# Chapter 7

# Timers

Correct system timing is a fundamental requirement for the proper operation of a real-time application. The timing definition can dictate how the data information processed during the execution of the application program. The clock implementations vary between devices in the MSP430 family. Each device provides different clock sources, controls and uses. This chapter discusses the clock controls included in the platforms used.

The MSP430 4xx family has two general-purpose 16-bit or 8-bit counters and event timers, named Timer\_A, Timer\_B, and a Basic Timer. The Basic Timer module is only implemented in '4xx devices. The 2xx device family also has Timer\_A and Timer\_B, but the clock signals are provided by the basic clock module+.

The timers may receive an internal or external clock. Timer\_A and Timer\_B also include multiple independent capture and compare blocks, with interrupt capabilities.

The capture and compare blocks are suited for applications such as timed events and Pulse Width Modulator (PWM) respectively.

7.1 Ti	mers Introduction7-3
7.2 Ba	sic Timer 17-4
7.2.1	Registers7-5
7.3 Ti	mer_A and Timer_B7-6
7.3.1	Operating modes characteristics7-7
7.3.2	Timer reset7-8
7.3.3	Timer_A registers7-8
7.4 Ca	pture/Compare blocks7-9
7.4.1	Capture mode7-9
7.4.2	Compare mode7-10

# Topic

7.4.3 Capture/Compare blocks registers7-12
7.5 Timer_A Interrupts7-13
7.6 Timer_B special features7-13
7.6.1 Timer_B registers7-14
7.7 Laboratory 3: Timers use7-16
7.7.1 Lab3A_1: Memory clock with Basic Timer17-16
7.7.2 Lab3A_2: Real Time Clock with Basic Timer17-20
7.7.3 Lab3B: Memory Clock with Timer_A7-23
7.7.4 Lab3C: Buzzer tone generator7-26
7.7.5 Lab3D: Frequency measurement7-32
7.8 Quiz
7.9 FAQs

I

# 7.1 Timers Introduction

In *Chapter 5 – Device Systems and Operating Modes* described the different system clocks sources that allow the CPU and peripherals to operate, depending on the device in the MSP430 family. This chapter will focus mainly on the configuration and operation of timers and their different uses.

This chapter describes the basic clock module+ (BCS+) three byteaddressable registers, all of which are fully software controllable. These allow the clock sources to be configured by one or two oscillators (depending on the device) or by external crystals or resonators, as wells as by the internal digitally controlled oscillator (DCO). The DCO allows a working frequency up to 16 MHz, lower power consumption and lower internal oscillator start up time.

The system timing is fundamental to almost any embedded application. The timers are used in applications to:

- □ Generate fixed-period time events;
- □ Allow a periodic wakeup from sleep mode;
- □ Count transitional signal edges;

□ Replace delay loops with timer calls to allow the CPU to sleep, consuming much less power.

The MSP430 devices contain several timer modules, each one suitable for different types of tasks. Chapter 5 described the timer modules as Basic Clock Module+ implemented in the MSP430x2xx family and the Watchdog Timer present in all families of MSP430 devices. This chapter will focus on:

□ Basic Timer 1 module (implemented only in the MSP430x4xx family);

- □ Timer\_A;
- □ Timer\_B.

The clock signals of the MSP430x4xx family devices are controlled by two sets of registers. The first set of registers configures the lowfrequency signals for use by peripheral modules. These registers, namely Basic Timer Control Register (BTCTL), Basic Timer Counter 1 (BTCNT1) and Basic Timer Counter 1(BTCNT2) control the Basic Timer 1 module.

The second set of registers is dedicated to the configuration of the general-purpose clocks system. They comprise of the System Clock Control (SCFQCTL), System Clock Frequency Integrator 0 (SCFI0), System Clock Frequency Integrator 1 (SCFI1), and the two Frequency Locked Loop control registers, FLL+CTL0 and FLL+CTL1.

# 7.2 Basic Timer 1

The Basic Timer 1 module is formed by two independent 8-bit timers: Basic Timer 1 Counter 1 (BTCNT1) and Basic Timer 1 Counter 2 (BTCNT2), which can be used in cascade to form a 16-bit timer (software selectable by BTCTL register configuration).

The main characteristics of this module are:

- Clock for LCD module;
- Suitable for a RTC implementation;
- Basic interval timer;
- □ Simple interrupt capability.

These timers/counters have different applications:

□ BTCNT1: Used to generate the frame frequency for the LCD controller:

- Read/write 8-bit register;
- Clock source: ACLK;
- Frame frequency selectable by software (BTFRFQx bits on the BTCTL register) based on the ACLK division (x);
- $f_{LCD} = ACLK/x$ .

□ BTCNT2: Used as a programmable frequency divider with interrupt capability, to provide periodic CPU interrupts and/or a Real Time Clock system.

- Read/write 8-bit register;
- Three possible clock sources: ACLK or SMCLK, or ACLK/256 when cascaded with BTCNT1 (selected by the BTSSEL and BTDIV bits on the BTCTL register);
- Sources the Basic Timer1 interrupt BTIFG, with an interval selected by the BTIPx bits in the BTCTL register;

Figure 7-1. Basic Timer 1 block diagram.



# 7.2.1 Registers

# BTCTL, Basic Timer 1 Control Register

BTSSEL BTHOLD BTDIV BTFRFO1 BTFRFO0 BTIP2 BTIP1 BTIF	7	6	5	4	3	2	1	0
	BTSSEL	BTHOLD	BTDIV	BTFRFQ1	BTFRFQ0	BTIP2	BTIP1	BTIP0

Bit		Description		
7	BTSSEL	BTCNT2 clock select (together wit	h the B	TDIV bit)
6	BTHOLD	Basic Timer 1 hold:		
		$BTHOLD = 0 \qquad \Rightarrow \qquad$	<b>BTCN</b>	F1 and BTCNT2 active
		$BTHOLD = 1 \qquad \Rightarrow \qquad$	<b>BTCN</b>	Γ1 hold, if BTDIV = 1
		$BTHOLD = 1 \qquad \Rightarrow \qquad$	BTCN	F1 and BTCNT2 hold
5	BTDIV	Basic Timer 1 clock divider (togetl BTCNT2 clock signal):	ner with	the BTSSEL bit selects the
		BTSSEL BTDIV = $0.0$	$\Rightarrow$	ACLK
		BTSSEL BTDIV = $0.1$	$\Rightarrow$	ACLK/256
		BTSSEL BTDIV = $10$	$\Rightarrow$	SMCLK
		BTSSEL BTDIV = $1 \ 1$	$\Rightarrow$	ACLK/256
4-3	BTFRFQx	LCD frame frequency:		
		BTFRFQ1 BTFRFQ0 = $0 0$	$\Rightarrow$	f <sub>ACLK</sub> /32
		BTFRFQ1 BTFRFQ0 = 0 1	$\Rightarrow$	f <sub>ACLK</sub> /64
		BTFRFQ1 BTFRFQ0 = $1 0$	$\Rightarrow$	f <sub>ACLK</sub> /128
		BTFRFQ1 BTFRFQ0 = $1 1$	$\Rightarrow$	f <sub>ACLK</sub> /256
2-0	BTIPX	Basic Timer 1 interrupt interval:		
		BTIP2 BTIP1 BTIP0 = $0 0 0$	$\Rightarrow$	f <sub>CLK2</sub> / 2
		BTIP2 BTIP1 BTIP0 = $0 \ 0 \ 1$	$\Rightarrow$	f <sub>CLK2</sub> / 4
		BTIP2 BTIP1 BTIP0 = $0 1 0$	$\Rightarrow$	f <sub>CLK2</sub> / 8
		BTIP2 BTIP1 BTIP0 = $0 1 1$	$\Rightarrow$	f <sub>CLK2</sub> / 16
		BTIP2 BTIP1 BTIP0 = $100$	$\Rightarrow$	f <sub>CLK2</sub> / 32
		BTIP2 BTIP1 BTIP0 = $1 \ 0 \ 1$	$\Rightarrow$	f <sub>CLK2</sub> / 64
		BTIP2 BTIP1 BTIP0 = $110$	$) \Rightarrow$	f <sub>CLK2</sub> / 128
		BTIP2 BTIP1 BTIP0 = $1 \ 1 \ 1$	. ⇒	f <sub>CLK2</sub> / 256

# IE2, Interrupt Enable Register 2

7	0
BTIE	

Bit		Description
7	BTIE	Basic Timer 1 interrupt enable when BTIE = 1

# IFG2, Interrupt Flag Register 2

7	0
BTIFG	

Bit		Description
7	BTIFG	Basic Timer 1 interrupt flag BTIFG = 1 when interrupt pending

# 7.3 Timer\_A and Timer\_B

Timer\_A and Timer\_B are two general-purpose 16-bit counters and event timers. There are slight differences between the two timers that will be explained in this section.

Both timers feature:

- Asynchronous 16-bit timer/counter with four operating modes:
  - Timer\_A length: 16 bits;
  - Timer\_B length: programmable to be 8, 10, 12, or 16 bits.
  - The timer/counter register, TAR (Timer\_A) or TBR (Timer\_B) -from now on described as TxR- increments or decrements (depending on mode of operation) with each rising edge of the clock signal;
  - The timer can generate an interrupt when it overflows;
  - Wide interrupt interval range: 1/MCLK to 32 seconds.
- □ Selectable and configurable clock source:
  - ACLK, SMCLK, or externally via TxCLK or INCLK (selected with the TASSELx bits);
  - The selected clock source may additionally be divided by 2, 4, or 8 (IDx bits configuration).
- □ Configurable capture/compare registers:
  - Timer\_A has three or five capture/compare registers;
  - Timer\_B has three or seven capture/compare registers;
  - Timer\_B capture/compare registers can be grouped.

□ Configurable outputs and several internal connections to other modules, allowing a faster response because no cycles are wasted while the ISR loads/executes and avoids CPU wakeup, which in turn saves power:

- Outputs with Pulse Width Modulation (PWM) capability;
- Comparator\_A;
- Direct Memory Access;
- Digital-to-Analogue Converter (DAC12);

□ Asynchronous input and output latching:

- Timer\_A capture/compare registers are not buffered, being updated immediately when written to;
- Timer\_B capture/compare registers are double-buffered with synchronized loading.

□ Interrupt vector register for fast decoding of all Timer\_A and Timer\_B interrupts.

- TACCR0 (or TBCCR0) interrupt vector for TACCR0 (or TBCCCR0) CCIFG;
- TAIV (or TBIV) interrupt vector for the remainder CCIFG flags and TAIFG (or TBIFG).

The timer has four modes of operation as described in *Table 7–1*, selected with the MCx bits in the Timer\_A or Timer\_B Control Registers.

Table 7-1. Timer\_A and Timer\_B operating modes.

MCx	Mode	Description
00	Stop	The timer is halted
01	Up	Up counting repeatedly mode (from 0x0000 to the value in the TACCR0 or TBCCR0 register)
10	Continuous	Continuous counting repeatedly mode (from 0x0000 to 0xFFFF)
11	Up/down	Up/down counting repeatedly mode (from 0x0000 to the value in the TACCR0 or TBCCR0 register and back down to zero)

Because the main characteristics provided by the two timers are similar, the following sections will only discuss the features of Timer\_A. To use these features with Timer\_B, it is only necessary to configure the appropriate registers. The register descriptions for Timer\_A are valid for Timer\_B.

# 7.3.1 Operating modes characteristics

#### Up mode

The main characteristics of up mode are:

□ TAR counts up with each clock pulse until it reaches the value in the TACCR0 (or TBCCR0) register;

□ The TACCR0 (or TBCCR0) interrupt flag, CCIFG, is set when the timer counts to the TACCR0 (or TBCCR0) value;

 $\Box$  When it reaches this value, EQU0 = 1 (restarts TAR counting from zero);

□ The TAIFG (or TBIFG) interrupt flag is set when the timer counts from TACCR0 (or TBCCR0) value to zero.

- Interrupt period:  $t_{INT} = 1/[f_{CLK}/Prescaler/(TxCCR0+1)];$ 
  - $t_{INT}$ : TxIFG interrupt period [sec];
  - f<sub>CLK</sub>: Clock source frequency [Hz];
  - Prescaler: Divider (IDx bits).

#### Continuous mode

The main characteristics of continuous mode are:

□ TxR counts up with each clock pulse till 0xFFFF (65536 counts);

 $\hfill\square$  When it reaches this value, at the next clock pulse it will restart the TxR counting from zero;

 $\hfill\square$  The TxIFG (or TxIFG) interrupt flag is set when the timer counts from 0xFFFF to zero.

■ Interrupt period:  $t_{INT} = 1/[fCLK/Prescaler/65536];$ 

□ (Correct only for TAR; for TBR 4 there are different end values. See User's Guide for additional details).

#### Up/down mode

The main characteristics of the up/down mode are:

□ TxR counts up with each clock pulse until it reaches the value in the TxCCR0 register;

□ The TxCCR0 interrupt flag, CCIFG, is set when the timer counts from TxCCR0 – 1 to TxCCR0;

□ When it reaches this value, the counting is inverted, starting at the next clock pulse to decrement till zero;

 $\Box$  The interrupt flag TxIFG is set when the timer completes counting down from 0x0001 to 0x0000.

■ Interrupt period:  $t_{INT} = 1/[fCLK/Prescaler/(TxCCR0×2];$ 

# 7.3.2 Timer reset

The timer can be reset using the following operations:

□ Writing 0 at the TAR (or TBR) register;

□ Writing 0 at the TACCR0 (or TBCCR0) register provided that the timer is not in continuous mode;

□ Setting the TACLR (or TBCLR) bit in the Timer Control Register (TACTL or TBCTL).

# 7.3.3 Timer\_A registers

The description of the following registers relates to Timer\_A registers. Timer\_B registers have the same features, but the Control Register has different bit names and provides additional features. For more details, refer to MSP430x4xx User's Guide <slau056g.pdf> Chapter 16 – Timer\_B to find the differences. The special features of Timer\_B are described in section 7.6.

#### TACTL, Timer\_A Control Register

15					10	9	8
		Unu	sed		-	TASSEL1	TASSEL0
7	6	5	4	3	2	1	0
ID1	ID0	MC1	MC0	Unused	TACLR	TAIE	TAIFG

Bit		Description
9-8	TASSELx	Timer_A clock source:TACLKTASSEL1 TASSEL0 = 00 $\Rightarrow$ TACLKTASSEL1 TASSEL0 = 01 $\Rightarrow$ ACLKTASSEL1 TASSEL0 = 10 $\Rightarrow$ SMCLKTASSEL1 TASSEL0 = 11 $\Rightarrow$ INCLK
7-6	IDx	Clock signal divider: $ID1 ID0 = 00 \Rightarrow / 1$ $ID1 ID0 = 01 \Rightarrow / 2$ $ID1 ID0 = 10 \Rightarrow / 4$ $ID1 ID0 = 11 \Rightarrow / 8$
5-4	MCx	Clock timer operating mode:Stop modeMC1 MC0 = 00 $\Rightarrow$ Stop modeMC1 MC0 = 01 $\Rightarrow$ Up modeMC1 MC0 = 10 $\Rightarrow$ Continuous modeMC1 MC0 = 11 $\Rightarrow$ Up/down mode
2	TACLR	Timer_A clear when TACLR = 1
1	TAIE	Timer_A interrupt enable when TAIE = 1
0	TAIFG	Timer A interrupt pending when TAIFG = 1

# 7.4 Capture/Compare blocks

Both TIMER\_A (and TIMER\_B) contain independent capture and compare blocks, TACCRX (or TBCCRX) that may be used to capture the timer register contents, as they are at time of an event, or to generate an event when the timer register contents correspond to capture/compare register contents, e.g. to generate time intervals.

The mode is selected by the mode bit CAP in their individual Capture/Compare Control register, TACCTLx (or TBCCTLx).

# 7.4.1 Capture mode

The capture mode is used to measure the period of timed events, with minimal CPU intervention.

Capture mode configuration is achieved by carrying out the following steps:

□ Set CAP bit to select the capture mode;

□ Set SCS bit to synchronize the capture with the next timer clock (recommended to avoid race conditions);

□ The input signal is sampled by the CCIxA (or CCIxB) input, selected by the CCISx bits in the Capture/Compare Control Register, TACCTLx (or TBCCTLx);

□ The capture edge of the input signal (rising, falling, or both) is selected by the CMx bits;

□ When the appropriate edge is detected on the selected input line, the value in the Timer register is latched into the TACCRx (or TBCCRx) register, providing a time mark for the event;

The interrupt flag CCIFG is set;

 $\Box$  The bit COV (=1) controls an overflow event when a second capture is performed before the value from the first capture is read.

#### 7.4.2 Compare mode

The compare mode is used for pulse generation or interrupts at specific time intervals. One of its common applications is to generate Pulse Width Modulation (PWM) output signals.

Compare mode operation is configured as follows:

- Reset CAP bit to select compare mode;
- □ TxR counts to the value programmed in the TxCCRx register;

 $\hfill \Box$  When the timer value is equal to the value in the TxCCRx register, an interrupt is generated:

- Interrupt flag CCIFG is set;
- Internal signal EQUx = 1 (x is the number of the CCR channel).

□ EQUx affects the output compare signal OUTx according to the output mode (defined by the OUTMODx bits in the TxCCTLx register – see *Table 7-2*).

□ The input signal CCI is latched into SCCI.

The output unit is used to generate output signals (such as PWM signals) based on the EQU0 and EQUx signals. There are eight different operating modes numbered from 0 to 7. The OUTx signal changes with the rising edge of the timer clock for all modes, except mode 0.

The output operating modes are described in Table 7-2 and the examples of the output signals for different timer modes are presented in *Figure 7-2*.

Output operating modes uses:

- □ Modes 2, 3, 6 and 7: generation of PWM output signals:
  - Mode 3: active PWM signal at low state;
  - Mode 7: active PWM signal at high state;

- Modes 2 and 6: complementary PWM signals separated by a deadband when both signals are reset (application example: H-bridges for bi-directional motor drive).
- □ Modes 1 and 5: single event generation;
- □ Mode 4: signal with half the frequency of the timer signal.

Table 7-2. Output operating modes.

	OUTMODx	Mode	Description
	000	Output	The output signal OUTx is defined by the bit OUTx
	001	Set	$OUTx = 1 \implies timer = TACCRx$
			$OUTx = 0 \Rightarrow timer = 0$ or until another output mode is
			selected and affects the output
	010	Toggle/Reset	$OUTx = toggle \implies timer = TACCRx$
			$OUTx = 0 \Rightarrow timer = TACCR0$
	011	Set/Reset	$OUTx = 1 \implies timer = TACCRx$
			$OUTx = 0 \Rightarrow timer = TACCR0$
	100	Toggle	$OUTx = toggle \implies timer = TACCRx$
			The output period is double the timer period
	101	Reset	$OUTx = 0 \Rightarrow timer = TACCRx$
			$OUTx = 1 \Rightarrow another output mode is selected and affects$
			the output
	110	Toggle/Set	$OUTx = toggle \Rightarrow timer = TACCRx$
			$OUTx = 1 \Rightarrow timer = TACCR0$
	111	Reset/Set	$OUTx = 0 \Rightarrow timer = TACCRx$
			$OUTx = 1 \Rightarrow timer = TACCR0$



# 7.4.3 Capture/Compare blocks registers

The description of the following registers relates to Timer\_A registers. Timer\_B registers have the same features, but the Capture/Compare Control Register have different bit names and provide additional features. Refer to MSP430x4xx User's Guide <slau056g.pdf> Chapter 16 – Timer\_B to find the differences. The special features of Timer\_B are described in section 7.6.

# TACCTLx, Timer\_A Capture/Compare Control Register

15	14	13	12	11	10		9	8
CM1	CM0	CCIS1	CCIS0	SCS	SCCI	Un	used	CAP
7	6	5	4	3		2	1	0
OUTMOD2	OUTMOD1	OUTMOD0	CCIE	CCI		TUC	COV	CCIFG

Bit		Description
15-14	CMx	Capture mode:
		$CM1 CM0 = 00 \qquad \Rightarrow \qquad No  capture$
		CM1 CM0 = 01 $\Rightarrow$ Capture on rising edge
		CM1 CM0 = 10 $\Rightarrow$ Capture on falling edge
		CM1 CM0 = 11 $\Rightarrow$ Capture on both edges
13-12	CCISx	Capture/compare input select:
		$CCIS1 \ CCIS0 = 00  \Rightarrow \qquad CCIxA$
		$CCIS1 CCIS0 = 01 \implies CCIxB$
		$CCIS1 \ CCIS0 = 10  \Rightarrow \qquad GND$
		$CCIS1 CCIS0 = 11 \implies V_{cc}$
11	SCS	Synchronize capture input signal with timer clock:
		SCS = 0 $\Rightarrow$ Asynchronous capture
		SCS = 1 $\Rightarrow$ Synchronous capture
10	SCCI	Synchronized capture/compare input
8	CAP	Mode:
		Capture mode $\Rightarrow$ CAP = 1
		Compare mode $\Rightarrow$ CAP = 0
7-5	OUTMODX	Output mode:
		OUTMOD2 OUTMOD1 OUTMOD0 = $000 \Rightarrow$ Bit OUT
		$OUIMOD2 OUIMOD1 OUIMOD0 = 001 \implies Set$
		$OUIMOD2 OUIMOD1 OUIMOD0 = 010 \implies Ioggle/Reset$
		$OUIMOD2 OUIMOD1 OUIMOD0 = 011 \implies Set / Reset$
		$OUTMOD2 OUTMOD1 OUTMOD0 = 100 \implies Toggle$
		$OUIMOD2 OUIMOD1 OUIMOD0 = 101 \implies Reset$
		$OUIMOD2 OUIMOD1 OUIMOD0 = 110 \implies Ioggle / Set$
	0.015	$OUIMOD2 OUIMOD1 OUIMOD0 = 111 \implies \text{Reset / Set}$
4	CCIE	Capture/compare interrupt enable when CCIE = 1.
3		Capture/compare input
2	001	Output state
Ţ	COV	Capture overflow when $COV = 1$
0	CCIFG	Capture/compare interrupt flag $CCIFG = 1$ when interrupt pending

# 7.5 Timer\_A Interrupts

Interrupt characteristics:

- □ Capture mode:
  - Any CCIFG flag is set when a timer value is captured in the associated TACCRx register.
- □ Compare mode:
  - Any CCIFG flag is set if TAR counts to the associated TACCRx value.
- □ Software may also set or clear any CCIFG flag;

□ All CCIFG flags request an interrupt when their corresponding CCIE bit and the GIE bit are set.

There are two interrupt vectors associated with Timer\_A:

- □ TACCR0 interrupt vector for TACCR0 CCIFG:
  - TACCR0 CCIFG flag has the highest Timer\_A interrupt priority;
  - The TACCR0 CCIFG flag is automatically reset when the TACCR0 interrupt request is serviced.

□ TAIV interrupt vector for TACCR1 CCIFG to TACCR4 CCIFG and TAIFG:

- The flags are prioritized and combined to source a single interrupt vector (decreasing priority);
- The interrupt vector register TAIV is used to determine which flag requested an interrupt;
- Disabling Timer\_A interrupts does not affect the TAIV value;
- Any access, read or write, of the TAIV register automatically resets the highest pending interrupt flag;
- If another interrupt flag is set, another interrupt is immediately generated after servicing the initial interrupt.

# 7.6 Timer\_B special features

Timer\_B presents some special features:

 $\hfill \ensuremath{\square}$  Programmable length of the TBR register (equivalent to TAR in Timer\_A) to be 8, 10, 12, or 16 bits:

- Configurable through CNTLx bits selection in TBCTL (equivalent to TACTL in Timer\_A);
- The maximum count value, TBR(max), for the selectable lengths is 0FFh, 03FFh, 0FFFh, and 0FFFFh, respectively;

- □ Three or seven capture/compare blocks TBCCRx;
- Double-buffered compare latches with synchronized loading:
  - In Timer\_A, the signal generation in compare mode may cause noise during compare period updates, because the TACRRx value is used directly to compare with timer value;
  - To avoid this condition, the compare latches, TBCLx, buffered by TBCCRx, hold the data for the comparison to the timer value in compare mode;
  - The CLLDx bits in the TBCCTLx register configure the timing of the transfer from TBCCRx to TBCLx.
- Grouping channels capability:
  - Multiple compare latches may be grouped together for simultaneous updates using the TBCLGRPx bits;
  - Two conditions are required:
    - All TBCCRx registers must be updated;
    - $\circ~$  The load event controlled by the CLLDx bits must occur.
- □ All outputs can be put into a high-impedance state:
  - TBOUTH = 1 put Timer\_B outputs into a high-impedance state, allowing higher security and lower delay time answering to failures.
- □ The SCCI bit function is not implemented.

#### 7.6.1 Timer\_B registers

This section describes only the Timer\_B bits of the registers TBCTL and TBCCTL that differ from Timer\_A features. Refer to Timer\_A registers description to determine the common function. The different bits are highlighted in **bold**.

15	14	13	12	11	10	9	8
Unused	TBCLGRP1	TBCLGRPC	CNTL1	CNTLO	Unused	TBSSEL1	TBSSEL0
7	6	5	4	3	2	1	0
ID1	ID0	MC1	MC0	Unused	TBCLI	R TBIE	TBIFG

#### TBCTL, Timer\_B Control Register

Bit		Description
14-	TBCLGRPx	TBCLx group:
13		TBCLGRP1 TBCLGRP0 = $00$
		$\Rightarrow$ Each TBCLx latch loads independently
		TBCLGRP1 TBCLGRP0 = 01
		$\Rightarrow$ TBCL1+TBCL2 (update control: TBCCR1 CLLDx)
		$\Rightarrow$ TBCL3+TBCL4 (update control: TBCCR3 CLLDx)
		$\Rightarrow$ TBCL5+TBCL6 (update control: TBCCR5 CLLDx)
		$\Rightarrow$ TBCL0 independent
		TBCLGRP1 TBCLGRP0 = 10
		$\Rightarrow$ TBCL1+TBCL2+TBCL3 (update control: TBCCR1 CLLDx)
		$\Rightarrow$ TBCL4+TBCL5+TBCL6 (update control: TBCCR4 CLLDx)
		$\Rightarrow$ TBCL0 independent
		TBCLGRP1 TBCLGRP0 = $11$
		$\Rightarrow$ TBCL0+TBCL1+TBCL2+TBCL3+TBCL4+TBCL5+TBCL6
		(update control: TBCCR1 CLLDx)
12-	CNTLx	Counter Length:
11		CNTL1 CNTL0 = 00 $\Rightarrow$ 16-bit, TBR(max) = 0FFFFh
		CNTL1 CNTL0 = 01 $\Rightarrow$ 12-bit, TBR(max) = 0FFFh
		CNTL1 CNTL0 = 10 $\Rightarrow$ 10-bit, TBR(max) = 03FFh
		CNTL1 CNTL0 = 11 $\Rightarrow$ 8-bit, TBR(max) = 0FFh

# TBCCTLx, Timer\_B Capture/Compare Control Register

15	14	13	12	11	10	9	8
CM1	CM0	CCIS1	CCIS0	SCS	CLLD1	CLLDO	CAP
7	6	5	4	3	2	1	0
OUTMOD2	OUTMOD1	OUTMODO	CCIE	CC	I OL	JT COV	CCIFG

TBCLx loads on write to TBCCRx
TBCLx loads when TBR counts to 0
TBCLx loads when TBR counts:
<ul><li>to 0 (up/continuous mode);</li></ul>
<ul> <li>to TBCL0 or to 0 (up/down mode)</li> </ul>
TBCLx loads when TBR counts:
- to TBCLx

# 7.7 Laboratory 3: Timers use

This hands-on laboratory consists of configuring the LCD\_A controller of the MSP430FG4618 device of the Experimenter's board to put a message on the display. Like the previous exercise (Lab2), this laboratory is composed of some sub tasks. This laboratory has been developed only for the *Code Composer Essentials version 3* software development tool.

# 7.7.1 Lab3A\_1: Memory clock with Basic Timer1

#### Project files

C source files:	Chapter 7 > Lab3 > Lab3a_1_student.c
	Chapter 7 > Lab3 > LCD_defs.h
Solution file:	Chapter 7 > Lab3 > Lab3a_1_solution.c

#### Overview

This laboratory implements a memory clock using the features provided by Timer1. The clock is updated once every second by the Basic Timer1 interrupt service routine (ISR). This procedure also performs switching of LED1. In order to evaluate the execution time of the routine, LED2 is kept active during the execution of the ISR. When the ISR has completed, the device goes into low power mode, until the new interrupt wakes it up.

#### A. Resources

This application sets Basic Timer1 to generate an interrupt once every second. The interrupt service routine generated by this peripheral is required to update the clock stored in memory. Moreover, it must refresh the content of the clock displayed on the LCD.

Thus, the system resources used by this application are:

- □ Basic Timer1;
- □ I/O ports;
- LCD;
- □ Interrupts;
- □ Low power modes.

The default configuration of the FLL+ is used, so, all the clock signals required for the operation of the components of the device assume their default values.

# B. Software application organization

The first task is to disable the Watchdog Timer. It should be stated that this feature, when used correctly, makes the application more robust.

The resources needed for the LCD are all configured. This code is given, since its operation will be analysed in a later laboratory. Once the LCD configured, it is cleared by the execution of the routine *LCD\_all\_off()*.

The memory clock consists of setting three global variables: hour, min, and sec, all of the type unsigned char, used to store the hours, minutes and seconds values elapsed respectively since the beginning of the execution of the application. These variables are initialized with zero values.

The LCD is refreshed at startup to show the initial clock value.

LED1 is used as an indicator of Basic Timer1 ISR execution. The execution time can be determined through it. In addition, LED2 state switches whenever the Basic Timer1 ISR is executed.

The Basic Timer1 is set to generate an interrupt once every second.

The routine *main()* ends with the interrupts global activation and puts the device in low power mode, awaiting the next interrupt.

Basic Timer1 ISR begins by activating LED2, indicating the beginning of the routine execution and then switches the state of LED1. The counters are updated in cascade and their contents updated on the LCD, through routines *LCD\_sec()*, *LCD\_min()* and *LCD\_hour()*. The routine ends with switching the state of the clock separation points. Finally, LED2 is turned off.

# C. System configuration

# Disabling the Watchdog Timer

The Watchdog Timer is disabled with the objective of reducing energy consumption, but giving up the protection afforded by it. This peripheral is configured by the WDTCTL register. Its access is protected by a password. What is the value to write to disable it?

WDTCTL = \_\_\_\_;

# □ FLL+ configuration

A 32.768 kHz crystal is applied to the oscillator LFXT1. Since it is possible to select the internal capacitors using software, what is the value to write to the FLL\_CTL0 configuration register to select the 8 pF capacitors?

FLL\_CTL0 |= \_\_\_\_;

Taking into consideration the change mentioned earlier to the FLL+ module, what are the frequencies of each of the clock signals?

ACLK =	;
MCLK =	;
SMCLK =	;

#### □ LED ports configuration

LED1 and LED2 are connected to ports P2.2 and P2.1 respectively. How should they be configured so that just the bits related to these ports have digital output functions?

P2DIR = \_\_\_\_;

How should the P2OUT register be configured so that the application starts with LED1 on and LED2 off?

P2OUT = \_\_\_\_;

#### □ Basic Timer1 configuration

Basic Timer1 should generate an interrupt once every second. It uses two counters in series, so that the input of the BTCNT2 counter is the output of the BTCNT1 counter divided by 256. The BTCNT1 counter input is the ACLK with a 32.768 kHz frequency. If the selected output of the BTCNT2 counter is divided by 128, what is the time period associated with the Basic Timer1 interrupt?

What are the values to write in the configuration registers?

BTCTL = \_\_\_\_; IE2 = \_\_\_\_;

#### □ Low power modes

The task simply updates the counters periodically and refreshes the LCD contents. It is possible to configure the registers for an energy-efficient operation.

Which low power mode should be used?

Which system clocks are activated in the low power mode selected?

#### D. Analysis of operation

Knowing the values and configurations of each register, complete the file LAB3a\_1\_student.c. The complete solution can be found in the file LAB3a\_1\_solution.c. One of these files should be included in the building of the project. Afterwards, compile and debug the project.

#### □ System clocks inspection

The MCLK, SMCLK and ACLK system clocks are available at ports P1.1, P1.4 and P1.5 respectively. These ports are located on the SW2, RESET\_CC and VREG\_EN lines, which are available on the H2 Header pins 2, 5 and 6. All these resources are available because the Chipcon RF module is not installed and SW2 is not used.

Using the Registers view, set bits 1, 4 and 5 of P1SEL and P1DIR registers, to choose the secondary function of these ports configured as outputs. By connecting an oscilloscope to those lines, it is possible to monitor the clock signals.

What are the values measured for each of the system clocks?

ACLK:	 	
SMCLK:	 	

MCLK:

#### □ ISR execution time

The Basic Timer1 ISR execution time is fundamental to energy conservation, in order to extend the life of the system battery. The routine execution time can be measured by connecting the oscilloscope to port P2.1, which controls LED2. This output is available on pin 2 of Header H4.

The execution time of this routine varies with the number of the counter updates and respective updates to the LCD. What are the times measured for each of these situations and what their frequencies?

Seconds update: \_\_\_\_\_ with a time period of \_\_\_\_\_

Seconds and minutes update: \_\_\_\_\_ with a time period of \_\_\_\_\_

LCD fields update: \_\_\_\_\_ with a time period of \_\_\_\_\_

If the developer chooses to update all the LCD fields at each interrupt, the time required is much greater than the solution presented. Efficient programming contributes to a reduction in the system power consumption.

#### □ Measurement of electrical current drawn

The power consumption was discussed in the previous point. The electrical power required by the system during operation is measured by replacing the jumper on the Header PWR1 by an ammeter, which indicates the electric current taken by device during operation.

What is the value read? \_\_\_\_\_

```
MSP-EXP430FG4618
                                                 SOLUTION
Implement a memory clock using the features supported by
Timer1.
Disabling the Watchdog Timer:
WDTCTL = WDTPW | WDTHOLD;
                               // Stop WDT
□ FLL+ configuration:
FLL_CTL0 |= XCAP18PF;
                              // Set load cap for 32k xtal
□ LED ports configuration:
P2DIR | = 0 \times 06;
                               // P2.2 and P2.1 as output
P2OUT | = 0 \times 04;
                               // LED1 on and LED2 off
□ Basic Timer1 configuration:
BTCTL = BTDIV | BT_fCLK2_DIV128; // (ACLK/256)/128
IE2 |= BTIE;
                           // Enable Basic Timer1 interrupt
□ Low power modes:
BIS_SR(LPM3_bits + GIE); // Enter LPM3 with interrupts
enabled
```

# 7.7.2 Lab3A\_2: Real Time Clock with Basic Timer1

#### **Project files**

C source files:	Chapter 7 > Lab3 > Lab3a_2_student.c
	Chapter 7 > Lab3 > LCD_defs.h
Solution file:	Chapter 7 > Lab3 > Lab3a_2_solution.c

#### Overview

The Real Time Clock (RTC) has a 32-bit counter, to automatically control the clock calendar. This peripheral is present on the MSP430FG461x devices. The application developed in the previous laboratory will now be modified to incorporate this module.

#### A. Resources

This application is based on the same resources used in the previous laboratory. In addition, there is an additional RTC peripheral and two push buttons, SW1 and SW2. The first module works in automatic mode to manage the clock calendar, while the push buttons switch the information displayed on the LCD between the clock and calendar.

#### B. Software application organization

The organization of the software is identical to that of *LAB3A\_1* laboratory. The Basic Timer1, LCD and LEDs continue to perform the same functions. They are configured similarly, but with the changes described below.

In routine *main()*, the configurations for RTC and SW1/SW2 are added.

The memory addresses corresponding to the clock calendar values are initialized with the default values, that is zero hours, zero minutes and zero seconds, on August 9, 2008. The RTC is then activated in calendar mode, with the interrupt disabled. This mode affects the Basic Timer1 operation.

The switches SW1 and SW2 are connected to the microcontroller ports P1.0 and P1.1 respectively. Hence, these ports are configured as inputs and their interrupts activated by a high-to-low transition at the input.

# C. System configuration

# **Gamma** Real Time Clock configuration

The RTC is configured in calendar mode and enabled. The counting registers provide the values of seconds, minutes, hours, days, day of the week, day of the month, month and year. The registers are stored in BCD format to speed up the data writing process to the LCD. The interrupt for this peripheral should be disabled (disabling the Basic Timer1 interrupt). Given these objectives, what is the configuration value of the following register?

RTCCTL = \_\_\_\_;

The RTC operation in calendar mode automatically configures some of the Basic Timer1 features. The content of the bits BTSSEL, BTHOLD and BTDIV of BTCNT register are ignored. Thus, the BTCNT1 and BTCNT2 counters work in cascade. The clock source of the BTCNT1 counter is the ACLK clock signal. The output of the BTCNT1.Q7 counter is selected as the input of the BTCNT2 counter (frequency: ACLK/256). The RTC uses the BTCNT2.Q6 output as clock source (frequency: ACLK/32768).

#### Basic Timer1 configuration

This peripheral is automatically configured with the RTC in calendar mode. To enable the interrupt once every 0.5 seconds, what is the value to write to the following register:

BTCNT = \_\_\_\_;

#### Ports P1.0 and P1.1 configuration

The switches SW1 and SW2 are connected to ports P1.0 and P1.1 respectively. How should the following registers be configured in order to set just the bits that affect the digital inputs, with high-to-low transition interrupts?

P1SEL &=	_;
P1DIR &=	_;
P1IFG =	_;
P1ES &=	_;
P1IE  =	_;

#### D. Analysis of the operation

Knowing what values and configurations to give each of the device's registers, complete the blank spaces in the file **LAB3a\_2\_student.c**. The solution of the laboratory can be found in the file **LAB3a\_2\_solution.c**. One of these files should be included in the construction of the project. After compiling, debug the project.

#### □ ISR execution time

Performing similar procedures to those described in laboratory *Lab3A\_1* measure the ISR execution time. What is the value measured?

LCD refresh: \_\_\_\_\_

The LCD write routines were changed. Taking advantage of storing the data in the BCD format, the division operation can be ignored, resulting in the reduction of execution time of the Basic Timer1 ISR. Is the processing time required to refresh the LCD constant?

#### Measurement of electrical current drawn

Using the procedure similar to that described at the corresponding point of *Lab3A\_1*, measure of the value of current drawn by the device.

What is the value measured? \_\_\_\_\_

SOLUTION

#### MSP-EXP430FG4618

Implement a Real Time Clock (RTC) using Basic Timer1.

# Real Time Clock configuration: RTCCTL = RTCBCD | RTCHOLD | RTCMODE\_3; // BCD mode, RTC and BT disable

# □ Basic Timer1 configuration:

BTCTL = BT_fCLK2_DIV	764;		// (ACLE	K/256)	)/64	
IE2  = BTIE;	// Ena	ble BT	interrupt	with	0.5	period

#### □ Ports P1.0 and P1.1 configuration:

P1SEL &= ~0x03;	// P1.0 and P1.1 as inputs
P1DIR &= ~0x03;	// Pl.0 and Pl.1 digital inputs
P1IFG = 0x00;	
P1IES &= ~0x03;	// high-to-low transition interrupts
P1IE  = 0x03;	// enable port interrupts

# 7.7.3 Lab3B: Memory Clock with Timer\_A

# **Project files**

C source files:	Chapter 7 > Lab3 > Lab3b_student.c
	Chapter 7 > Lab3 > LCD_defs.h
Solution file:	Chapter 7 > Lab3 > Lab3b_solution.c

# Overview

The objective of this laboratory is to build a memory clock similar to the one that was developed using the Basic Timer1, in laboratory *Lab3A\_1*. Timer\_A is configured to generate an interrupt once every 100 msec. The ISR manages the memory clock. LED1 and LED2 are used to monitor the operation of the system state.

#### A. Resources

This application makes use of Timer\_A to generate an interrupt when the value in the TACCR0 unit is reached. The ISR updates the contents of the memory clock variables.

LED1 monitors the system operation, switching state whenever the Timer\_A ISR runs. LED2 can be used to monitor the ISR execution time. The contents of the LCD is updated every interrupt. When the ISR finishes, the device returns to low power mode.

Hence, the system resources used by this application are:

- □ Timer\_A;
- □ I/O ports;
- □ LCD;
- □ Interrupts;
- □ Low power modes.

The default configuration of the FLL+ is used, so all the clock signals required for the operation of the device assume their default values.

#### B. Software application organization

The first task is to disable the Watchdog Timer. All the resources needed for the LCD are then configured. The complete code is given, because its operation will be analysed in a later laboratory. Once configured, the LCD is cleared by the execution of the routine *LCD\_all\_off()*.

The memory clock consists of three global variables: min, sec, msec, of the type unsigned char, to store the minutes, seconds and milliseconds respectively of the values elapsed since the beginning of the execution of the application. These variables are initialized with zeros.

The LCD is refreshed at startup to display the initial clock value.

LED2 is used as an indicator of Timer\_A ISR execution. The execution time can be monitored using it. In addition, LED1 switches state whenever Timer\_A ISR is executed.

Timer\_A is configured to generate an interrupt once every 100 milliseconds.

The routine *main()* ends with a global interrupt enable and puts the device into a low power mode, where it waits for the next interrupt.

Timer\_A ISR begins by activating LED2, indicating the beginning of execution of the routine and then switches LED1 state. The counters are updated in cascade and their contents are used to update the LCD, through the routines *LCD\_msec()*, *LCD\_sec()* and *LCD\_min()*. The routine ends by switching the state of the clock separation points. Finally, LED2 is turned off.

# C. System configuration

#### Disable the Watchdog Timer

The Watchdog Timer is configured as in the above examples.

#### □ FLL+ configuration

FLL+ is configured as in the above examples.

#### □ LED ports configuration

LED ports are configured as in the above examples.

#### □ Timer\_A configuration

The Timer\_A is configured to count until it reaches the value written in the TACCR0 unit. An interrupt is generated when it reaches that value. Which is the interrupt vector to use? \_\_\_\_\_

Timer\_A clock signal is the ACLK without division. What is the value to write in the configuration register?

TACTL = \_\_\_\_;

The TACCR0 capture/compare unit determines the Timer\_A counting range. For a 100 msec response, what is the value to write in the register?

TACCR0 = \_\_\_\_;

The interrupt is configured in TACCR0 capture/compare unit. What is the value to write to the following register?

TACCTL0 = \_\_\_\_;

#### □ Low power mode

The low power mode is configured as in the above examples.

#### D. Operation analysis

Knowing the values and configurations for each register, complete the file **LAB3b\_student.c.** The complete solution can be found in the file **LAB3b\_solution.c.** One of these files should be included in building the project. After compiling, debug the project.

#### □ ISR execution time

Using similar procedures to those described in laboratory Lab3A measure the ISR execution time. What is the value measured?

LCD refresh: \_\_\_\_\_

#### Measurement of electrical current drawn

Using a procedure similar to that described at the corresponding point of *Lab3A\_1*, measure of the value of current drawn by the device.

What is the value measured? \_\_\_\_\_

MSP-EXP430FG4618 SOLUTION

Implement a Memory Clock using Timer\_A.

Timer\_A configuration:

```
TACTL = TASSEL_1 | MC_1 | ID_0; // ACLK, up mode
TACCR0 = 3268; // this count corresponds to 100 msec
TACCTL0 = CCIE; // TACCR0 interrupt enabled
```

# 7.7.4 Lab3C: Buzzer tone generator

#### Project files

C source files:	Chapter 7 > Lab3 > Lab3c_student.c
Solution file:	Chapter 7 > Lab3 > Lab3c_solution.c

#### Overview

The purpose of this laboratory is to build a sound generator using Timer\_B. The PWM signal produced by this peripheral drives the buzzer, producing a sequence of musical notes at regular time intervals. At the same time, LED1 and LED2 switch state alternately. The volume of sound produced by the buzzer can be controlled by push buttons SW1 and SW2.

#### A. Resources

The implementation of this application requires the production of specific frequency signals corresponding to musical notes. For each frequency, the duty-cycle can be modified in order to control the volume of sound produced. This task is carried out using Timer\_B and one of its compare units. The buzzer is operated by Port P3.5, configured to work in its special function as TB4 compare unit output. This output corresponds to the TBCCR4 output compare unit.

The push buttons SW1 and SW2 are connected to ports P1.0 and P1.1 respectively. An interrupt is generated when either of these

buttons are pressed. The duty-cycle of the generated note is modified in response.

Basic Timer1 is configured to generate an interrupt once every second. The interrupt service routine updates the musical notes produced by the buzzer, which are stored in an array.

LED1 and LED2 are driven from P2.2 and P2.1 respectively, and their state is switched alternately once every second.

The module FLL+ is configured to a 7.995392 MHz frequency, for the MCLK and SMCLK clock signals.

The resources used by the application are:

- □ Timer\_B;
- □ Basic Timer1;
- □ I/O ports;
- □ FLL+;
- □ Interrupts.

#### B. Software application organization

The application consists of the routine *main()*, which is used to configure all system resources, before entering into a standby mode, waiting for one of two interrupts.

This routine starts by disabling the watchdog timer and starting the module FLL+ to produce the desired clock signals of the correct frequency for the SMCLK and MCLK. Then, the Basic Timer1 and Timer\_B are configured in order to perform the desired functions.

The ports connected to the LEDs, buttons and buzzer are then initialized.

Finally, the interrupts are activated, and the application waits for the execution of one of two interrupts.

The Basic Timer1 interrupt executes at a frequency of once every second. When this interrupt is occurs, it begins by switching the state of LED1 and LED2. Afterwards, it accesses the memory to fetch the next musical note to be performed. The routine ends with memory pointer management.

The Port 1 ISR begins by evaluating the source of the interrupt. The sound volume is reduced if the button SW1 is pressed. The sound volume is increased if button SW2 is pressed.

#### C. System configuration

#### □ Timer\_B

It is the responsibility of Timer\_B to produce the PWM signal that activates the Buzzer. Timer\_B counts until the value contained in the TBCCR0 register is reached. It does not generate an interrupt, and must be sourced by SMCLK clock signal.

What is the value to write to this configuration register?

TBCTL = \_\_\_\_;

Each PWM signal produced by Timer\_B corresponds to a musical note. The relationship between the frequency and the musical note is given in *Table 7-3*.

Table 7-3. Relationship table between the frequency and the musical note.

Note	SI0	DO	RE	MI	FA	SOL	LA	SI	D02
Freq [Hz]	503	524	587	662	701	787	878	1004	1048

Timer\_B has a frequency clock input equal to 7.995392 MHz.

What is the value to write in the TBCCR0 register in order to generate the desired frequency?

Table 7-4. TBCCR0	value of	the musical	notes.
-------------------	----------	-------------	--------

Note	SI0	DO	RE	MI	FA	SOL	LA	SI	D02
TBCCR0									

The TBCCR4 compare unit is used to produce the PWM signal. The set/reset compare mode is used.

What is the value to write to the configuration register?

TBCCTL4 = \_\_\_\_;

The volume control consists of varying the PWM signal duty-cycle. Initially its default value is 50%. What is the configuration value to write to the register?

TBCCR4 = \_\_\_\_;

#### Basic Timer1

The Basic Timer1 generates an interrupt once every second. It uses two counters in series, where the BTCNT2 counter input uses the BTCNT1 counter output divided by 256. The BTCNT1 counter input is the ACLK clock signal with a frequency of 32.768 kHz.

If BTCNT2 counter selected output is divided by 128, what is the time period associated with the Basic Timer1 interrupt? \_\_\_\_\_

What are the values to write to the configuration registers?

BTCTL = \_\_\_\_; IE2 = \_\_\_\_;

# □ I/O Ports

Three ports are used by this application. In port P1, bits P1.0 and P1.2 are used to activate the ISR whenever the buttons SW1 and SW2 are activated.

How should just the bits related to these ports be configured in order to have digital input functions, with high-to-low transition interrupts?

P1SEL	&= .	·	;
P1DIR	&= .		;
Plifg	=		;
P1IES	&= .		;
P1IE	=		;

LED1 and LED2 are connected to ports P2.2 and P2.1 respectively. How should just the bits related to these ports be configured in order to have digital output functions?

P2DIR = \_\_\_\_;

Configure the P2OUT register in order to initialize the application with the LED1 on and the LED2 off.

P2OUT = \_\_\_\_;

Configure the P3 register, with P3.5 connected to the buzzer. Remember to configure it as special function output (Timer\_B compare output – TB4). P3SEL = \_\_\_\_\_; P3DIR = \_\_\_\_\_;

#### FLL+ configuration

This module uses the 32.768 kHz frequency crystal to produce a 7.995392 MHz frequency at the SMCLK and MCLK clock signals. What are the values to write to the configuration registers?

FLL_CTL0	=	;
SCFIO  =		;
SCFQCTL =	:	;

#### D. Analysis of operation

#### **G** System clocks inspection

The MCLK, SMCLK and ACLK system clocks are available at ports P1.1, P1.4 and P1.5 respectively. These ports are located on the SW2, RESET\_CC and VREG\_EN lines, which are available on the H2 Header pins 2, 5 and 6. All these resources are available because the Chipcon RF module is not installed and SW2 is not used.

Using the Registers view, set bits 1, 4 and 5 of P1SEL and P1DIR registers to choose the secondary function of their ports, that is, configured as outputs. Connect an oscilloscope probe at these positions to monitor the clock signals.

What are the values measured for each of the system clocks?

ACLK: \_\_\_\_\_\_

MCLK:

#### □ TBCCR4 unit output frequency

With the help of an oscilloscope, it is possible to evaluate the operation of the application. Alternatively, it is possible to listen to the sound produced. By removing jumper JP1 and connecting the oscilloscope to this pin, it is possible to view the PWM signal produced by the microcontroller. The duty-cycle can be reduced or increased by pressing the push buttons SW1 and SW2.

#### □ Port P1 interrupt source decoding

All Port P1 interrupt lines share the same interrupt vector. The decoding is done through the P1IFG register.

This process can be observed by entering a breakpoint at the first line of the ISR code.

Execute the application.

The application's execution is suspended at the breakpoint by pressing either button SW1 or SW2. From this point onwards, run the lines of code step-by-step and observe changes in the register values.

#### Measurement of electrical current drawn

Using a procedure similar to that described at the corresponding point of *Lab3A\_1*, measure of the value of current drawn by the device.

What is the value read? \_\_\_\_\_

# MSP-EXP430FG4618

SOLUTION

Implement a Buzzer tone generator.

Timer_B configuration:
// SMCLK, continuous mode
TBCTL = TBSSEL_2   CNTL_0   TBCLGRP_0  MC_1   ID_0;
// TBCCR0 value of the musical notes
#define SI0 15895
#define DO 15258
#define RE 13620
#define MI 12077
#define FA 11405
#define SOL 10159
#define LA 9106
#define SI 7963
#define DO2 7629
TBCCTL4 = OUTMOD_3; // CCR4 interrupt enabled
TBCCR4 = space[0]/2;

□ Basic Timer1 configuration:

```
BTCTL = BTDIV | BT_fCLK2_DIV128; // (ACLK/256)/128
IE2 |= BTIE; // enable BT interrupt
```

# □ I/O Ports configuration:

```
PlIES &= 0xFF // high-to-low transition interrupt
PlIE |= 0xFF; // enable port interrupts
// LED1 and LED2 configuration (Port2):
P2DIR |= 0x06; // P2.2 and P2.1 as outputs
P2OUT = 0x04; // LED1 on and LED2 off
// Buzzer port configuration (Port3)
P3SEL |= 0x20; // P3.5 as special function
P3DIR |= 0x20; // P3.5 as digital output
```

```
FLL_CTL0 |= DCOPLUS + XCAP18PF; //DCO+ set,freq=xtal*D*N+1
SCFI0 |= FN_4; // x2 DCO freq, 8MHz nominal DCO
SCFQCTL = 121; // (121+1) x 32768 x 2 = 7.99 MHz
```

#### 7.7.5 Lab3D: Frequency measurement

#### **Project files**

□ FLL+ configuration:

C source files:	Chapter 7 > Lab3 > Labd_student.c
	Chapter 7 > Lab3 > LCD_defs.h
Solution file:	Chapter 7 > Lab3 > Labd_solution.c

#### Overview

This laboratory implements an application designed to measure a PWM signal frequency. If a signal generator is not available, the microcontroller generates a PWM signal based on the frequencies stored in a file. The frequencies generated are read and updated with a fixed time period using the features of CCE. The measured value is shown on the LCD in Hz.

#### A. Resources

The module FLL+ is configured to a frequency of 7.995392 MHz for the MCLK and SMCLK clock signals. This application performs the two tasks simultaneously.

On the one hand, it generates a PWM signal with a frequency of 200 Hz and a duty cycle of 50%. Alternatively, the PWM signal frequency can be read from a file using a breakpoint. This function is performed by Timer\_B, using the compare unit to generate the PWM signal.

The time period between two consecutive PWM signals low-to-high transitions is measured by Timer\_A. The capture unit of this timer is configured to collect the Timer\_A counter register's contents when a PWM signal low-to-high transition is detected at its input.

The Basic Timer1 generates an interrupt once every second. The ISR updates the PWM signal frequency generated by the Timer\_B. If you choose to use this feature, a breakpoint associated with this ISR execution allows reading a file with the value of the frequency that will be generated.

The microcontroller's ports are configured in order that the PWM signal generated by Timer\_B through the TBCCR4 compare unit available at Port P3.5/TB4 can be connected to the Port P1.2/TA1 of the Timer\_A TACCR1 capture unit. If you plan to use this feature, these pins must be connected together. Port P3.5 pin is available on Header 7 pin 6, while the Port P1.2 pin is available on Header H2 pin 3.

Ports P2.1 and P2.2 are used to monitor the state of the LED2 and LED1, respectively.

The resources used by the application are:

- □ Timer\_A;
- □ Timer\_B;
- □ Basic Timer1;
- □ I/O ports;
- □ FLL+;
- □ Interrupts.

# B. Software application organization

The software structure allows various tasks to be performed simultaneously. The routine main(O) is responsible for configuring all the resources used by the application. Once started, the application enables all the interrupts and waits for an interrupt request.

There are two routines that separately service the two possible interrupts. The routine *TimerA1\_ISRO* services interrupts required by the Timer\_A overflow and by the TACCR1 capture unit. For every interrupt caused by a TACCR1 capture, the value collected in the TACCR1 register is stored in *T1*, if it is the first low-to-high transition, or stored in *T2* if it is the second low-to-high transition. This sequence is controlled by the variable *capture*. The variable *flag* is used to flag the measurement process. This process occurs between the capture of the first low-to-high transition and the second transition. The counting of clock pulses is done by Timer\_A, in the time interval between the T1 and T2 acquisition, assigned to the variable *T*. The process is synchronized when Timer\_A

overflows, restarting the measurement process. The LCD is refreshed once every 0.5 seconds with the latest measured frequency value, using the control variable control tick that corresponds to 0.5 seconds.

The routine **basic\_timer\_ISR()** services the interrupt produced by Basic Timer1 once every second. This routine begins by switching the state of LED1 and LED2. In addition, it updates the Timer\_B counting period. The variable **read\_data** allows the counting period to be changed.

# C. System configuration

#### Basic Timer1

Basic Timer1 generates an interrupt once every second. Use the two counters in series, where the BTCNT2 counter input is selected as the BTCNT1 counter output divided by 256. The BTCNT1 counter input is the ACLK clock signal with a frequency of 32.768 kHz.

If BTCNT2 counter selected output is divided by 128, what is the time period associated with the Basic Timer1 interrupt? \_\_\_\_\_

What are the values to write to the configuration registers?

BTCTL = \_\_\_\_\_; IE2 = \_\_\_\_\_;

# □ Timer\_B

The TBCCR4 compare unit is used to generate the PWM signal. The set/reset compare mode is used.

What is the value to write to the configuration register?

TBCCTL4 = \_\_\_\_;

The TB4 PWM output signal has a frequency X, with a 50% duty-cycle. The SMCLK clock signal is used as input of Timer\_B.

What are the values to write to the registers?

TBCCR0	=	 ;
TBCCR4	=	;

What the largest and lowest generated frequency?

Maximum frequency value: \_\_\_\_\_\_

#### □ Timer\_A

Timer\_A is sourced by the SMCLK clock signal. It counts to the value 0xFFF, in continuous mode. An interrupt is generated when the TAR counter overflows. What is the value to write to its configuration register?

TACTL = \_\_\_\_;

The capture unit captures the TAR register value to the TACCR1 register when it detects a low-to-high transition at the TA1 input. What is the value to write to the configuration register?

TACCTL1 = \_\_\_\_;

Determine the maximum and minimum frequency values detected. Note that these values do not take into account the execution time of the application. The PWM signals should be applied at frequencies well below the maximum value determined.

Maximum frequency value: \_\_\_\_\_\_

The TACCR1 capture unit is configured to generate an interrupt when it detects a low-to-high transition. What is the value to write to the configuration register?

TACCTL1 = \_\_\_\_;

#### □ Ports P3.5/TB4 and P1.2/TA1 configuration

These ports perform special functions. Thus, the Port P3.5 is configured as an output, selected by the special function TB4, with the values:

P3SEL = \_\_\_\_; P3DIR = \_\_\_\_;

The Port P1.2 is configured as input, with the special function TA1, using the values:

P1SEL = \_\_\_\_;

P1DIR = \_\_\_\_;

#### LED ports configuration

The LEDs should be configured as in *Lab3C*.

#### D. Analysis of operation

# $\hfill\square$ Run the application using the frequency generator based on Timer\_B

Without a frequency generator, the Timer\_B generates a PWM signal at the TBCCR4 unit output that can be fed back to Timer\_A TACCR1 capture unit input. These two pins must therefore be connected together. By default, the PWM signal frequency is 200 Hz. Add a breakpoint at the line of code 233 belonging to the Basic Timer1 ISR to modify this value.

TBCCR0 = 7995392/read\_data;

If the variable *read\_data* has the value 200, it will generate a 200 Hz frequency. The value of this variable can be changed by associating a breakpoint to that line of code. Before the line of code is executed, the value of the data file is read and assigned to the variable *read\_data*. The signal will oscillate at the desired frequency, loading the value in TBCCR0. The breakpoint configuration is as follows:

- □ Action: read data from file
- □ File: address of the data file (example in freq.txt)

 $\hfill\square$  Wrap Around: activate this option to restart reading at the beginning

- □ Start address: &read\_data
- □ Length: 1 in order to read a value from the file each time

Alternatively, the breakpoint can be imported from the file *Lab3d\_breakpoint.bkpt* to debug. The file has a format already described in the chapter devoted to the CCE. Each line describes in Hz the value of the PWM signal frequency.

#### **Q** Run the application using a frequency generator

The operation of the application can be verified using a frequency generator. The generator should generate a PWM signal with voltage and frequency values compatible with the device's input (2.5 to 3.3 volts).

#### Observe the measured frequency

The PWM signal applied to the TA1 input can be viewed using an oscilloscope, connected to pin 3 of Header 2. Perform this task and confirm the values present at the LCD.

#### □ Measurement of electrical current drawn

Carry out the procedure as described in previous laboratories for measuring the device current.

What is the value read? \_\_\_\_\_

MSP-EXP430FG4618

SOLUTION

Perform a PWM signal frequency measurement.

#### □ Basic Timer1 configuration:

```
BTCTL = BTDIV | BT_fCLK2_DIV128; // (ACLK/256)/128
IE2 |= BTIE; // Enable BT interrupt with 1 sec period
```

#### □ Timer\_B configuration:

TBCTL = TBSSEL\_2 | CNTL\_0 | TBCLGRP\_0 |MC\_1 | ID\_0; // SMCLK, continuous mode TBCCTL4 = OUTMOD\_3; // CCR4 output mode 3 (set/reset) TBCCR0 = 39977;// Output 200 Hz signal with 50% duty cycle TBCCR4 = TBCCR0/2;

#### □ Timer\_A configuration:

#### □ Ports P3.5/TB4 and P1.2/TA1 configuration:

// TB4 configuration (Port3)
P3SEL = 0x20; // P3.5 as special function (TB4)
P3DIR = 0x20; // P3.5 as output
// TA1 (TACCR1) configuration (Port1)
P1SEL = 0x04; // P1.2 as special function (TA1)
P1DIR = 0x00; // P1.2 as input

# 7.8 Quiz

**1**. The timer/counter suitable for LCD controller frame frequency generation in the MSP430FG4618 is:

- (a) Timer\_A;
- (b) BTCNT2;
- (c) Timer\_B;
- (d) BTCNT1.

**2.** To set Timer\_A to repeatedly count from 0x000 to 0xFFFF, the operating mode selected must be:

- (a) Up/Down mode (MCx = 3);
- (b) Up mode (MCx = 1);
- (c) Continuous (MCx = 2);
- (d) Stop (MCx = 0).
- 3. Timer\_A configured in continuous operating mode is reset with:
- (a) Write 0xFFFF to TAR register;
- (b) Reset TACCR0;
- (c) Set TACLR bit in the TACTL register;
- (d) None of the above.

**4**. When TASSELx = 1, the timer (Timer\_A or Timer\_B) is sourced by:

- (a) ACLK;
- (b) TACLK;
- (c) INCLK;
- (d) SMCLK.

**5**. To generate a PWM signal with active high state, it is necessary to configure the output operating mode to:

- (a) Mode 2;
- (b) Mode 3;
- (c) Mode 6;
- (d) Mode 7.

**6.** With Timer\_A sourced by the ACLK (32768 Hz) and is configured in compare mode, the value in TACCR0 register to enable interrupts once each second is:

(a) 32768;

(b) 32767;

(c) 16384;

(d) 65536;

# (d) .0 ;(b) .2 ;(c) .4 ;(c) .5 ;(c) .1 :noitulo2

#### 7.9 FAQs

1. What are the Basic Timer1 registers initial condition?

None. They must be configured by user software before use.

**2.** What is the result of reading or writing to BTCNT1 and to BTCNT2 when CPU clock and counter clock are asynchronous:

Any read may be unpredictable. Any write take effect immediately.

3. Can Timer\_A registers be modified during its operation?

Yes, if they are the TAIE, TAIFG or TACLR. For other configuration types, it is recommended to stop the timer before modifying its operation to avoid erroneous operating conditions.

**4.** Can the output mode be switched during the operation of a timer?

It is recommended to the remaining bits of OUTMODx bits to 1s during the transition between output modes, or otherwise to configure mode 7 as transition state while switching between output modes.