Hacettepe University Department of Electrical and Electronics Engineering

ELE 417-EMBEDDED SYSTEM DESIGN

Experiment 3 Report – Interrupt Usage on MSP430

Student Name: Baki ALMACI **Due Date**: 12.23.2020

Student ID: 21627983

Questions and Answers

1. **What are the modes for MSP430 Timer A? Give at least one example of usage for each mode.**

Timer A supports 4 modes of operation which are:

i. **Stop Mode**: In this mode Timer is halted.

```
#include <msp430.h>
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watch dog timer
    PM5CTL0 &= ~LOCKLPM5; //previous port settings are activated.
    P1DIR |= BIT0; // P1.0 is configured as output
    P1OUT = ~BIT0; // Led on P1.0 is turned off
    TA1CCR0 = 40000;
    TA1CTL = TASSEL_1 | MC_1 | TACLR;
    while (1)
    {
        if (TA1CTL & TAIFG == TAIFG)
        {
            P1OUT ^= BIT0;
            TA1CTL &= ~TAIFG;
            TA1CTL =  MC_0 | ~MC_1; // Change mode to the MC_0
        }
    }
}
```

Given code above is holds turn on the LED when timer flag has been triggered by stop mode of timer. So, in that case LED will be turn on forever.

ii. **Up Mode:** Timer repeatedly counts from Zero to value stored in Capture/Compare Register 0 (TACCR0).

```
#include <msp430.h>
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watch dog timer
    PM5CTL0 &= ~LOCKLPM5; //previous port settings are activated.
    P1DIR |= BIT0; // P1.0 is configured as output
    P1OUT = ~BIT0; // Led on P1.0 is turned off
    TA1CCR0 = 40000;
    TA1CTL = TASSEL_1 | MC_1 | TACLR;
    while (1)
    {
        if (TA1CTL & TAIFG == TAIFG)
        {
            P1OUT ^= BIT0;
            TA1CTL &= ~TAIFG;
        }
    }
}
```

Given code above is a LED blinking example using timer mode “Up to CCR0”. When the timer reaches to the value of CCR0(CCR0=40000 in that case) the flag has been triggered.

iii. Continuous: Timer repeatedly counts from Zero to 0xFFFF, which is maximum value for 16-bit TAR.

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                       // Stop WDT
    P1DIR |= BIT0;                                  // P1.0 output
    PM5CTL0 &= ~LOCKLPM5; // previously configured port settings
    TA0CCTL0 |= CCIE;                               // TACCR0 interrupt enabled
    TA0CTL |= TASSEL__SMCLK | MC__CONTINOUS;        // SMCLK, continuous mode
    __bis_SR_register(LPM0_bits | GIE);             // Enter LPM3 w/ interrupts
    __no_operation();                               // For debugger
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
    P1OUT ^= BIT0;
}
```

Continuous mode is increments the TA0R until it reaches 65535 (max value). This register will be set to zero after each reach. Also, each reach toggles the LED state.

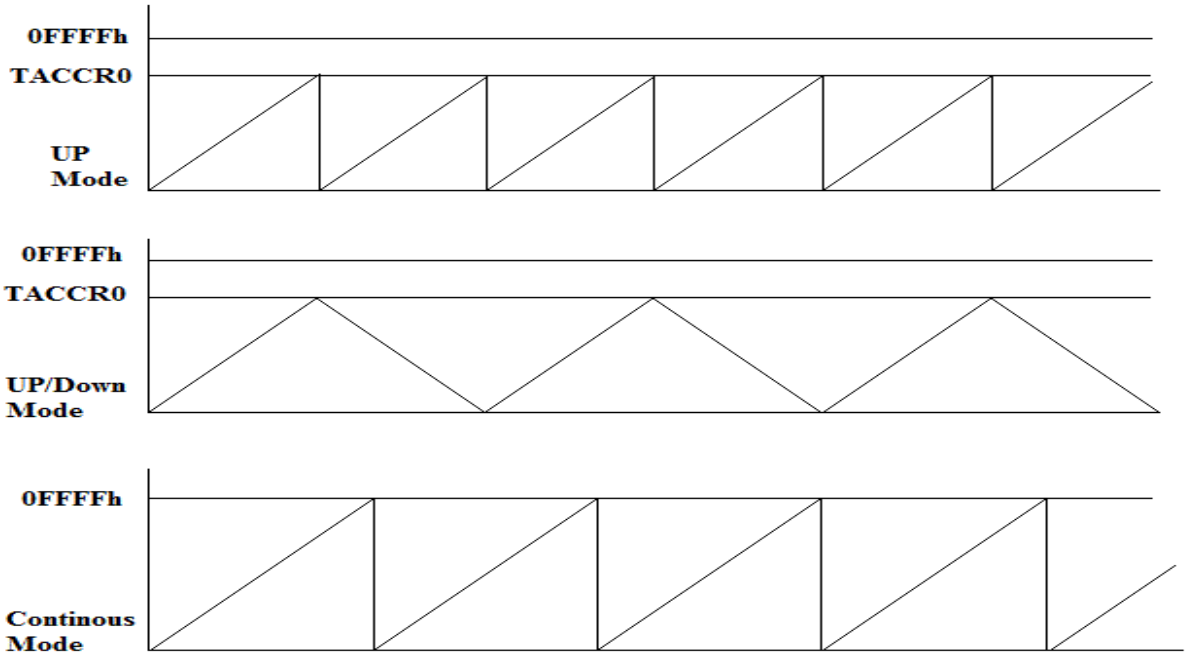
iv. **Up/Down Mode:** Timer repeatedly counts from Zero up to the value in TACCR0 and back down to zero.

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                       // Stop WDT
    P1DIR |= BIT0;                                  // P1.0 output
    PM5CTL0 &= ~LOCKLPM5; // previously configured port settings
    TA0CCTL0 |= CCIE;                               // TACCR0 interrupt enabled
    TA0CCR0 = 60000;
    TA0CTL |= TASSEL__SMCLK | MC__UPDOWN;           // SMCLK, up-down mode
    __bis_SR_register(LPM0_bits | GIE);             // Enter LPM3 w/ interrupts
    __no_operation();                               // For debugger
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
    P1OUT ^= BIT0;
}
```

In up-down mode, timer turn around specified value which is 60000. Timer will turn back instead of being zero in that case. Each turn will toggles the LED state.

Working principles of these modes given below as graphical.



2. **What is an interrupt and interrupt service routine(ISR)? What is the meaning of pragma macro TIMERA0 VECTOR or PORT1 VECTOR?**

An interrupt service routine (ISR) is a software routine that hardware invokes in response to an interrupt. ISR examines an interrupt and determines how to handle it executes the handling, and then returns a logical interrupt value. If no further handling is required, the ISR notifies the kernel with a return value. An ISR must perform very quickly to avoid slowing down the operation of the device and the operation of all lower-priority ISRs.

The vector pragma sets the interrupt service routine vector for the following function definition, if that function is an interrupt function.
An interrupt vector is the memory location of an interrupt handler, which prioritizes interrupts and saves them in a queue if more than one interrupt is waiting to be handled.

3. **What does the interrupt edge select register do in the concept of port interrupts?**

Interrupt Edge Select register controls which edge an interrupt happens on. We use the term edge to mean the transition from when a signal changes from low-to-high or high-to-low. When a signal goes from low to high, we call that a rising edge, since the signal value "rises up" to a higher level. When a signal goes from high to low, we call that a falling edge, since the signal value "falls down" to a lower level.

4. **Write a code in C that turns off/on an LED by using port interrupts. LED should be on when the button is pressed down, it should turn off when the button is released. Usage of any kind of delay function/loop is FORBIDDEN.**

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                       // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;
    P1DIR |= BIT0;                                  // P1.0 output
    P1OUT &= ~BIT0; // turn off the LED

    P1REN |= BIT2; // configure P1.2 as input Switch connection
    P1IE |= BIT2; // enable P1.2 interrupt
    P1IES |= BIT2; // configure the edge of the interrupt
    P1IFG &= ~BIT2; // clear the interrupt flag
    __bis_SR_register(GIE);
    while (1)
    {
    }
}

#pragma vector=PORT1_VECTOR
__interrupt void Port_1_IntHandler(void)
{
    P1OUT ^= BIT0; // toggle LED
    P1IES ^= BIT2; // toggle the edge of the interrupt
    P1IFG &= ~BIT2; // clear interrupt flag
}
```

5. **Write a code in C with following flow of operation(Use interrupt service routine for handling timer interrupts): · Configure Timer A to work in up mode · Configure LEDs as outputs on necessary ports · Configure Timer A interval to generate an initial interrupt by setting timer compare value as small as possible. · Turn on and off the LEDs with such intervals that blinking interval gets multiplied by 2 every time until your timer compare value is at its maximum. After that it should start from the initial value you set earlier.**

   I've divided my clock source and timer clock respectively by 32 and 8 to observe blinking easily. Also, I multiplied CCR0 by 2 with bit shifting. I've checked out CCR0 value when the interrupt triggered using ternary operator.

```
#include <msp430.h>
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;

    CSCTL1 |= DCORSEL_1; // Set DCOCLK = 1MHz
    CSCTL2 = FLLD__32;  // FLLD = 32, DCODIV = DCO/32 MHz

    P1DIR |= BIT0;  // P1.0 output
    P1OUT &= ~BIT0; // turn off the LED

    TA0CCTL0 |= CCIE; // TACCR0 interrupt enabled
    TA0CCR0 = 1000;
    TA0CTL = TASSEL__SMCLK | MC__UP | ID__8; // Divide by 8
    __bis_SR_register(GIE); // Interrupts are enabled
    while (1)
    {

    }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
    P1OUT ^= BIT0; // toggle LED
    TA0CCR0 = TA0CCR0 >= 64000 ? 1000 : TA0CCR0 << 1;
}
```

6. **Write a code in C that blinks one of the LEDs on your launchpad. Use interrupt service routine for handling timer interrupts. Blinking intervals should use Timer A operation similar to the previous question. Usage of any kind of delay function/loop is FORBIDDEN. Blinking pattern should alternate between a short blink and long blink with each cycle, i.e. LED is on → Short delay → LED is off → Short delay → LED is on → Long delay → LED is off → Long delay → LED is on → Short delay. . .**

```
#include <msp430.h>
#define short_delay 1000
#define long_delay  10000
volatile unsigned int counter = 0;
unsigned const int delays[] = { short_delay, short_delay, long_delay };
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // Stop WDT
    PM5CTL0 &= ~LOCKLPM5;

    CSCTL1 |= DCORSEL_1; // Set DCOCLK = 1MHz
    CSCTL2 = FLLD__32;  // FLLD = 32, DCODIV = DCO/32 MHz

    P1DIR |= BIT0;  // P1.0 output
    P1OUT &= BIT0; // turn on the LED

    TA0CCTL0 |= CCIE; // TACCR0 interrupt enabled
    TA0CCR0 = delays[counter];
    TA0CTL = TASSEL__SMCLK | MC__UP | ID__8; // Divide by 8
    __bis_SR_register(GIE); // Interrupts are enabled
    while (1)
    {

    }
}

#pragma vector = TIMER0_A0_VECTOR
__interrupt void Timer_A(void)
{
    P1OUT ^= BIT0; // toggle LED
    counter = counter > 2 ? 0 : ++counter;
    TA0CCR0 = delays[counter];
}
```