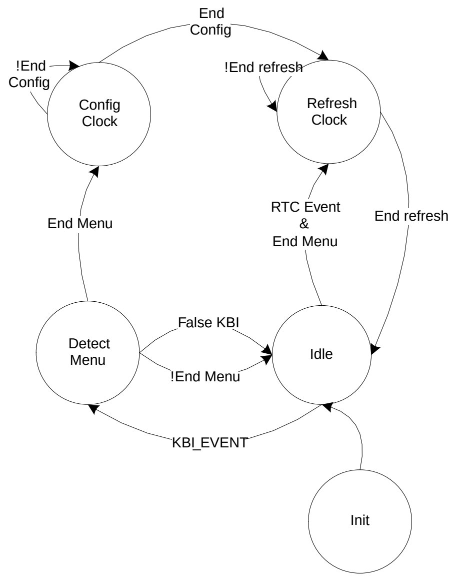

Figure 3. Application state machine

4.2 Software Architecture

The code for this application note is written in such a way that the software modules interacting with the peripherals are independent from the modules that process the information. The modules used as an interface with the MCU peripherals are known as the hardware abstraction layer (HAL).The modules interacting with HAL that pass information to the main application are known as a hardware independent layer (HIL).

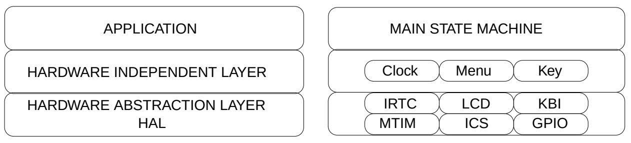

Figure 4. Software layered model

Figure 4 shows a layered model of the structured software.

4.2.1 Hardware Abstraction Layer (HAL)

The HAL is defined as the collection of software components that directly access hardware resources. In this layer, the LCD driver defines macros and functions that configure the custom glass requirements in the LCD module registers.

4.2.2 Hardware Independent Layer (HIL)

The HIL is defined as the collection of software components that access hardware resources through the HAL. Peripheral drivers are implemented in this layer.

5 IRTC Implementation

The user can add IRTC.c and IRTC.h files to their respective applications. To have the IRTC driver up and running, the main function needs to call the initialization function also named vfnIRTC_Init

The IRTC.h, and IRTC.c files provide functions and macros that control any one of the IRTC settings and control register bits. The user can set the _YEAR, _MONTH, _DAY, _HOUR, _MINUTES, and _SECONDS; to their macros, these changes are reflected inside the vfnIRTC_Init function.

6 Software Overview

The software is written in a way that every functionality is divided by blocks, to know the settings for these blocks, look at the following points.

6.1 Keys

This block detects when a key is pressed and determines what key was pressed. This module debounces the KBI event, makes a KBI_PORT reading, and verifies if a valid key has been detected. MTIM and KBI HAL modules are needed to perform this.

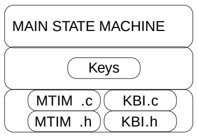

Figure 5. Main state machine keys

The LCD driver used for this project can also be found in the application note titled *LCD Driver Specifications* (document AN3796). This document details how to migrate the software driver provided to a specific custom LCD.

6.2 Menu

The main application determines how to respond with the key previously detected to the Keys module. To have better interaction with the user, the application displays the current configuration on the DEMOBOARD liquid crystal display (LCD).

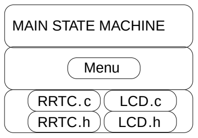

Figure 6. Main state machine menu

6.3 Clock

This module configures the independent real time peripheral with the settings gathered on the Menu module and for refreshing the display each time an IRTC event occurs.

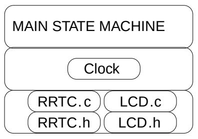

Figure 7. Main state machine clock

7 Conclusion

This driver provides a software base for applications that need implementation of Real Time Clocks. Minimal changes are required to customize the IRTC settings, those changes are explained inside the Appendix application programming interface (API) .

Appendix A Application Programming Interface (API)

A.1 IRTC.h

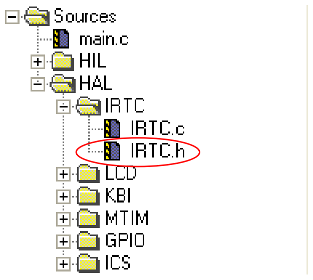

Figure A-1. IRTC.h file

Set Year

Description—This macro calculates the offset in years from the base year (hard coded as 2112). For example, if the current year is 2010, then this will be represented as –102 or 0x9A (signed value)

Name:
  SetYear (Year)

Parameters:
  Year—The value that needs to be related to the base year

Example:
  IRTC_YEARMON_bin_YEAR = SetYear(2010);

Get Year

Description—Converts in a date, the offset in years from the base year (hard coded as 2112). For example, if the current offset is -102, the result of using this macro will be 2010

Name:
  GetYear(offset)

Parameters:
  offset—Offset in years related to 2112

Example:
  Year = GetYear(-102);

_RTC_CLCK_OUTPUT

This macro enables or disables the RTC oscillator clock that is output from the RTC block.

- 1—Clock output enable
- 0—Clock output disable

_RTC_SELECT_CLCK

This macro defines the IRTC module clock that is output for external use.

- 0—No clock is output
- 1—Compensated 1 Hz
- 2—Buffered oscillator clock

_RTC_DAYLIGHT

This macro enables or disables the daylight savings feature.

- 0—Enable daylight
- 1—Disable daylight

_RTC_SW_RESET

This macro clears the content of the alarm and interrupt registers

- 0—No software reset
- 1—Software reset

_RTC_BCD

This macro controls whether the read and write value of time and date registers are in binary or BCD

- 0—No filter operation
- 1 to 15—Clock counts

_RTC_TAMP_FTR

This macro defines the IRTC number of 32 kHz oscillator clocks counted when a tamper event is asserted. These bits are used mainly to generate a tamper filtering operation.

- 0—No filter operation
- 1 to 15—Clock counts

A.2 IRTC.c

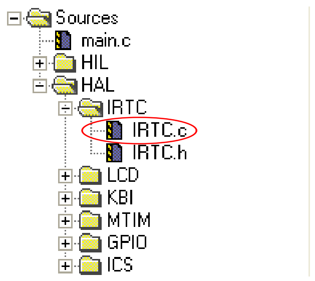

Figure A-2. IRTC.c file

vfnIRTC_Init

Description—Independent Real Time Clock (IRTC) Initialization.

ANSIC prototype:
    void vfnRTC_Init (void);

Input parameters:
    None

Return value:
    None

vfnUnprotectRTC

Description—This function disables the IRTC registers write protection.

ANSIC prototype:
void vfnUnprotectRTC (void);

Return value:
None

u16IRTC_Time_GetYear

Description—Unprotect the IRTC and it returns the Year status from the RTC_YEARMON register.

ANSIC prototype:
UINT16 u16IRTC_Time_GetYear (void);

Input parameters:
None

Return value:
UINT16 Year—Time status in years

u8IRTC_Time_GetMonth

Description—Unprotect the IRTC and it returns the Time Month status from the RTC_YEARMON register

ANSIC prototype:
UINT8 u8IRTC_Time_GetMonth (void);

Input parameters:

None

Return value:

UINT8—Current month

u8IRTC_Time_GetDay

Description—Unprotect the IRTC and it returns the Day status from the RTC_DAYS register

ANSIC prototype:

UINT8 u8IRTC_Time_GetDay (void);

Input parameters:

None

Return value:

UINT8—Current Day

u8IRTC_Time_GetHour

Description—Unprotect the IRTC and it returns the Hour status from the RTC_HOURMIN register

ANSIC prototype:
UINT8 u8IRTC_Time_GetHour (void);

Input parameters:

None

Return value:

UINT8—Current Hour

u8IRTC_Time_GetMinute

Description—Unprotect the IRTC and it returns the Minute status from the RTC_HOURMIN register

ANSIC prototype:
UINT8 u8IRTC_Time_GetMinute (void);

Input parameters:
None

Return value:
UINT8—Current minute

u8IRTC_GetCountDown

Description—Unprotect the IRTC and it returns the Countdown status from the RTC_COUNT_DN register

ANSIC prototype:
UINT8 u8IRTC_GetCountDown (void);

Input parameters:
None

Return value:
UINT8—Current countdown status

vfnIRTC_Time_SetYear

Description—This function disables the IRTC security protection and then converts the given value into offset years by using the SetYear macro and finally stores the result in the RTC_YEARMON register.

Input parameters:
UINT16 Year—The value that needs to be related to the base year

Return value:
None

vfnIRTC_Time_SetMonth

Description—This function disables the IRTC security protection and then stores the input parameter in the RTC_YEARMON register.

Input parameters:
UINT8 Month—Month counter value in binary mode.
1 = January 12 = December

Return value:
None

vfnIRTC_Time_SetDay

Description—This function disables the IRTC security protection and then stores the input parameter in the RTC_DAYS register.

Input parameters:
UINT8 Day – Day counter value in binary mode Valid count 1 – 31

Return value:
None

vfnIRTC_Time_SetHour

Description—This function disables the IRTC security protection and then stores the input parameter in the RTC_HOURMIN register.

Input parameters:
UINT8 Hour—Hour counter value in binary mode
Valid count 0–23; 0–11 is AM and 12–23 is PM.

Return value:
None

vfnIRTC_Time_SetMinute

Description—This function disables the IRTC security protection and then stores the input parameter in the RTC_HOURMIN register.

Input parameters:
UINT8 Minute—Minute counter value in binary mode
Valid count 0–59

Return value:
None

vfnIRTC_Time_SetSeconds

Description—This function disables the IRTC security protection and then stores the input parameter in the RTC_SECONDS register.

Input parameters:
UINT8 Second—Second counter value in binary mode
Valid count 0—59

Return value:
None

A.3 LCD.c

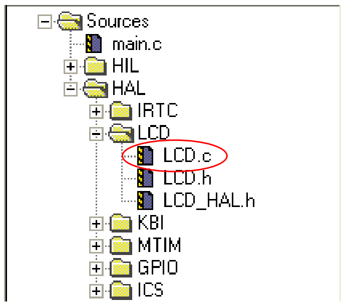

Figure A-3. LCD.c

vfnLCD_Write_Msg

Function name:
vfnLCD_Write_Msg

Description—Writes a message to the LCD alphanumeric space. If the message is longer than the maximum number of alphanumeric characters, it truncates the message. If the message is shorter than the maximum number of alphanumeric characters it writes the message and the remaining alphanumeric characters are written with spaces.

ANSIC prototype:

void vfnLCD_Write_Msg (UINT8 _POINTER lbpMessage);

Input parameters:
UINT8 *pu8Message—The first character on the array to write.

Return Value:
None

vfnLCD_WriteDecimal

Function name:
vfnLCD_WriteDecimal

Description—Converts a hexadecimal value to a decimal, then the decimal value is converted to ASCII. After the number is converted to ASCII the function starts to write the conversion result on the display starting at u8Place.

ANSIC prototype:
void vfnLCD_WriteDecimal (UINT32 u32DecimaValue, UINT8 u8Place)

Input parameters:
UINT32 u32DecimaValue—Value that needs to be converted
UINT8 u8Place—Alphanumeric place to start writing

Return value:
None

The LCD driver used for this project can found in the application note titled *LCD Driver Specifications* (document AN3796). This document details how to migrate the software driver provided to a specific custom LCD.

A.4 KBI.c

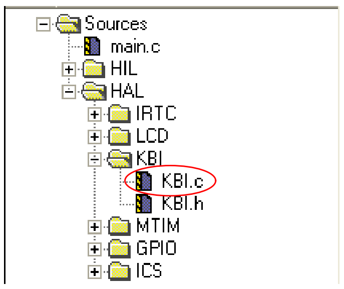

Figure A-4. KBI.c file

vfnKBI_Init

Function Name:
vfnKBI_Init

Description—Initializes the Keyboard Interrupt module with the macros previously defined in the KBI.h file

ANSIC prototype:
void vfnKBI_Init (void);

Input parameters:
None

Return Value:
None

KBI_ISR

Function Name:
  KBI_ISR

Description—KBI interrupt service routine. This function clears the interrupt flag, stores it in a global variable as KBI_PORT status and finally sets a flag to tell the main application that a KBI event has occurred.

ANSIC prototype:
  interrupt VectorNumber_Vkbi1 void KBI_ISR (void);

Input parameters:
  None

Return Value:
  None

A.5 MTIM.c

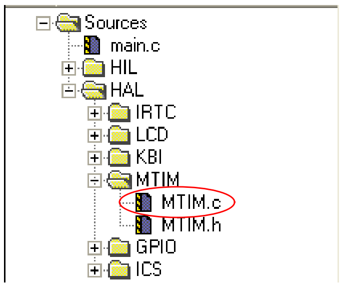

Figure A-5. MTIM.c file

vfnMTIM_Delay

Function Name:
  vfnMTIM_Delay

Description—Initializes the MTIM module with the u16Delay parameter and pulls the timer overflow flag. This function ends after the delay has passed.

ANSIC prototype:
  void vfnMTIM_Delay (UINT16 u16Delay);

Input parameters:
  UINT16 u16Delay—Number of counts needed for a certain delay.
  Delay = (256 * u16Delay) / 20 MHz

Return Value:
  None

A.6 Keys.c

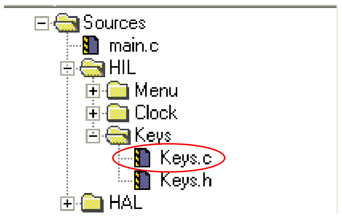

Figure A-6. Keys.c

u8fnGetKey

Function name:
u8fnGetKey

Description—After receiving a KBI event this function is called by the main state machine after debouncing the push button the status of the keyboard interrupt is then returned.

ANSIC prototype:
UINT8 u8fnGetKey (void);

Input parameters:
None

Return Value:
UINT8

0—False key
1—Enter event
2—Click event

A.7 Menu.c

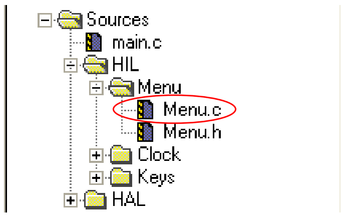

Figure A-7. Menu.c

vfnClockMenu

Function Name:
vfnClockMenu

Description—After receiving a keyboard event this function determines how to respond with the key previously detected. If no Enter event has been detected, the function displays the main menu messages. After a main menu has been selected, the function runs a secondary state machine that manages the menu.

ANSIC prototype:
void vfnClockMenu (UINT8 u8KeyPress);

Input parameters:

UINT8 u8KeyPress—Keyboard pressed

1—Enter event

2—Click

Return Value:

None

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: AN4023
Rev. 0, 7/2010