XC800 Family

AP08109

Implementing a Capacitive Touch-Controlled Alarm Clock Application with XC836

Application Note

V1.0, 2010-06

Microcontrollers

Edition 2010-06

Published by
Infineon Technologies AG
81726 Munich, Germany

LEGAL DISCLAIMER

THE INFORMATION GIVEN IN THIS APPLICATION NOTE IS GIVEN AS A HINT FOR THE IMPLEMENTATION OF THE INFINEON TECHNOLOGIES COMPONENT ONLY AND SHALL NOT BE REGARDED AS ANY DESCRIPTION OR WARRANTY OF A CERTAIN FUNCTIONALITY, CONDITION OR QUALITY OF THE INFINEON TECHNOLOGIES COMPONENT. THE RECIPIENT OF THIS APPLICATION NOTE MUST VERIFY ANY FUNCTION DESCRIBED HEREIN IN THE REAL APPLICATION. INFINEON TECHNOLOGIES HEREBY DISCLAIMS ANY AND ALL WARRANTIES AND LIABILITIES OF ANY KIND (INCLUDING WITHOUT LIMITATION WARRANTIES OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS OF ANY THIRD PARTY) WITH RESPECT TO ANY AND ALL INFORMATION GIVEN IN THIS APPLICATION NOTE.

Information

For further information on technology, delivery terms and conditions and prices, please contact the nearest Infineon Technologies Office (**www.infineon.com**).

Warnings

Due to technical requirements, components may contain dangerous substances. For information on the types in question, please contact the nearest Infineon Technologies Office.

Infineon Technologies components may be used in life-support devices or systems only with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

XC83x
Revision History: V1.0 2010-06

Previous Version(s):

Page	Subjects (major changes since last revision)
–	

We Listen to Your Comments
Is there any information in this document that you feel is wrong, unclear or missing? Your feedback will help us to continuously improve the quality of this document. Please send your proposal (including a reference to this document) to:
mcdocu.comments@infineon.com

Table of Contents

1 Introduction

The Real-Time Clock (RTC) and the LED Touch Sense Control Unit (LEDTSCU) are two of the new modules introduced in the Infineon XC800 Family of 8-bit microcontrollers, and which are available in the XC82x and XC83x devices.

This application note highlights features of these modules and their ease of use in a capacitive touch-controlled Alarm Clock application example, implemented on the XC836 Easy Kit board. The Easy Kit board schematics and the application source code are included together with this application note.

For a detailed description of the XC836, please refer to the XC83x User's Manual.

1.1 Overview

Topics covered in this application note include:

- Features of the Alarm Clock
- Modules involved
- Implementation details of the Alarm Clock
- Program flow chart/state diagram

2 Features of the Alarm Clock

The Alarm Clock features in this application example are:

- Display Current Time via 7-segment LED Displays
  - display hours and minutes
  - display seconds counting only
- Display Date
- Display Alarm Time
- User set-up via Touch Pads
- Buzzer activated at alarm time
- Seconds-Indication LEDs
  - to blink every second
- Mode-Indication LEDs
  - to indicate to the user which mode the clock is in

2.1 Hardware Features

Figure 1 below provides an illustration on the components of the XC836 Easy Kit board that are used to implement the Alarm Clock. **Figure 2** shows the Mode-Indication LEDs assignment.

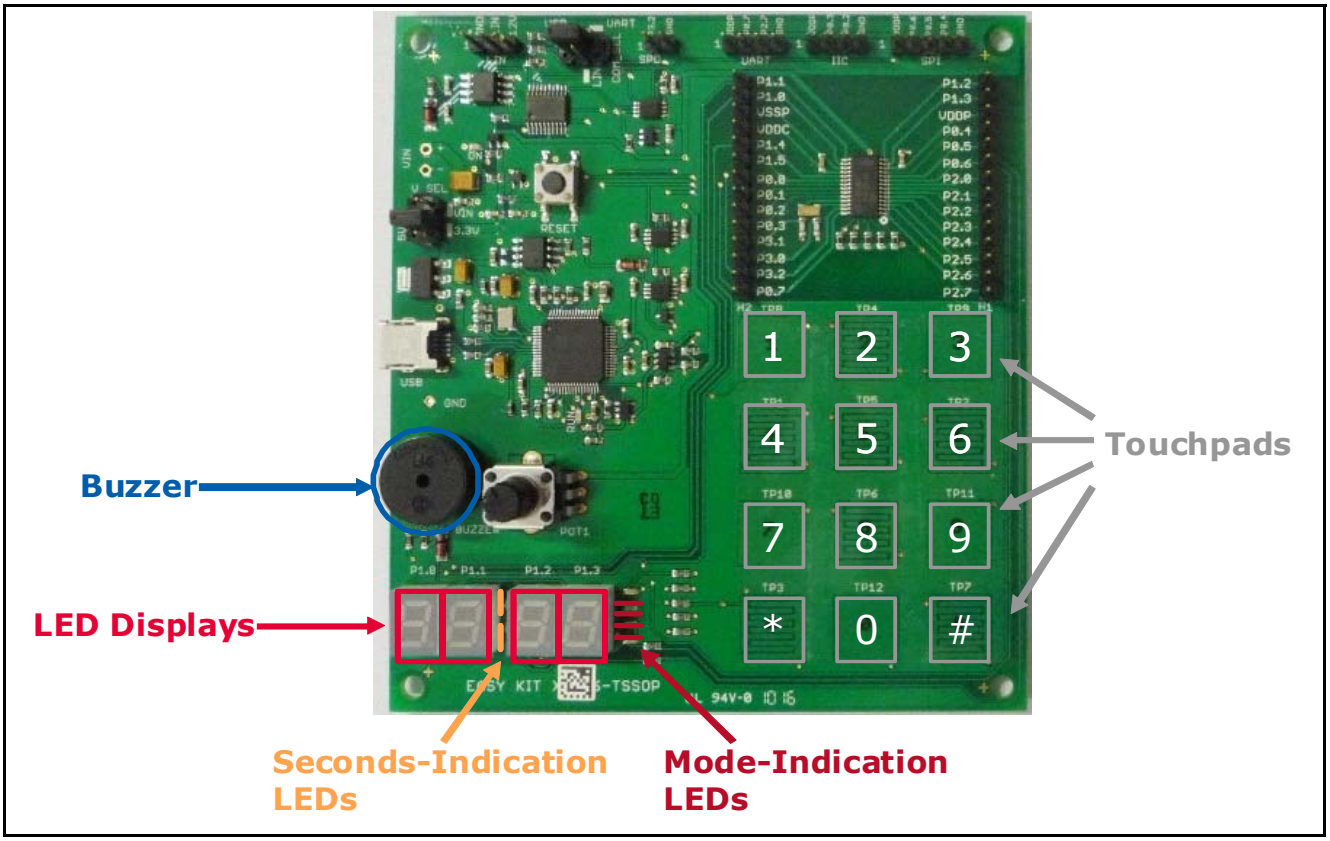

Figure 1 Implementation of an Alarm Clock application with XC836 Easy Kit board

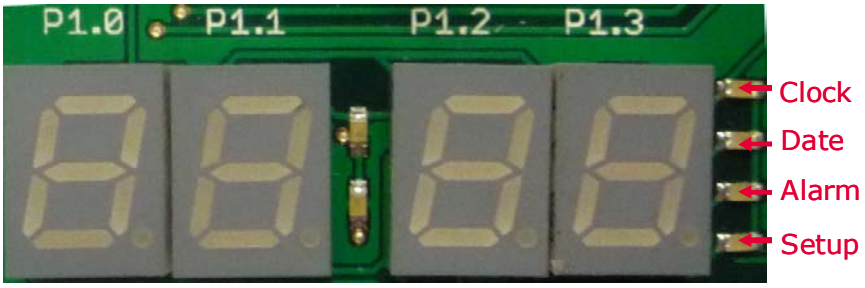

Figure 2 Mode-Indication LEDs

2.2 Software Features

At power-on, the display shows the clock time in hours and minutes, with the default values of 00:00. A list of default values used in the application code can be found in **Table 1**.

The user now has a choice of three operations:

1. Switch the display to count Seconds by pressing and holding down either the '*', '0' or '#' touch pad keys.

Note: It is only possible to display the Seconds count while in the Display Clock Mode. To alternate between the modes, press and hold either '', '0' or '#' touch pad keys. Note that when in Seconds-counting mode, the feature to set up the clock is disabled.*

2. Show the Date or Alarm Time by pressing the '#' touch pad once for Date or twice for Alarm Time.
3. Setup the Clock by pressing the touch pad marked '*'.

Note: Depending on the current mode, 'C' (Clock mode), 'd' (Date mode) or 'A' (Alarm mode) will flash on the right hand 7-segment LED display for 2 seconds at each change of the Display mode.

In Setup Mode, the 7-segment LED Displays will go blank, and a blinking cursor will appear on the leftmost 7-segment LED display.

A flow chart view of the features mentioned above can be seen in **Figure 3**.

Table 1 Alarm clock variables' default values

Variable	Default Value
Clock Time	00h:00min:00s (12.00am)
Date	01:01 (1st Jan)
Alarm Time	00h:01min:00s (12.01am)

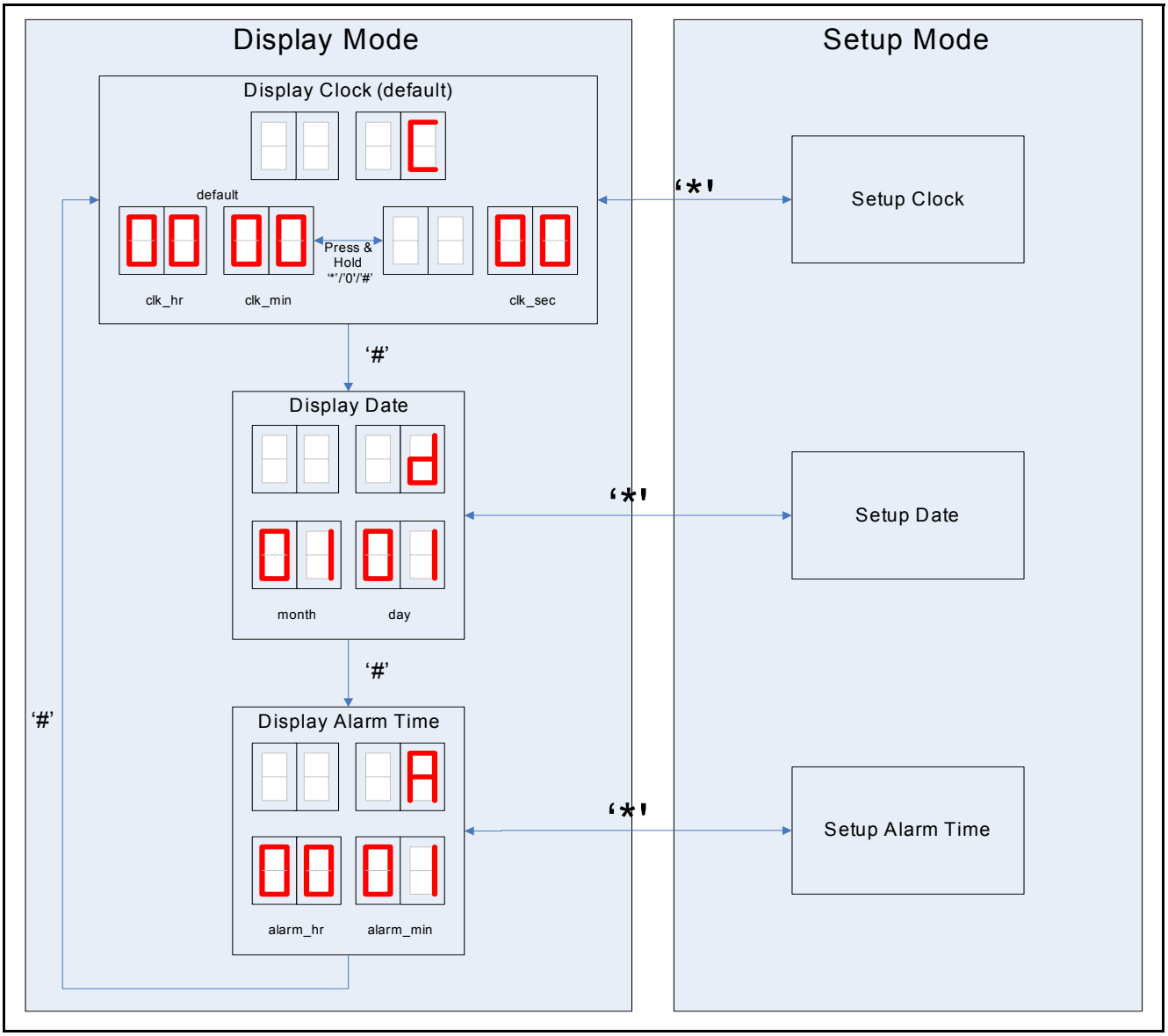

Figure 3 Flow chart of Alarm Clock features

3 XC83x Modules Configuration

Only a few of the XC83x device's modules are required for the Alarm Clock application example. These modules are:

- Real-Time Clock (RTC)
- LED and Touch Sense Control Unit (LEDTSCU)
- LEDTS ROM Library
- Timer 2 (T2)

Configurations for each module are presented in the sections which follow.

Note: Configuration is setup via the free DAvE tool from Infineon. DAvE will generate skeleton code based on the configuration entered. All the screenshots that follow are taken from DAvE.

3.1 Real-Time Clock (RTC)

"Mode 0: Time keeping mode with 32.768KHz crystal clock" is selected for the RTC. Interrupt at every second (ESRTC) is enabled with the RTC to start clocking after initialization (**Figure 4**).

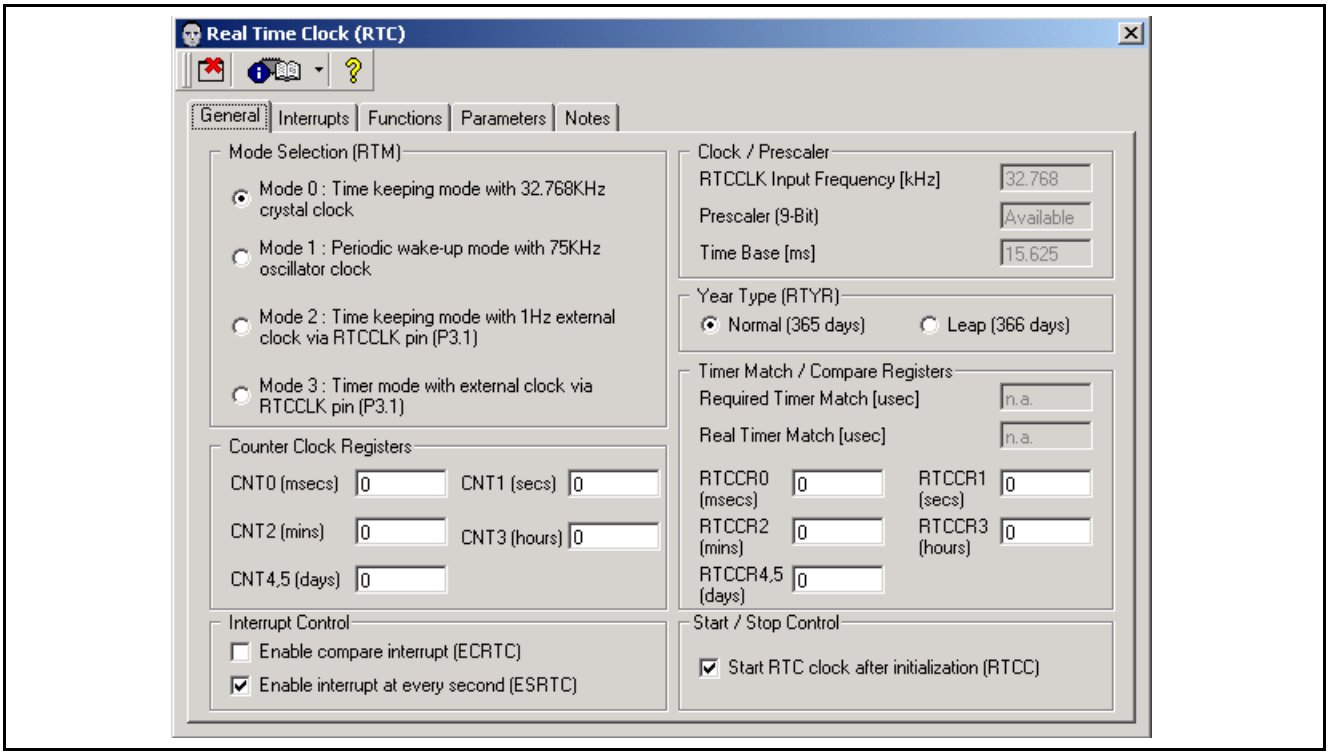

Figure 4 RTC DAvE configurations

3.2 LED and Touch Sense Control Unit (LEDTSCU) and LEDTS ROM Library

For the LEDTSCU module pin control:

- P0.0-P0.4 are selected as touch sense pins
- P0.0-P0.6 are selected for LED line pins
- P1.0-P1.4 are selected as LED column pins (**Figure 5**)

The LEDTS ROM Library is enabled (**Figure 6**). The brightness of the LEDs can be adjusted by setting the COMPARE parameter under the LED box. For the Touch Sense, select the oscillation window checkbox and set the trip point to 'fixed' for all 5 touch pads. Error detection is also enabled to have a long touch/press feature for the application.

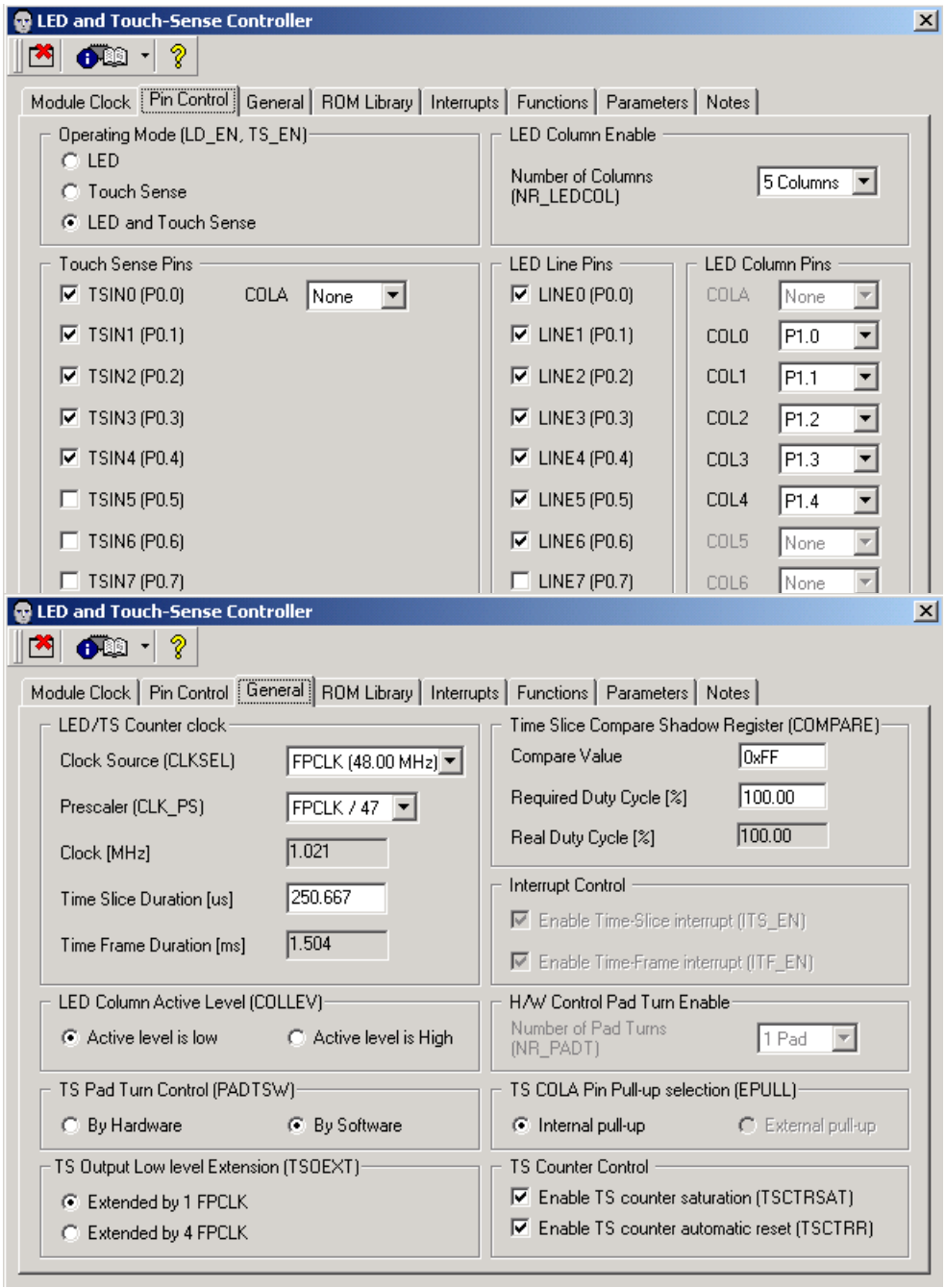

Figure 5 LEDTSCU module DAvE configurations

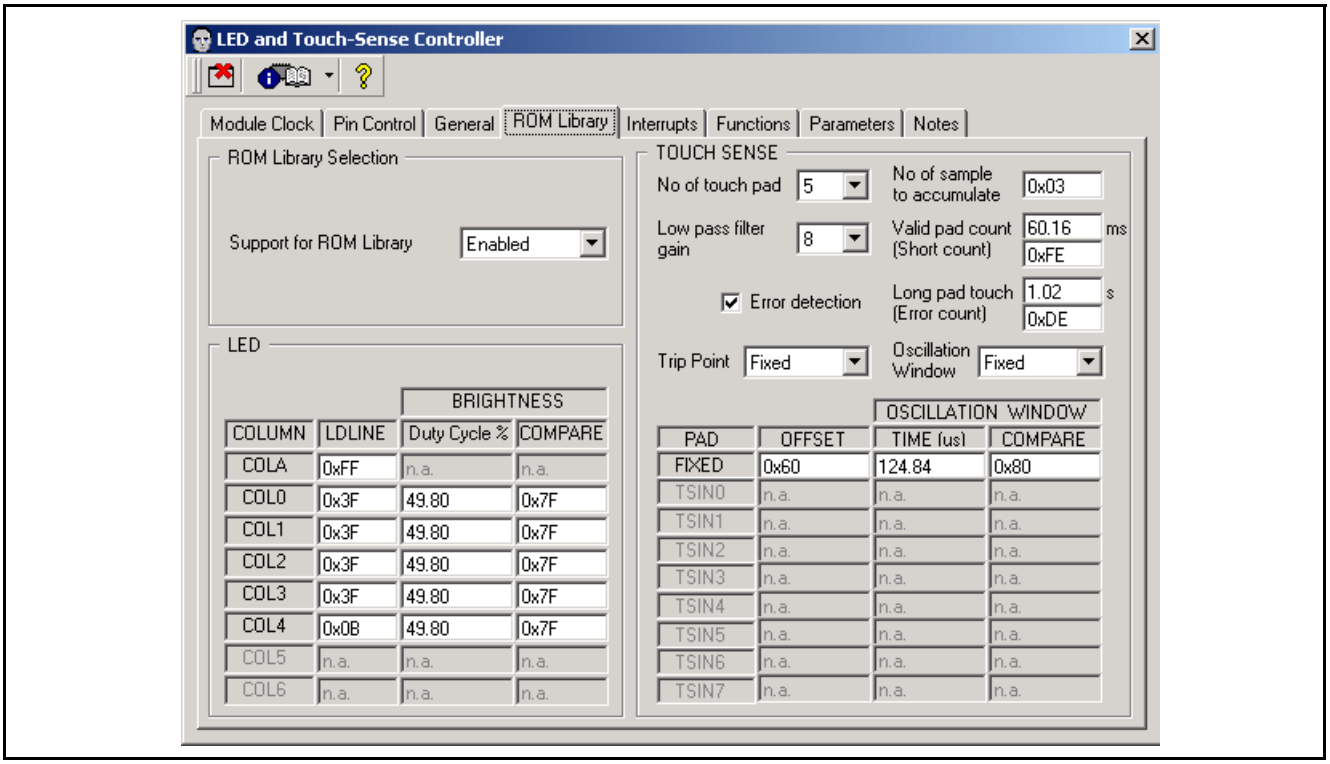

Figure 6 LEDTS ROM Library DAvE configurations

3.3 Timer 2 (T2)

The 16-bit timer function is selected with automatic reloading and interrupt enabled when an overflow of 250ms occurs (**Figure 7**).

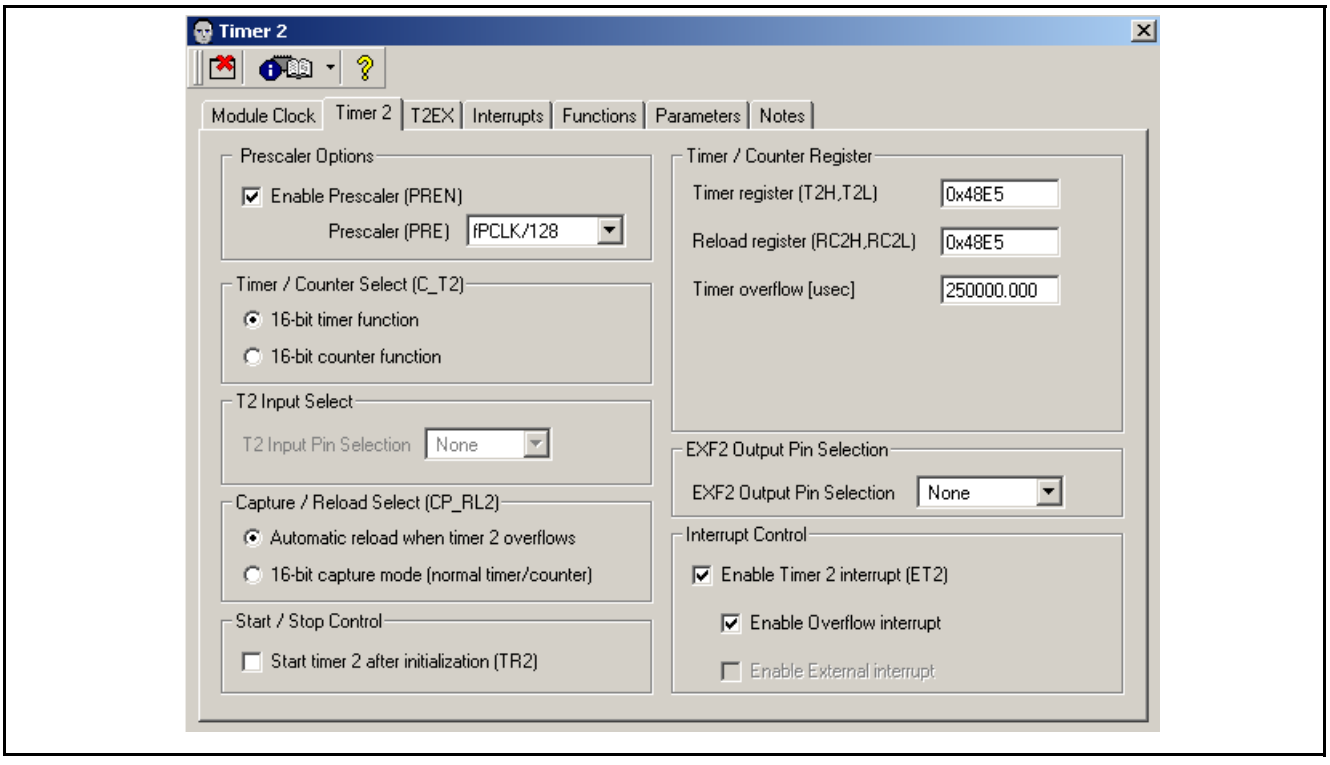

Figure 7 T2 DAvE configurations

4 Implementation of Alarm Clock

This chapter describes how the XC836 modules, using the configurations given in **Chapter 3**, can be used to implement the features of the alarm clock outlined in **Chapter 2**.

For ease of understanding, the implementation details are given for each module, along with an explanation of how each module is used to implement the different Alarm Clock features.

4.1 RTC

This section describes the Alarm Clock features implemented by utilizing the RTC.

4.1.1 Time-Keeping

The RTC registers CNT0-CNT5 hold the actual real time value in the range of milliseconds, seconds, minutes, hours and days respectively (in hexadecimal). To read the value of these registers, a capture event can be triggered by setting the RTCCT bit in RTCON register.

Two CPU cycles after the trigger, the RTC compare/capture registers, RTCCRx, will be overwritten with the captured CNT values. The capture event will result in the capture of all 6 CNT register values. Only the values of CNT1-CNT3 are required for time-keeping task. The values of CNT4-CNT5 are used for date-tracking (see **Chapter 4.1.2**).

The function `void RTC_vReadClkCounter(stRTC_REGx *pstCaptValue)` can be enabled and generated via DAvE. This function reads the content of the RTC_CNTx registers via the capture mechanism and the captured values will then be saved in an assigned structure variable, for example, *current_time*. **Table 2** gives an overview of the handling of the captured values.

Table 2 Structure variable for holding captured RTC_CNT values

Structure Variable Element	Parameter
current_time.ubREG0	milliseconds
current_time.ubREG1	seconds
current_time.ubREG2	minutes
current_time.ubREG3	hours
current_time.uwREG4_5	days

4.1.2 Date-Tracking

CNT4 and CNT5 form a 9-bit register to hold the value for the number of days. As mentioned in **Chapter 4.1.1**, the hexadecimal value for the number of days can be extracted via the capture event.

Note: CNT4 and CNT5 count the number of days from 0 up to 364 (normal year) or 365 (leap year) instead of from 1 to 365 (normal year) or 366 (leap year), before resetting.

Two forms of conversion are required to handle the date-tracking for the Alarm Clock application:

1. Conversion from hexadecimal day to the respective decimal day and month. These 2 values are needed for display on the 7-segment LEDs. This conversion is carried out in the function **void MAIN_vConvert_Date_FromHex(uword hex_date)**. **Figure 8** illustrates the flow of this conversion.
2. Conversion from decimal day and month to hexadecimal day. This is required after the user sets up the date, where the conversion result is written over the value of registers CNT4 and CNT5 so that the RTC can keep track of the date. This conversion is carried out in the function **uword MAIN_uwConvert_Date_ToHex(ubyte date_array[])**. **Figure 9** illustrates the flow of this conversion.

Both of these conversions involve a look-up table which provides the default number of days in each calendar month.

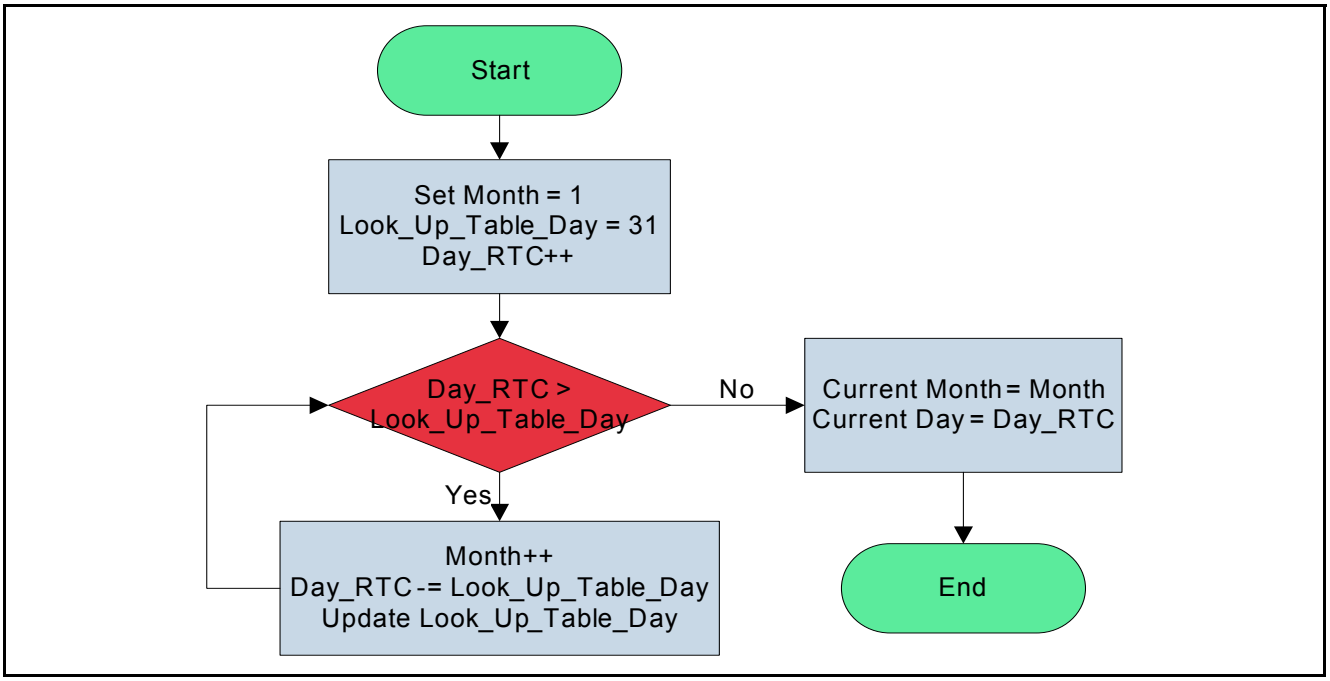

Figure 8 Flowchart for converting hexadecimal day value to decimal day and month

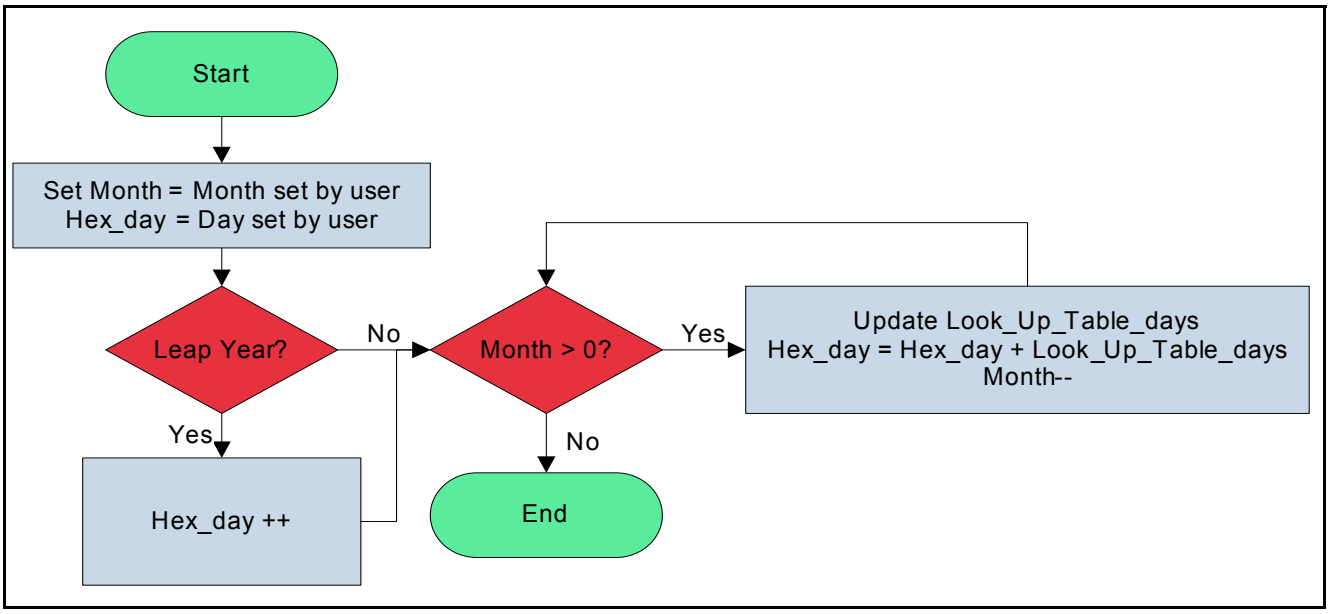

Figure 9 Flowchart for converting decimal day and month to hexadecimal day value

4.1.3 Saving User Time/Date

This section covers the topic of re-writing the RTC_CNTx values with the user-defined values. For the topic of handling the user input, please refer to **Chapter 4.2.2**.

The registers RTC_CNTx cannot be re-written while the RTC is still counting, so RTC operation must first be stopped. In addition, the RTC_CNTx registers are bit-protected, so the user will also have to open access to the register before re-writing with a new value.

The example code to perform this task is in the Appendix under the heading **/*Saving User Time/Date*/**.

4.2 LEDTSCU & LEDTS ROM Library

This section looks at the use of the LEDTSCU and LEDTS ROM Library to implement some of the Alarm Clock application features.

It is important to note that the LEDTSCU module tasks of driving the LEDs and touch-pad sensing are controlled in a time-multiplexed manner.

Table 3 lists down the time-multiplexed column enabling sequence which is specific to this Alarm Clock application while **Figure 10** provides a temporal view of the column and component activation.

Table 3 LEDTSCU Time-Multiplexed Column Enabling Sequence

Column	Column Pin	Column No. (for ROM Library input)	Corresponding active HW function
COL4	P1.4	5	Buzzer, Seconds- & Mode-Indication LEDs
COL3	P1.3	4	Rightmost 7-segment LED
COL2	P1.2	3	2nd 7-segment LED from the right
COL1	P1.1	2	2nd 7-segment LED from the left
COL0	P1.0	1	Leftmost 7-segment LED
COLA	-	0	Touchpads

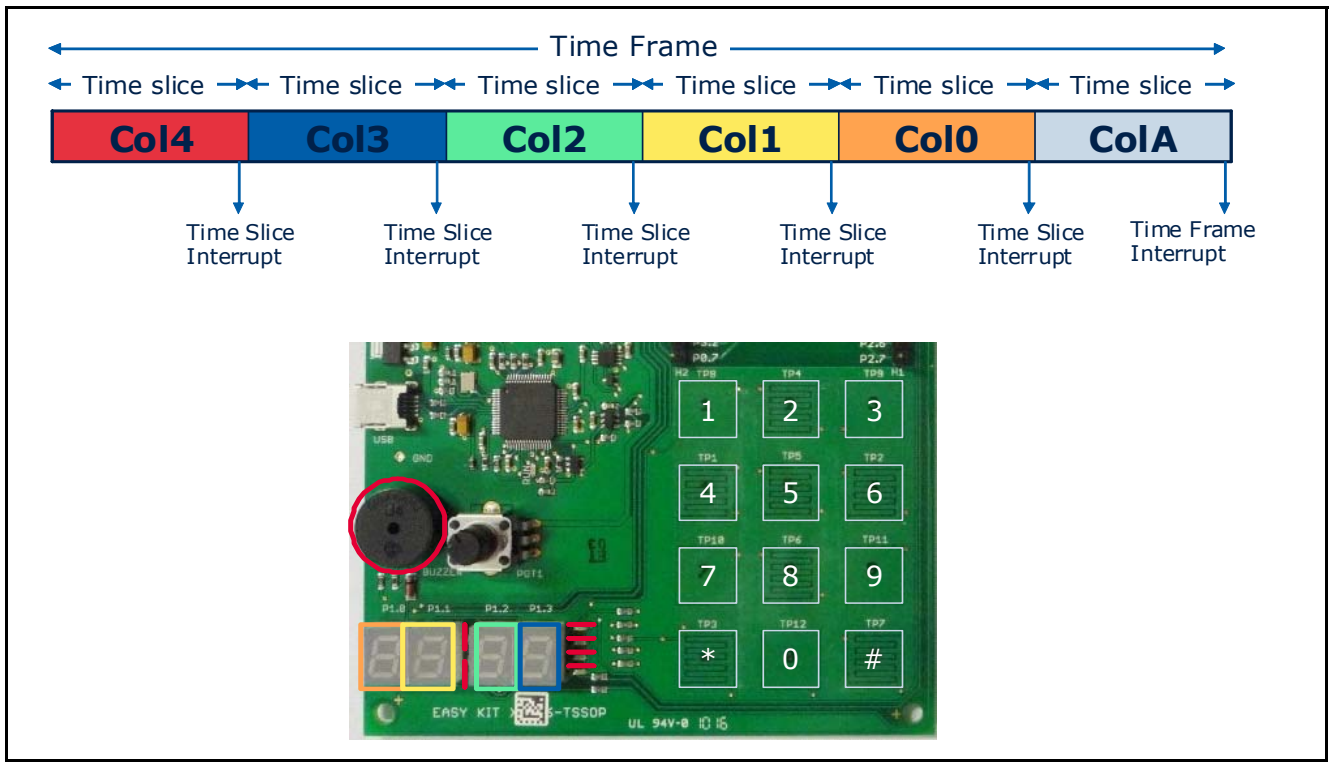

Figure 10 Activation Sequence of XC83x Easy Kit board components

4.2.1 Display

To drive the 7-segment LEDs and the small LEDs (which are used as the Seconds-indication and Mode-indication LEDs), the function ***SET_LDLINE_CMP*** in the LEDTS ROM Library is utilised. In brief, this function keeps track of the currently enabled LED column, and performs a shadow transfer to program the LTS_LDLINE and LTS_COMPARE register values corresponding to the LED column activated. This function is automatically called in the time slice interrupt service routine, when the code is generated from DAvE.

To drive any of the 7-segment LEDs, the user will have to set the LDLINE_VALUE[x], where x is the column number (for ROM Library) as listed in **Table 3**. As an example, to display '1' on the second 7-segment LED from the left, LDLINE_VALUE[2] = 06_H. This is based on the PCB connection of the XC836 Easy Kit board.

The brightness of the LEDs can also be adjusted, by changing the value of TS_COMPARE[x], where x is the column number (for ROM Library) as listed in **Table 3**. The value for maximum brightness level is FF_H whereas setting TS_COMPARE[x] = 00_H will effectively switch the LED off.

Seconds-Indication LEDs

The seconds-indication LEDs are wired to P1.4 (LED column) and P0.1 and P0.3 (LED line). These two LEDs can be switched on by setting bits 1 and 3 of LDLINE_VALUE[5]. To perform the blinking effect, the **RTC's Interrupt at Every Second** feature is used. In this interrupt service routine, both bits 1 and 3 of LDLINE_VALUE[5] are toggled.

Mode-Indication LEDs

The mode-indication LEDs are wired to P1.4 (LED column) and P0.0, P0.2, P0.4 and P0.6 (LED line). These four LEDs can be switched on by setting bits 0, 2, 4 and 6 of LDLINE_VALUE[5].

4.2.2 Touch Sensing

The column corresponding to the touch sensing pads is activated in the last time slice of a time frame (**Figure 10**). To capture the touch pad that is being pressed, the LEDTS ROM Library *FINDTOUCHEDPAD()* function can be called in the time frame interrupt.

The function *FINDTOUCHEDPAD()* has three output parameters; PADFLAG, PADRESULT and PADERROR. From the values of these three parameters, it is possible to determine whether a pad has a 'normal' touch, a 'long' touch or whether no pads are being touched at all. **Figure 11** illustrates the flow of operations in the time frame interrupt service routine to implement some of the features in our alarm clock.

Determining the pad that was pressed

The value of PadResult is determined from the way in which the touch pads are connected, and therefore indicates the pad that was pressed. **Figure 12** shows the touch pad connections for the XC836 Easy Kit board, while **Table 4** lists the respective PADRESULT values for each touch pad. As an example, the PADRESULT value of 08_H would indicate that TP11(pad labelled '9') was pressed.

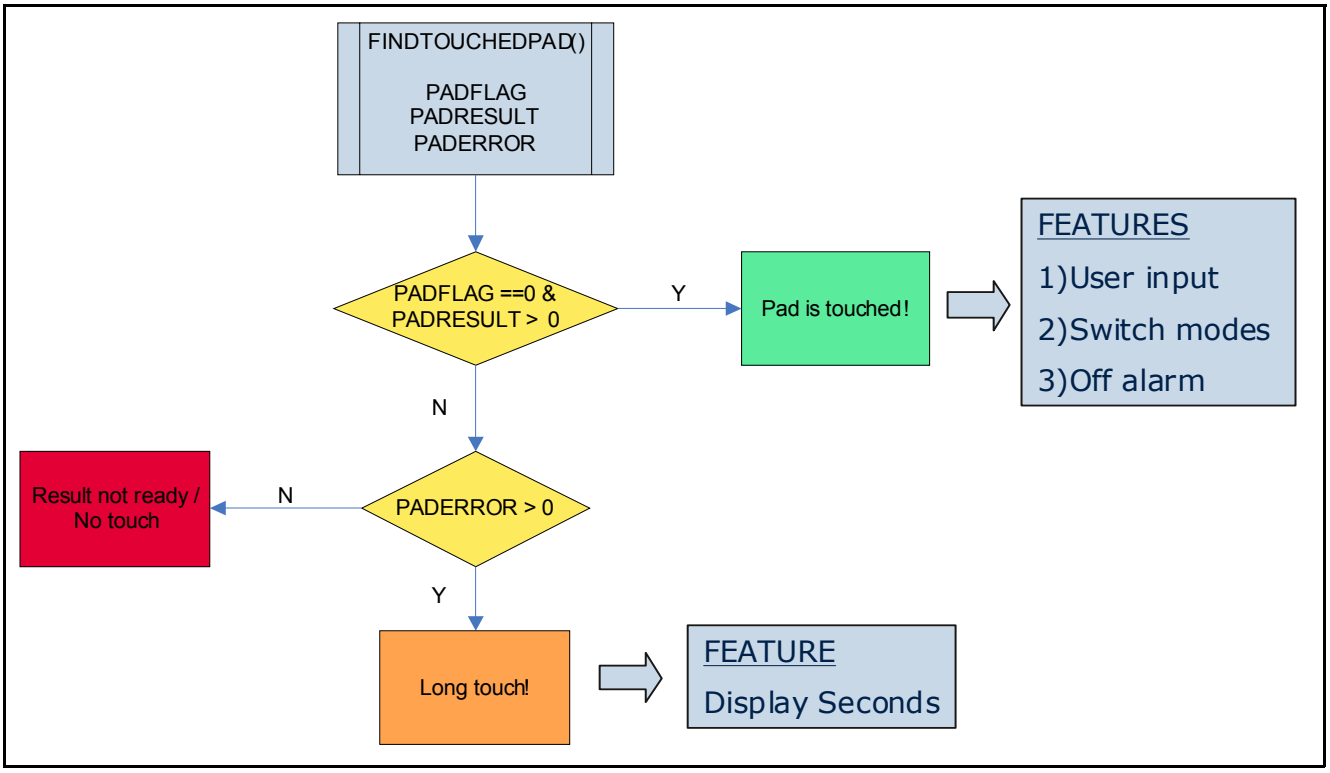

Figure 11 Flowchart of operations in Time Frame Interrupt

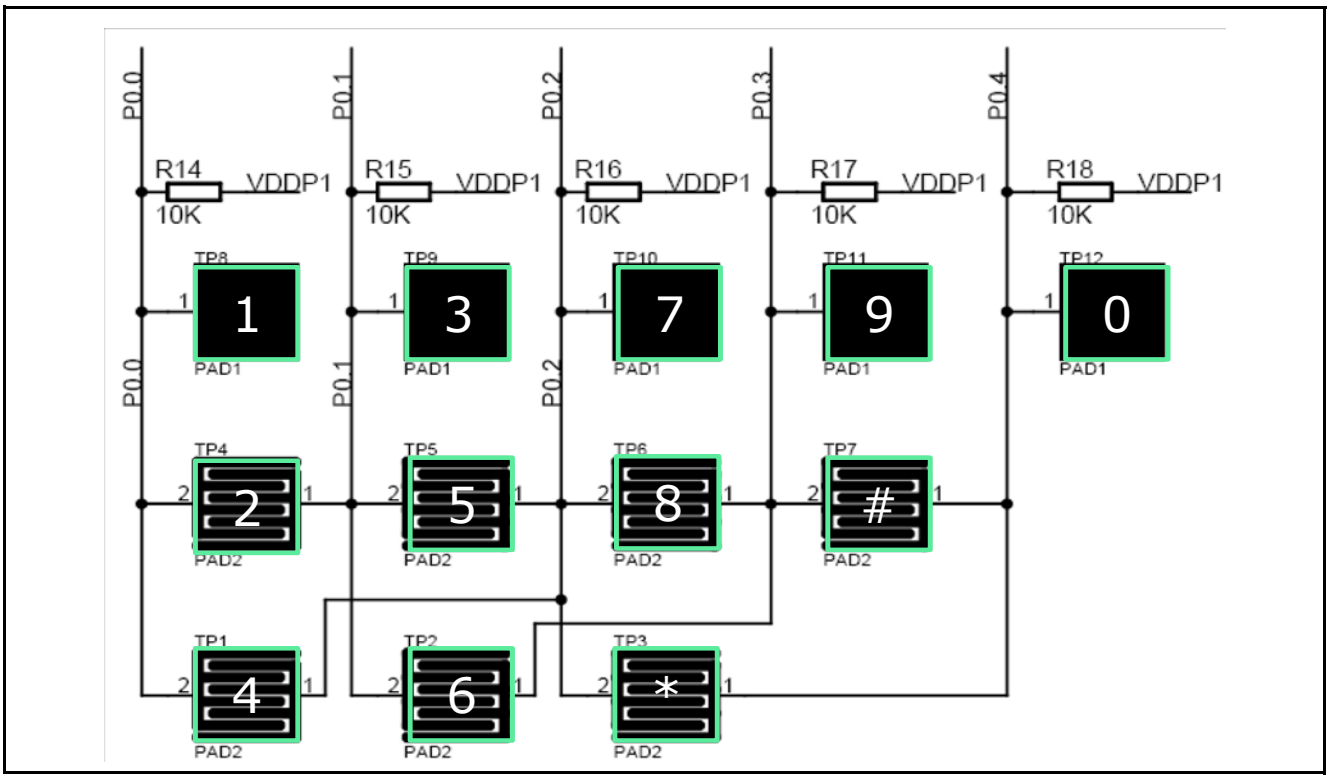

Figure 12 Touch pads on XC83x Easy Kit board schematic diagram

Table 4 Interpretation of PadResult values

	PadResult							
	PadTurn7	**PadTurn6**	**PadTurn5**	**PadTurn4**	**PadTurn3**	**PadTurn2**	**PadTurn1**	**PadTurn0**
	-	**-**	**-**	**P0.4**	**P0.3**	**P0.2**	**P0.1**	**P0.0**
TP8 (1)	0	0	0	0	0	0	0	1
TP4 (2)	0	0	0	0	0	0	1	1
TP9 (3)	0	0	0	0	0	0	1	0
TP1 (4)	0	0	0	0	0	1	0	1
TP5 (5)	0	0	0	0	0	1	1	0
TP2 (6)	0	0	0	0	1	0	1	0
TP10 (7)	0	0	0	0	0	1	0	0
TP6 (8)	0	0	0	0	1	1	0	0
TP11 (9)	0	0	0	0	1	0	0	0
TP3 (*)	0	0	0	1	0	1	0	0
TP12 (0)	0	0	0	1	0	0	0	0
TP7 (#)	0	0	0	1	1	0	0	0

4.2.3 Alarm Buzzer

The buzzer is connected between the LED column pin P1.4, and an LED line pin P0.5. Because of the time-multiplex operation of the LEDTSCU module, the buzzer will only be activated in the first time slice of a time frame (**Table 3**, **Figure 10**) together with the seconds- and mode-indication LEDs. Therefore, the buzzer can be triggered by setting bit 5 of LDLINE_VALUE[5].

4.3 T2

In this Alarm Clock application, the purpose of the T2 module is to periodically toggle the signal to the buzzer when the alarm is triggered, to provide a ‘beeping’ effect rather than a monotone. Timer 0 or Timer 1 could also be used for this task, but T2 is preferred because of its auto-reloading feature.

As stated in **Table 3.3**, the T2 is configured to overflow every 250ms, at which point an overflow interrupt is triggered. The overflow interrupt service routine handles the toggle of the buzzer signal.

5 Main Program Structure

The main program can be described as a system of finite states. **Figure 13** provides an illustration of this state machine, and how transition can happen between the states. In the code (Main.c), the variable *operation* is used to represent the state.

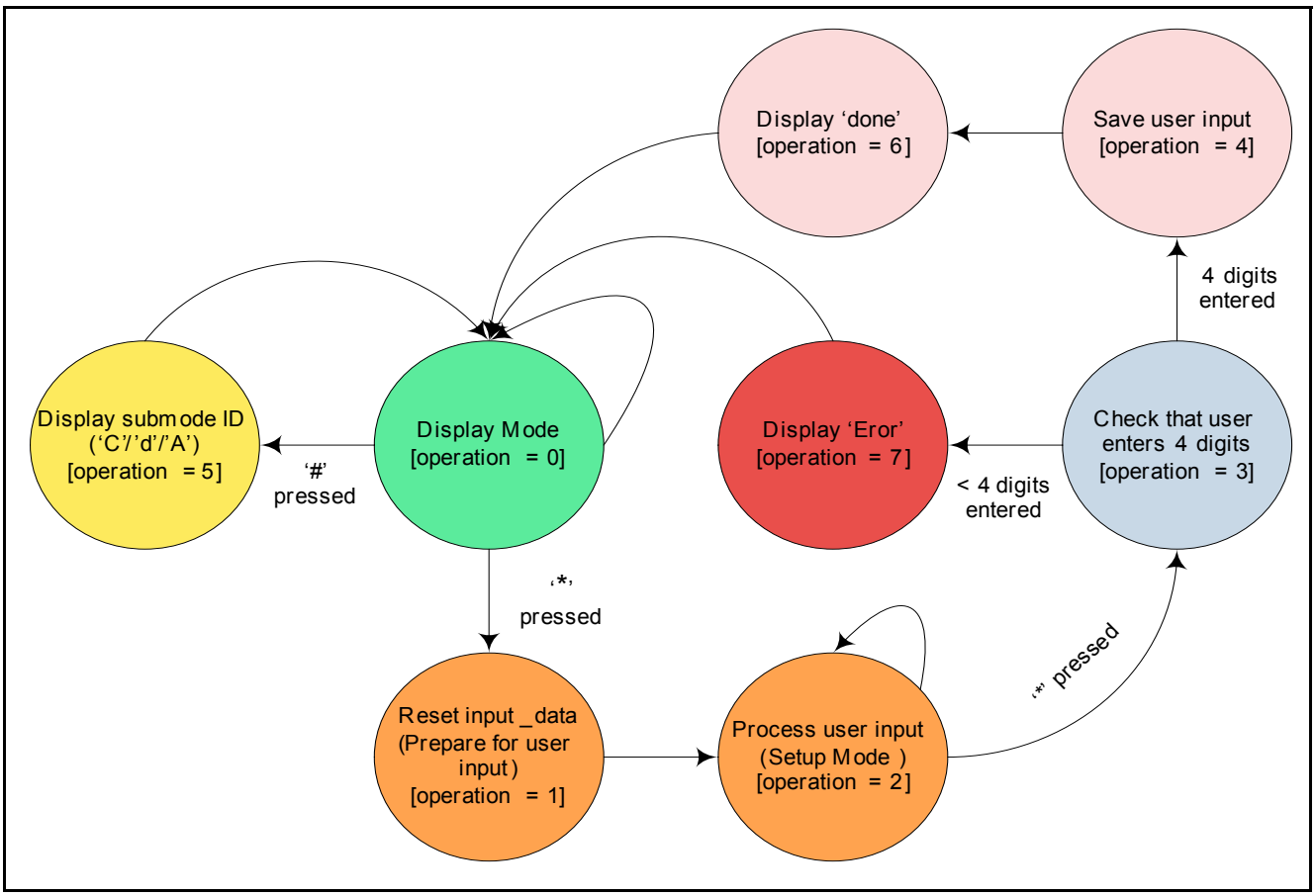

Figure 13 Main program state diagram

Actions/Tasks performed in each state:

Operation = 0 [default]:

1) Trigger RTC capture event to read current time.
2) Trigger buzzer if alarm time has been reached
3) Check for submode status (Clock/Date/Alarm), and display values accordingly.

Operation = 1:

1) Reset variable input_data in preparation to receive and hold user input.
2) Clear all 7-segment LEDs.
3) Switch to Operation = 2.

Operation = 2:

1) Blink cursor on leftmost 7-segment LED.
2) If a valid key is pressed, the number is stored in input_data and displayed on 7-segment LED.
3) Shift cursor.

Operation = 3:

1) If user had entered 4 digits, switch to Operation = 4.

2) Else, switch to operation = 0 (user input not saved).

Operation = 4:

1) Save user input accordingly (Re-writing the values of RTC_CNT0-5 for Clock & Date setting, Overwriting of value of variable *alarm_data* for Alarm time setting).

2) Switch to Operation = 6.

Operation = 5:

1) Display ‘C’/’d’/’A’ according to current submode status.

2) Check if 2 seconds have lapsed (via RTC Interrupt at every second).

3) Switch to Operation = 0.

Operation = 6:

1) Display ‘done’.

2) Check if 2 seconds have lapsed (via RTC Interrupt at every second).

3) Switch to Operation = 0.

Operation = 7:

1) Display ‘Eror’.

2) Check if 2 seconds have lapsed (via RTC Interrupt at every second).

3) Switch to Operation = 0.

Appendix - Example Code

The following are code fragments taken from the original code. These fragments do not function on their own and are included only for reference. The actual source code for the application can be found in the attachments supplied with this document.

void MAIN_vConvert_Date_FromHex(uword hex_date)

```
{
    uword day_cntr = 0;       //keep track of the day
    ubyte month_cntr = 0x01;   //keep track of the month
    ubyte check_days = 0x00;

    day_cntr = hex_date + 1;   //sets up day_cntr
    check_days = Days_LookUpTable[month_cntr]; //fetch no. of days in January
    while (day_cntr > check_days)
    {
       month_cntr++;
       day_cntr -= check_days; //calculate difference btwn day_cntr & check_days
       if ((month_cntr == 2) && (RTC_RTCON1 == 0x01))
       {
          check_days = 29;     //leap year February
       }
       else
       {
          check_days = Days_LookUpTable[month_cntr];
                          //update check_days with next month's no. of days
       }
    }
    display_data[2] = month_cntr/10;//update date display value holder
    display_data[3] = month_cntr%10;//update date display value holder
    display_data[0] = day_cntr/10;//update date display value holder
    display_data[1] = day_cntr%10;//update date display value holder
    return;
}
```

uword MAIN_uwConvert_Date_ToHex(ubyte date_array[])

```
{
    ubyte month = 0x00;
    uword total_days = 0x00;

    month = ((date_array[2]*10) + date_array[3]);//calculate month
    total_days = ((date_array[0]*10) + date_array[1]);//calculate day
    if ((RTC_RTCON1 == 0x01) && (month > 2))
    {
       total_days++; //to factor in leap year
    }
    while (month > 0)
    {
       total_days += Days_LookUpTable[month - 1];
                    //add no. of days in months leading up to the current month
       month--;
```

```
    }
    return (total_days - 1); //return result
}
```

/*Saving User Time/Date*/

```
RTC_RTCON &= ~(ubyte)0x01;    // real time clock operation stopped
SFR_PAGE(_su1, noSST);        // switch to page 1 without saving
MAIN_vUnlockProtecReg();      // open access to protected reg
RTC_CNT0 = 0x00;             // reset milliseconds reg
MAIN_vUnlockProtecReg();      // open access to protected reg
RTC_CNT1 = 0x00;             // reset seconds reg
MAIN_vUnlockProtecReg();      // open access to protected reg
RTC_CNT2 = new_min;           // load count clock register 2
MAIN_vUnlockProtecReg();      // open access to protected reg
RTC_CNT3 = new_hr;            // load count clock register 3
MAIN_vUnlockProtecReg();      // open access to protected register
RTC_CNT4 = new_date1;         // change RTC_CNT4 value
MAIN_vUnlockProtecReg();      // open access to protected register
RTC_CNT5 = new_date2;         // change RTC_CNT5 value
MAIN_vlockProtecReg();        // close access to protected register
SFR_PAGE(_su0, noSST);        // switch to page 0 without saving
RTC_RTCON |= 0x01;           // real time clock operation resumed
```

w w w . i n f i n e o n . c o m

Published by Infineon Technologies AG