ECE 3567 Microcontrollers Lab

Laboratory #2 – Timers and Interrupts

Spring 2020

Dr. Gregg Chapman

BACKGROUND

ECE 3567 LABS

Department of Electrical & Computer Engineering

THE OHIO STATE UNIVERSITY

COLLEGE OF ENGINEERING

HOME USEFUL DOCUMENTS BROKEN LAB INFO THE LABORATORIES

AUGUST 16, 2019

Welcome to ECE 3567

by Gregg Chapman at 4:01pm

Spring 2020 Labs will begin on Monday, January 6th. There are lectures during the lab time for the first 3 weeks of the course. You must sign in to receive credit (please see Syllabus , available here, under Lab Info, and on Carmen).

Please check Carmen regularly for any announcements.

Best Regards – Dr. Gregg Chapman

Leave a comment

Useful Documents

MSP430FR6989 Users Guide – slau367o

Launchpad Quick Start Guide – slau626

MSP430FR6989 Launchpad Development Kit – slau627a

NEW: msp430fr6989 MCU Specific Datasheet

Standard Header File ECE 3567 Autumn 2019

PxSEL Settings, P1.6

Table 6-21. Port P1 (P1.4 to P1.7) Pin Functions

PIN NAME (P1.x)	x	FUNCTION	CONTROL BITS AND SIGNALS (1)			
			P1DIR.x	P1SEL1.x	P1SEL0.x	LCDSz
P1.4/UCB0CLK/UCA0STE/TA1.0/Sz	4	P1.4 (I/O)	I: 0; O: 1	0	0	0
		UCB0CLK	X (2)	0	1	0
		UCA0STE	X (3)	1	0	0
		TA1.CCI0A	0	1	1	0
		TA1.0	1			
		Sz (4)	X	X	X	1
P1.5/UCB0STE/UCA0CLK/TA0.0/Sz	5	P1.5 (I/O)	I: 0; O: 1	0	0	0
		UCB0STE	X (2)	0	1	0
		UCA0CLK	X (3)	1	0	0
		TA0.CCI0A	0	1	1	0
		TA0.0	1			
		Sz (4)	X	X	X	1
P1.6/UCB0SIMO/UCB0SDA/TA0.1/Sz	6	P1.6 (I/O)	I: 0; O: 1	0	0	0
		UCB0SIMO/UCB0SDA	X (2)	0	1	0
		N/A	0	1	0	0
		Internally tied to DVSS	1			
		Sz (4)	X	X	X	1
P1.7/UCB0SOMI/UCB0SCL/TA0.2/Sz	7	P1.7 (I/O)	I: 0; O: 1	0	0	0
		UCB0SOMI/UCB0SCL	X (2)	0	1	0
		N/A	0	1	0	0
		Internally tied to DVSS	1			
		TA0.CCI2A	0	1	1	0
		TA0.2	1			
		Sz (4)	X	X	X	1

(1) X = Don't care
(2) Direction controlled by eUSCI_B0 module.
(3) Direction controlled by eUSCI_A0 module.
(4) The associated LCD segment is package dependent. See the Signal Descriptions tables and Pin Diagrams figures.

Useful Documents

MSP430FR6989 Users Guide – slau367o

Launchpad Quick Start Guide – slau626

MSP430FR6989 Launchpad Development Kit – slau627a

NEW: msp430fr6989 MCU Specific Datasheet

Standard Header File ECE 3567 Autumn 2019

Clock Modules

Here is some background on the MCU clock configurations. Use the ACLK for Timer A0

- Four Internal Clocks (Can be linked to CLK sources and adjusted) — Typical Values
  - MCLK Master Clocks
  - SMCLK Subsystem Master Clock
  - MODCLK Module Clock
  - ACLK Auxiliary Clock — 32.768 KHz

- Two External Clock Sources
  - LFXTCLK (Low Frequency XTALS) — 32.768 KHz
  - HFXTC (High Frequency XTALS) — 4 – 24 MHz

- Internal Clock Sources
  - DCOCLK Digitally Controlled Oscillator — 2.7-24 MHZ (16MHz typ.)
  - VLOCLK Very Low Power Clock — 10 KHz
  - MODCLK Module Clock — 5 MHz

Timer Hierarchy

Choosing Functionality

CAPTURE or COMPARE Modes

- Capture
- Compare

TIMER Counting Modes

- Stop
- Up
- Continuous
- Up/Down

OUTPUT Modes

- Output
- Set
- Toggle/Reset
- Set/Reset
- Toggle
- Reset
- Toggle/Set
- Reset/Set

OUTPUT MODE 7: Pulse Width Modulation

★ UP Mode

25.2.3.1 Up Mode

The up mode is used if the timer period must be different from 0FFFFh counts. The timer repeatedly counts up to the value of compare register TAxCCR0, which defines the period (see Figure 25-2). The number of timer counts in the period is TAxCCR0 + 1. When the timer value equals TAxCCR0, the timer restarts counting from zero. If up mode is selected when the timer value is greater than TAxCCR0, the timer immediately restarts counting from zero.

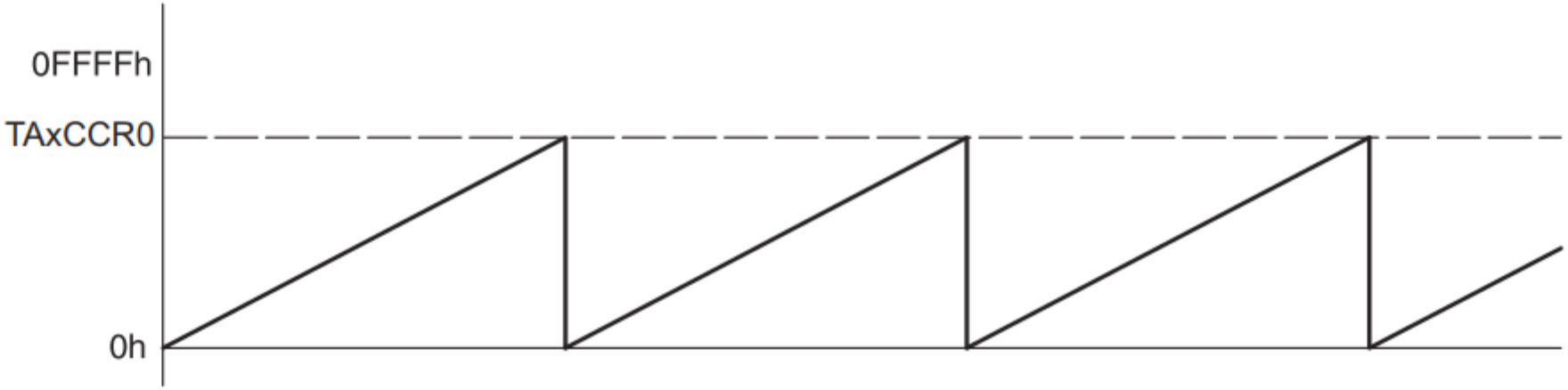


Figure 25-2. Up Mode

The TAxCCR0 CCIFG interrupt flag is set when the timer *counts* to the TAxCCR0 value. The TAIFG interrupt flag is set when the timer *counts* from TAxCCR0 to zero. Figure 25-3 shows the flag set cycle.

Output Examples

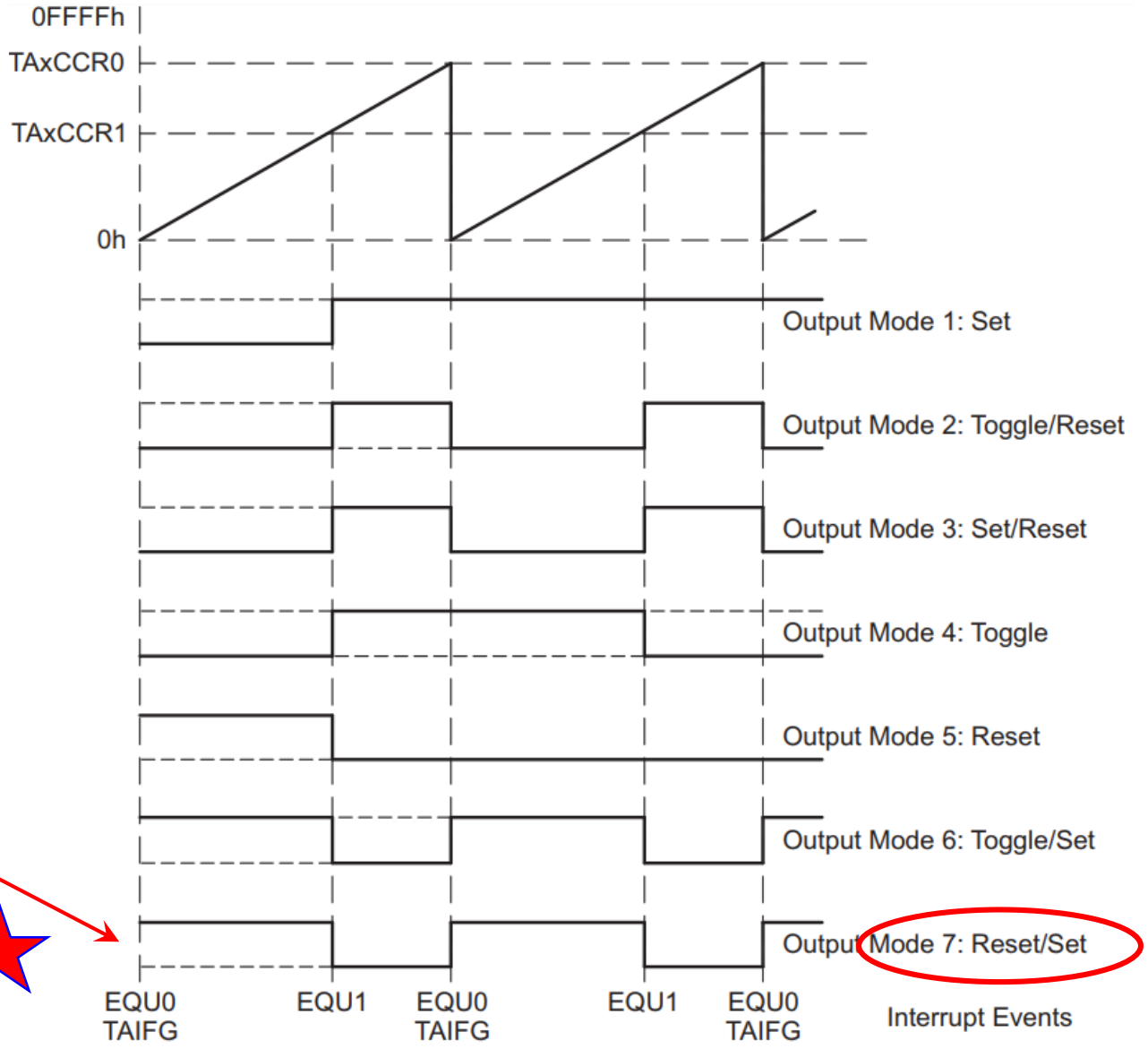


Figure 25-12. Output Example – Timer in Up Mode

"Pulse Width Modulation"

Project Set-up

1. Create a Lab 2 WorkSpace on the U: drive
2. Select File → Switch Workspace → Other, then navigate to your Lab 2 workspace and click LAUNCH
3. At the Getting Started Screen, Selet Project →New CCS Project
4. In the CCS Project window set Target to **MSP430FRxxx** and select **MSP430FR6989**
5. In the Project Templates and Examples window, scroll down to **MPS430 DriverLib** and select the ***Empty Project with DriverLib Source*** beneath that level.
6. Enter the project name (Lab2) and click Finish.
   NOTE: the project folder must match the project Name
7. Remember to Switch Workspace
8. Copy the main.c and 3567.h from Lab 1 into the Lab 2 project folder. This will save you an incredible amount of time. GET THIS WORKING AGAIN FIRST.
7. Select Project → Rebuild Project
8. At this point it is essential to connect the hardware
9. Make sure that the Project is selected as [Active – Debug]
10. Select the Debug ICON
11. Once the GREEN ARROW comes up you can run the code
12. Halt execution with the RED SQUARE

ECE 3567 – Lab #2

Checkpoint #1: Demonstrate that the Lab #1 project is operating correctly in the Lab 2 environment before you begin to edit the code.

1. One LED should flash at a time.
2. The GREEN LED should be the default after initialization
3. The LEDs should alternate, RED .. GREEN .. RED at a dismal approximation of 1 Hz.

ECE 3567 – Lab #2
Additional Files Needed

1. Download the Lab2.zip under Lab 2 on the ECE 3567 website and add ***unused_interrupts.c*** to the Lab 2 project.

unused_interrupts.c

```
// ************************************// MSP430FR6989 Unused Vectors// ********************************************************
// UNUSED_HWI_ISR()
// The default linker command file created by CCS links all interrupt vectors to their specified address location. This gives you a warning for vectors that are not
// associated with an ISR function. The following function (and pragma's)  handles all interrupt vectors.
// Just make sure you comment out the vector pragmas handled by your own code.
// For example, you will receive a "program will not fit into" error if you do not comment out the WDT vector below.
//  This occurs since the linker tries to place both of the vector addresses into the same memory locations.
// Gregg Chapman, The Ohio State University, February 2018
// ************************************************************************************************************************//
#pragma vector = ADC12_VECTOR                  // ADC
#pragma vector = AES256_VECTOR                 // AES256
#pragma vector = COMP_E_VECTOR                 // Comparator E
#pragma vector = DMA_VECTOR                    // DMA
#pragma vector = ESCAN_IF_VECTOR               // Extended Scan IF
#pragma vector = LCD_C_VECTOR                                  // LCD C
#pragma vector = PORT1_VECTOR                                  // Port 1
#pragma vector = PORT2_VECTOR                  // Port 2
#pragma vector = PORT3_VECTOR                  // Port 3
#pragma vector = PORT4_VECTOR                  // Port 4
#pragma vector = RESET_VECTOR                  // Reset
#pragma vector = RTC_VECTOR                                    // RTC
#pragma vector = SYSNMI_VECTOR                                 // System Non-maskable
// #pragma vector = TIMER0_A0_VECTOR                           // Timer0_A5 CC0
// #pragma vector = TIMER0_A1_VECTOR                           // Timer0_A5 CC1-4, TA
#pragma vector = TIMER0_B0_VECTOR              // Timer0_B3 CC0
#pragma vector = TIMER0_B1_VECTOR              // Timer0_B3 CC1-2, TB
#pragma vector = TIMER1_A0_VECTOR              // Timer1_A3 CC0
#pragma vector = TIMER1_A1_VECTOR              // Timer1_A3 CC1-2, TA1
#pragma vector = TIMER2_A0_VECTOR              // Timer2_A3 CC0
#pragma vector = TIMER2_A1_VECTOR              // Timer2_A3 CC1, TA
#pragma vector = TIMER3_A0_VECTOR              // Timer3_A2 CC0
#pragma vector = TIMER3_A1_VECTOR              // Timer3_A2 CC1, TA
#pragma vector = UNMI_VECTOR                   // User Non-maskable
// #pragma vector = USCI_A0_VECTOR             // USCI A0 Receive/Transmit
#pragma vector = USCI_A1_VECTOR                // USCI A1 Receive/Transmit
#pragma vector = USCI_B0_VECTOR                // USCI B0 Receive/Transmit
#pragma vector = USCI_B1_VECTOR                // USCI B1 Receive/Transmit
#pragma vector = WDT_VECTOR                    // Watchdog Timer

__interrupt void UNUSED_HWI_ISR (void)
{
  __no_operation();
}
/************************************************* END OF CODE ************************************************/
```

ECE 3567 – Lab #2
Additional Files Needed

2. Create a new Source File called ***myGpio.c***.

   File → New → Source File

   a. Add a standard header.

   b. Move the ***Init_GPIO*** function from ***main.c*** to the new file.

   c. Move the ***Init_GPIO*** function prototype to the ***3567.h*** header file.

ECE 3567 – Lab #2
Additional Files Needed

3. Create a new Source File called ***Timer.c***.
    a. Add a function prototype in **3567.h** called
        void ***Init_Timer_A0***(void);
    b. Create the function ***Init_Timer_A0()*** in ***Timer.c.***

ECE 3567 – Lab #2

Watchdog and GPIO Unlock

4. Watchdog disable and GPIO unlock don’t change:

```
WDT_A_hold(__MSP430_BASEADDRESS_WDT_A__);
PMM_unlockLPM5();
```

ECE 3567 – Lab #2
Variables

5. Add the following variables in main.c:

```
volatile unsigned int ISR_Counter;        // Used to count to 10 in order to delay exactly 1 second
volatile unsigned char ISR_Flag = 0;      // Flag to tell main() that a Timer A0 interrupt occurred
volatile unsigned char ISR_Flag_10 = 0; // Flag to tell main() that a Timer A0 interrupt occurred  10 times
```

MSP430FR6989 Project

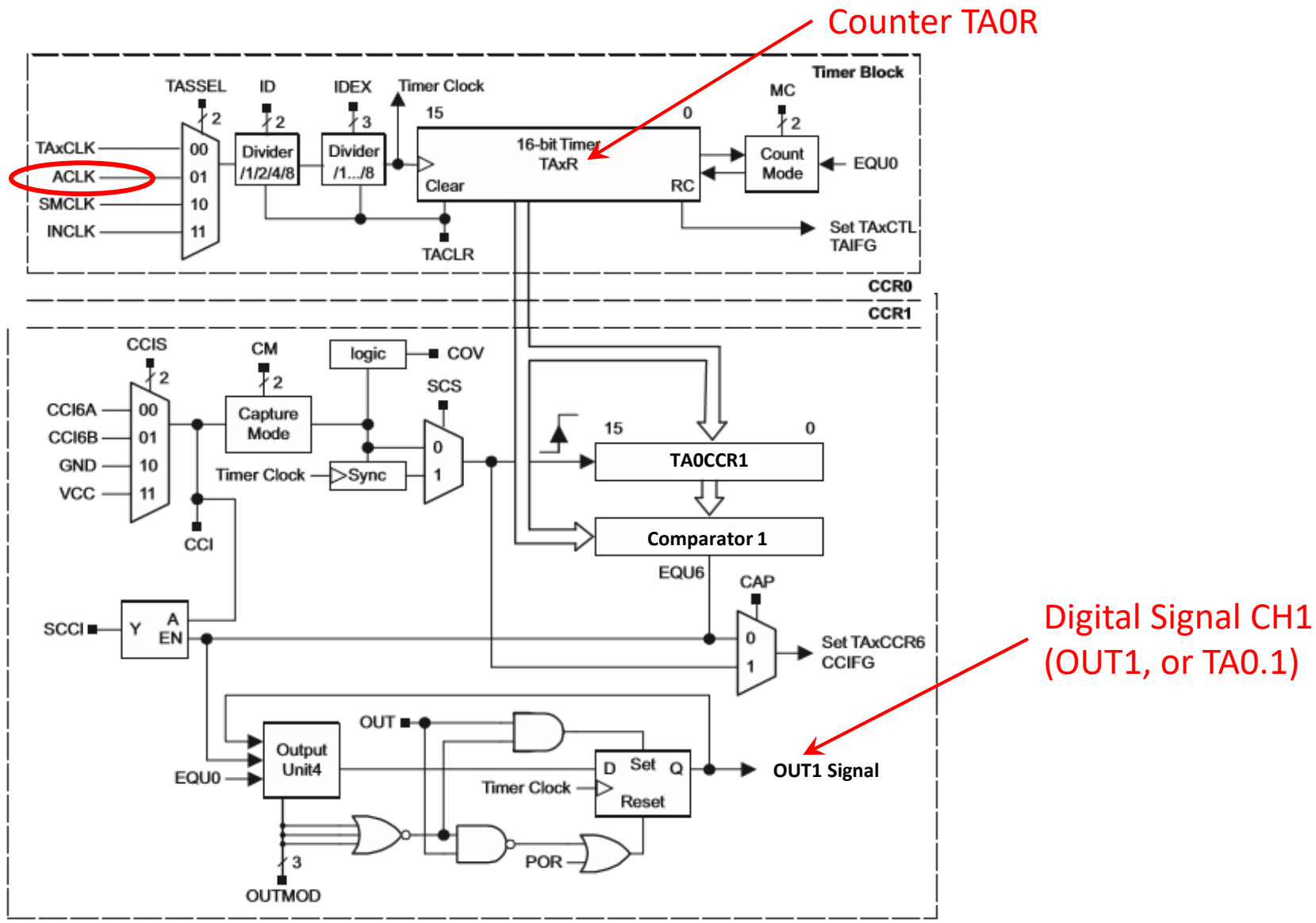
6. EDIT the main function to conditionally reset the ISR_Flag as shown:

```
void main (void)
{
// Initializations go here including Init_GPIO(), Init_Timer_A0(), etc
        while(1)
        {
                if(ISR_Flag==1)        // Timer A0 has occurred.
                {
                        ISR_Flag = 0;
                }
                if(ISR_Flag_10 ==1)            // 1 Sec interval
                {
                        ISR_Flag_10 = 0;
                        // MOVE YOUR LED XORs HERE
                }
        }
}
```

Timer A0 Initialization

Init_Timer_A0( )

Timer A0 Initialization



Timer A0 Initialization

Overview:

You will configure Timer A0 to generate another Interrupt every 100 milliseconds. To do this, you must configure the following registers:

TA0CTL – Timer A0 Control Register
TA0CCTL0 – Compare 0 Control Register
TA0CCTL1 – Compare 1 Control Register
You must also write compare values to the following registers

TA0CCR0 – Compare 0 Register
TA0CCR1 – Compare 1 Register

Timer A0 Initialization
Registers and Field:

- To Set up the TA0 timer for an interrupt every 100 mSec:
  - TA0CTL – Timer A0 Control register
    - TASSEL = ACLK // 32.768 KHz
    - ID = /1 // No Pre-Divide
    - MC = Up Mode // Timer A0 in Up Mode
  - TA0CCTL0 – Comparator 0 Control Register
    - CCIE = enabled (1) // Interrupt enabled for CCR0
  - TA0CCTL1 – Comparator 1 Control Register
    - OUTMOD = Reset/Set (111) // Reset/Set Mode for PWM
  - TA0CCR0 – Comparator 0
    - = 0x???? // 100 mSec period
  - TA0CCR1 – Comparator 1
    - = 0x???? // 50% Duty Cycle

Timer A0 Initialization

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

You should calculate the HEXADECIMAL value for the entire register and write it with one command:

TA0CTL = 0xXXXX;

Timer A0 Initialization – Clock Source

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Bit	Field	Type	Reset	Description
15-10	Reserved	RW	0h	Reserved
9-8	TASSEL	RW	0h	Timer_A clock source select 00b = TAxCLK 01b = ACLK 10b = SMCLK 11b = INCLK

6. Select the correct bits to use the ACLK. These bits will go in the **TASSEL** 2-bit field

Timer A0 Initialization- Clock Divider

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

7-6	ID	RW	0h	Input divider. These bits along with the TAIDEX bits select the divider for the input clock. 00b = /1 01b = /2 10b = /4 11b = /8

7. Select the correct bits to divide the ACLK by 1
These bits will go in the **ID** 2-bit field

Timer A0 Initialization – Timer Mode

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

5-4	MC	RW	0h	Mode control. Setting MC = 00h when Timer_A is not in use conserves power. 00b = Stop mode: Timer is halted 01b = Up mode: Timer counts up to TAxCCR0 10b = Continuous mode: Timer counts up to 0FFFFh 11b = Up/down mode: Timer counts up to TAxCCR0 then down to 0000h

8. Select the correct bits to Put the Timer A0 in UP MODE
These bits will go in the **MC** 2-bit field

Timer A0 Initialization – Other Settings

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

Let Bits 2, 1, and 0 remain 0 for now, and assume Bits 15, 14, 13, 12, 11, and 10, and 3 are 0s

Convert the 16-bit BINARY sequence of numbers that you derived into a 4-character HEXADECIMAL VALUE

9. Set the TA0CTL to the HEXADECIMAL value that you derived Write the value to the ***TA0CTL*** register with a single instruction (see next slide).

DO NOT USE BINARY NUMBERS TO CONFIGURE THE REGISTERS

Timer A0 Initialization- Initialize the Register

Figure 25-16. TAxCTL Register

15	14	13	12	11	10	9	8
Reserved						TASSEL	
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ID		MC		Reserved	TACLR	TAIE	TAIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	w-(0)	rw-(0)	rw-(0)

NOTE: When initializing a register for the first time, it is NOT NECESSARY to use bitwise operations. Just write the derived value to the register)(**TA0CTL = 0xXXXX**)

Timer A0 Initialization – Comparator 0

Figure 25-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

10. Enable the COMPARATOR 0 Interrupt for Timer A0 by SETTING the **CCIE** BIT in the ***TA0CCTL0*** Control Register

NOTE: Interrupts are enabled in the TA0CCTL0 Register, not the TA0CCTL1 Register

Timer A0 Initialization – Comparator 0

Figure 25-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

8	CAP	RW	0h	Capture mode 0b = Compare mode 1b = Capture mode

CAPTURE/ COMPARE MODE

NOTE: Bit 8 is Capture or Compare. It is Compare by DEFAULT. Since this is one of the 3-tier settings for Timer Configuration, it will likely show up on the quiz.

Timer A0 Initialization – Comparator 1

Figure 25-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

Configure COMPARATOR1 Control Register (***TA0CCTL1***)

Timer A0 Initialization – Out Mode 7

Figure 25-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

7-5	OUTMOD	RW	0h	Output mode. Modes 2, 3, 6, and 7 are not useful for TAxCCR0 because EQUx = EQU0. 000b = OUT bit value 001b = Set 010b = Toggle/reset 011b = Set/reset 100b = Toggle 101b = Reset 110b = Toggle/set 111b = Reset/set

11. Select the Bits for RESET/SET in the 3-bit field for **OUTMOD** in the ***TA0CCTL1*** *register.*

Assume all other Bits are 0s

Timer A0 Initialization – Out Mode 7

Figure 25-18. TAxCCTLn Register

15	14	13	12	11	10	9	8
CM		CCIS		SCS	SCCI	Reserved	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r-(0)	r-(0)	rw-(0)
7	6	5	4	3	2	1	0
OUTMOD			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

12. Set the ***TA0CCTL1*** Register Value to the HEXADECIMAL value that you derived in Step 11. Write the value to the ***TA0CCTL1*** register with a single instruction.

DO NOT USE BINARY NUMBERS TO CONFIGURE THE REGISTERS

Timer A0 Initialization – Period

25.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 25-19. TAxCCRn Register

15	14	13	12	11	10	9	8
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 25-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.

13. Set the ***TA0CCR0*** Comparator Register for a 10 Hz frequency (100 mSec. period) using the ACLK.

Timer A0 Initialization – Duty Cycle

25.3.4 TAxCCRn Register

Timer_A Capture/Compare n Register

Figure 25-19. TAxCCRn Register

15	14	13	12	11	10	9	8
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
TAxCCRn							
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

Table 25-7. TAxCCRn Register Description

Bit	Field	Type	Reset	Description
15-0	TAxCCR0	RW	0h	Compare mode: TAxCCRn holds the data for the comparison to the timer value in the Timer_A Register, TAR. Capture mode: The Timer_A Register, TAR, is copied into the TAxCCRn register when a capture is performed.

14. Set the ***TA0CCR1*** Comparator Register for a 50% duty cycle

ECE 3567 – Lab #2

Checkpoint #2: Ask a Lab Monitor to verify that your Init_Timer_A0() function is correct

ECE 3567 – Lab #2

enable interrupts

15. Back in main(), use the TI macro to enable all configured interrupts simultaneously at :

```
__enable_interrupt();
```

NOTE: This goes at the end of the initialization section, before entering the while(1) loop .

Timer A0
Interrupt Service Routine

Interrupt Service Routines

- Require a #pragma vector= NAME_OF_VECTOR

- There is USUALLY an unused_interrupt file with #pragmas for all possible interrupts. The vector you wish to use must be commented out in the unused_interrupt source file.

- Any code for an Interrupt Service Routine must also be preceded by the reserved for implementation name of :
  __interrupt

Timer A0 ISR

16. Back in main.c, ***AFTER*** the main() function, Create the Timer A0 Interrupt Service Routine:

USE THE FOLLOWING FORMAT:

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0(void)
{


return;
}
```

Timer A0 ISR

Inside the Timer_A0 ISR:

17. Set the ISR_Flag = 1;

18. Increment the ISR_Counter++;

Timer A0 ISR

Inside the Timer_A0 ISR, if the ISR_Counter is greater than or equal to 10:

19. SET the ISR_Counter_10
20. Reset the ISR_Counter to 0

ECE 3567 – Lab #1

Checkpoint #3: Ask a Lab Monitor to verify that your TA0 Interrupt Service Routine is correct.

MSP430FR6989 Project
Timer A0 ISR

21. In ***unused_interrupts.c ,*** comment out the if it is not already commented out.

```
#pragma vector = TIMER0_A0_VECTOR
```

MSP430FR6989 Project
Lab 2

Your Lab 2 code should now compile and run.

What is generating the interrupt in the Timer A0 module?
You chose it in a configuration register.

ECE 3567 – Lab #1

Checkpoint #4: Demonstrate that the Lab #2 project is operating correctly.

1. One LED should flash at a time.
2. The GREEN LED should be the default after initialization
3. The LEDs should alternate, RED .. GREEN .. RED at EXACTLY 1 Hz.

ECE 3567 – Lab #2

Pulse Width Modulation

Add code to the Timer A0 Interrupt Service Routine to

22. INCREMENT the Duty Cycle comparator (TA0CCR1) by 10 every interrupt.

23. If the Duty Cycle is >= TAOCCR0, reset it to 0x0010

24. Configure P1.6 to output TA0.1
NOTE: You will need to change the pin function to TERTIARY, by programming bit 6 in both P1SEL0 and P1SEL1 to 1. Don't forget to make P1.6 an OUTPUT.

25. Connect CHANNEL 1 of the oscilloscope to P1.6 on the Launchpad header and observe the Pulse Width Modulation.

ECE 3567 – Lab #1

Checkpoint #5: Demonstrate the PWM signal on your oscilloscope to one of the Lab Monitors

ECE 3567 – Lab #2

Pulse Width Modulation

26. Restore the 50% Duty Cycle comparator (TA0CCR1) value.

End of Laboratory #2