

**Embedded System design**

**EGR 226**

**Registers\_Summary\_and\_vectors**

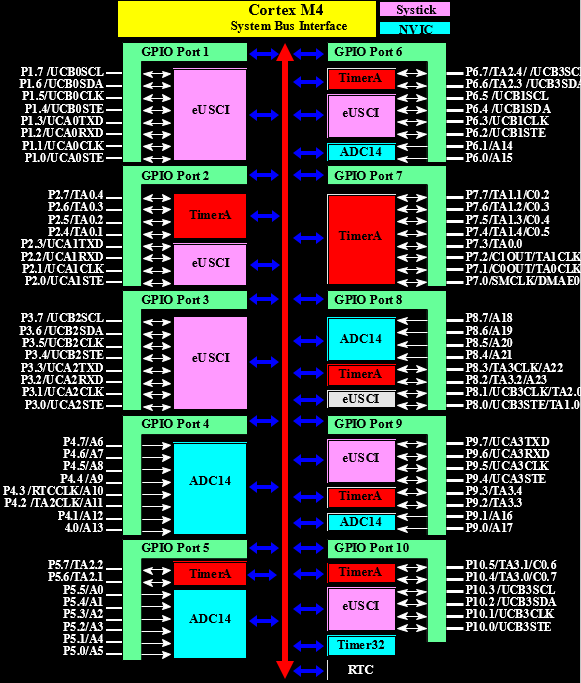
**MSP432**

**DR. Nabeeh Kandalaft**

**February, 2018**

**Revision I**

**REGISTER DESCRIPTION**



**Interrupts vectors**

**&**

**Registers**

***Interrupt vectors***

***startup\_msp432p401r\_ccs.c***

**Reset\_Handler,// the reset handler**

**NMI\_Handler,// The NMI handler**

**HardFault\_Handler,// the hard fault handler**

**MemManage\_Handler,// the MPU fault handler**

**BusFault\_Handler,// the bus fault handler**

**UsageFault\_Handler,// the usage fault handler**

**0,// Reserved**

**0 , // Reserved**

**0,// Reserved**

**0,// Reserved**

**SVC\_Handler,// SVCall handler**

**DebugMon\_Handler,// Debug monitor handler**

**0,// Reserved**

**PendSV\_Handler,// The PendSV handler**

**SysTick\_Handler,// The SysTick handler**

**PSS\_IRQHandler,// PSS Interrupt**

**CS\_IRQHandler,// CS Interrupt**

**PCM\_IRQHandler,// PCM Interrupt**

**WDT\_A\_IRQHandler,// WDT\_A Interrupt**

**FPU\_IRQHandler,// FPU Interrupt**

**FLCTL\_IRQHandler,// Flash Controller Interrupt**

**COMP\_E0\_IRQHandler,// COMP\_E0 Interrupt**

**COMP\_E1\_IRQHandler,// COMP\_E1 Interrupt**

**TA0\_0\_IRQHandler,// TA0\_0 Interrupt**

**TA0\_N\_IRQHandler,// TA0\_N Interrupt**

**TA1\_0\_IRQHandler,// TA1\_0 Interrupt**

**TA1\_N\_IRQHandler,// TA1\_N Interrupt**

**TA2\_0\_IRQHandler,// TA2\_0 Interrupt**

**TA2\_N\_IRQHandler,// TA2\_N Interrupt**

**TA3\_0\_IRQHandler,// TA3\_0 Interrupt**

**TA3\_N\_IRQHandler,// TA3\_N Interrupt**

**EUSCIA0\_IRQHandler,// EUSCIA0 Interrupt**

**EUSCIA1\_IRQHandler,// EUSCIA1 Interrupt**

**EUSCIA2\_IRQHandler,// EUSCIA2 Interrupt**

**EUSCIA3\_IRQHandler,// EUSCIA3 Interrupt**

**EUSCIB0\_IRQHandler,// EUSCIB0 Interrupt**

**EUSCIB1\_IRQHandler,// EUSCIB1 Interrupt**

**EUSCIB2\_IRQHandler,// EUSCIB2 Interrupt**

**EUSCIB3\_IRQHandler,// EUSCIB3 Interrupt**

**ADC14\_IRQHandler,// ADC14 Interrupt**

**T32\_INT1\_IRQHandler,// T32\_INT1 Interrupt**

**T32\_INT2\_IRQHandler,// T32\_INT2 Interrupt**

**T32\_INTC\_IRQHandler,// T32\_INTC Interrupt**

**AES256\_IRQHandler,// AES256 Interrupt**

**RTC\_C\_IRQHandler,// RTC\_C Interrupt**

**DMA\_ERR\_IRQHandler, // DMA\_ERR Interrupt**

**DMA\_INT3\_IRQHandler,// DMA\_INT3 Interrupt**

**DMA\_INT2\_IRQHandler,// DMA\_INT2 Interrupt**

**DMA\_INT1\_IRQHandler,// DMA\_INT1 Interrupt**

**DMA\_INT0\_IRQHandler,// DMA\_INT0 Interrupt**

**PORT1\_IRQHandler,// Port1 Interrupt**

**PORT2\_IRQHandler,// Port2 Interrupt**

**PORT3\_IRQHandler,// Port3 Interrupt**

**PORT4\_IRQHandler,// Port4 Interrupt**

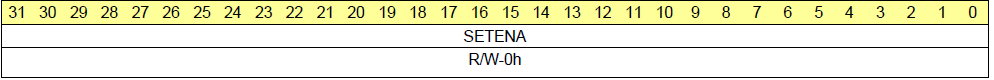
**PORT5\_IRQHandler,// Port5 Interrupt**

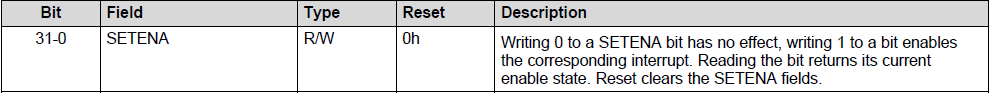
**PORT6\_IRQHandler// Port6 Interrupt**

**ISER0 Register IRQ 0 to 31** Set Enable Register.

Use the Interrupt Set-Enable Registers to enable interrupts and determine which interrupts are currently enabled.

**Figure 2-20. ISER0 Register**

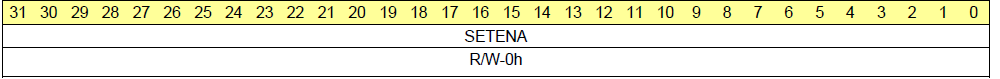


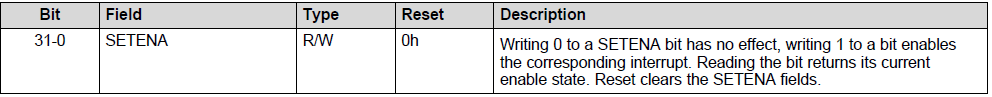


**ISER1 Register IRQ 32 to 63** Set Enable Register.

Use the Interrupt Set-Enable Registers to enable interrupts and determine which interrupts are currently enabled.

**Figure 2-21. ISER1 Register**





**PSS\_IRQn = 0, // 16 PSS Interrupt**

**CS\_IRQn = 1, // 17 CS Interrupt**

**PCM\_IRQn = 2, // 18 PCM Interrupt**

**WDT\_A\_IRQn = 3, // 19 WDT\_A Interrupt**

**FPU\_IRQn = 4, // 20 FPU Interrupt**

**FLCTL\_IRQn = 5, // 21 Flash Controller Interrupt**

**COMP\_E0\_IRQn = 6, // 22 COMP\_E0 Interrupt**

To enable the interrupt vectors from 0-31 **use ISER0 register**

**NVIC->ISER[0] = 1 << ( (ADC14\_IRQn) & 31);**

**NVIC->ISER[0] = 1 << ( (TA0\_0\_IRQn) & 31);**

**Or**

**NVIC\_EnableIRQ (TA0\_N\_IRQn);**

**NVIC\_EnableIRQ (ADC14\_IRQn);**

**COMP\_E1\_IRQn = 7, // 23 COMP\_E1 Interrupt**

**TA0\_0\_IRQn = 8, // 24 TA0\_0 Interrupt**

**TA0\_N\_IRQn = 9, // 25 TA0\_N Interrupt**

**TA1\_0\_IRQn = 10, // 26 TA1\_0 Interrupt**

**TA1\_N\_IRQn = 11, // 27 TA1\_N Interrupt**

**TA2\_0\_IRQn = 12, // 28 TA2\_0 Interrupt**

**TA2\_N\_IRQn = 13, // 29 TA2\_N Interrupt**

**TA3\_0\_IRQn = 14, // 30 TA3\_0 Interrupt**

**TA3\_N\_IRQn = 15, // 31 TA3\_N Interrupt**

**EUSCIA0\_IRQn = 16, // 32 EUSCIA0 Interrupt**

**EUSCIA1\_IRQn = 17, // 33 EUSCIA1 Interrupt**

**EUSCIA2\_IRQn = 18, // 34 EUSCIA2 Interrupt**

**EUSCIA3\_IRQn = 19, // 35 EUSCIA3 Interrupt**

To enable the interrupt vectors from 32-40 **use ISER1 register**

**NVIC->ISER[1] = 1 << ( (PORT1\_IRQn) & 31);**

**Or**

**NVIC\_EnableIRQ (PORT1\_IRQn);**

**EUSCIB0\_IRQn = 20, // 36 EUSCIB0 Interrupt**

**EUSCIB1\_IRQn = 21, // 37 EUSCIB1 Interrupt**

**EUSCIB2\_IRQn = 22, // 38 EUSCIB2 Interrupt**

**EUSCIB3\_IRQn = 23, // 39 EUSCIB3 Interrupt**

**ADC14\_IRQn = 24, // 40 ADC14 Interrupt**

**T32\_INT1\_IRQn = 25, // 41 T32\_INT1 Interrupt**

**T32\_INT2\_IRQn = 26, // 42 T32\_INT2 Interrupt**

**T32\_INTC\_IRQn = 27, // 43 T32\_INTC Interrupt**

**// Enable all interrupts for MSP432 (that are turned on)**

**// declare before the while loop**

**\_\_enable\_interrupt ( );**

**AES256\_IRQn = 28, // 44 AES256 Interrupt**

**RTC\_C\_IRQn = 29, // 45 RTC\_C Interrupt**

**DMA\_ERR\_IRQn = 30, // 46 DMA\_ERR Interrupt**

**DMA\_INT3\_IRQn = 31, // 47 DMA\_INT3 Interrupt**

**DMA\_INT2\_IRQn = 32, // 48 DMA\_INT2 Interrupt**

**DMA\_INT1\_IRQn = 33, // 49 DMA\_INT1 Interrupt**

**DMA\_INT0\_IRQn = 34, // 50 DMA\_INT0 Interrupt**

**PORT1\_IRQn = 35, // 51 Port1 Interrupt**

**PORT2\_IRQn = 36, // 52 Port2 Interrupt**

**PORT3\_IRQn = 37, // 53 Port3 Interrupt**

**PORT4\_IRQn = 38, // 54 Port4 Interrupt**

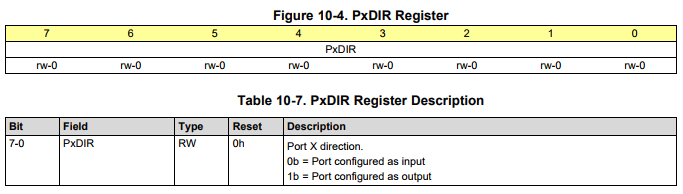
**PORT5\_IRQn = 39, // 55 Port5 Interrupt**

**PORT6\_IRQn = 40 // 56 Port6 Interrupt**

**MSP432**

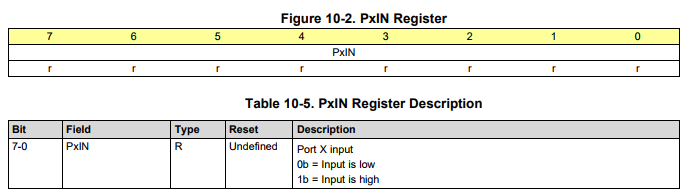
**Registers**

**Pin Configuration Registers**

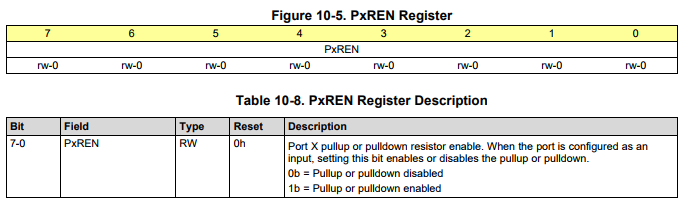


**Set PORT3 pin 1 to output P3->DIR |= BIT1;**

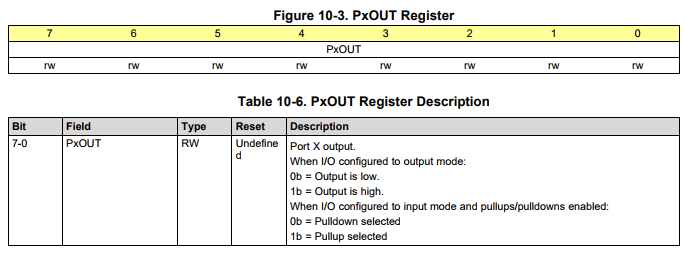
**Set PORT3 pin 1 to input P3->DIR &=~ BIT1;**



**To check a status of pin1 on PORT3 if (P3->IN & BIT1)**



**Use the internal pull-up / pulldown resistor on pin1 PORT2 P3->REN |= BIT1;**



**Set pin1 PORT 3 to high (assume the pin is an output) P3->OUT |= BIT1;**

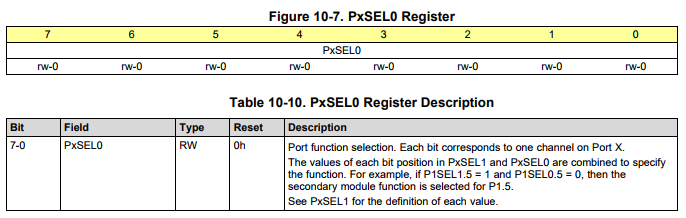
**Set pin1 PORT3 to pull-up (pin1 is an input with enable internal resistor) P3->OUT |= BIT1;**

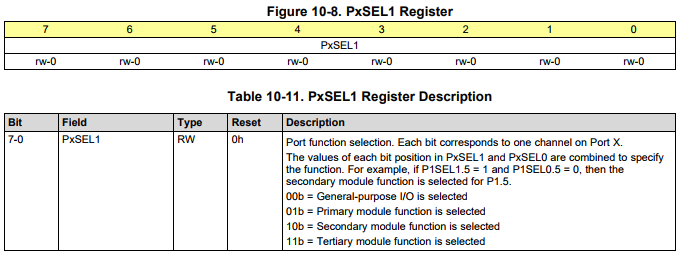
**Set pin1 PORT3 to pulldown (pin1 is an input with enable internal resistor) P3->OUT &=~ BIT1;**

***From Table 4.1 in Valvano’s textbook Introduction to the MSP432 Microcontroller***

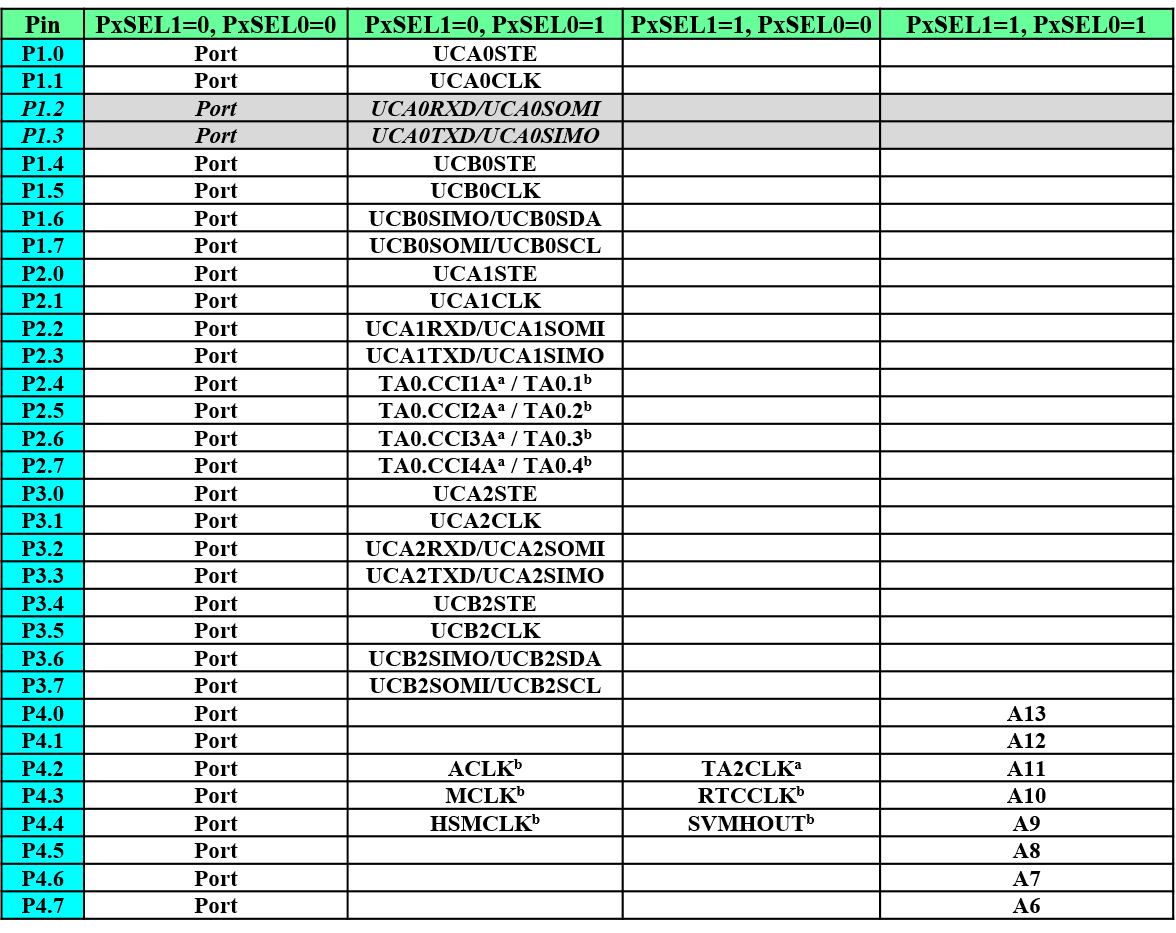
**PxSEL0 PxSEL1 Registers**

**Port X Function Selection Register 0 (X = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 or J)**





**Partial Table of Alternate Pin Functions**

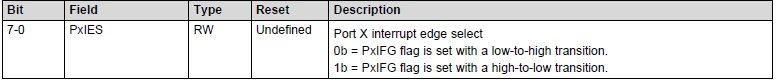


**PxIES Register**

**Port X Interrupt Edge Select Register (X = 1, 2, 3, 4, 5, 6, 7, 8, 9 or 10)**

**Figure 10-10. PxIES Register**





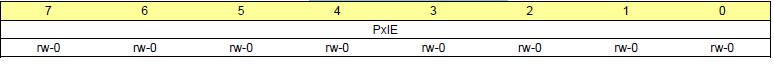
**Example**

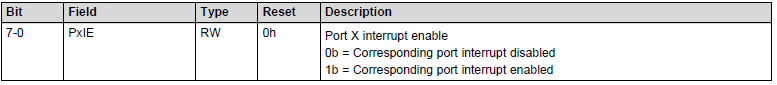
**P1->IES |= BIT1 |BIT4; //Set 1.1, 1.4 pins interrupt to trigger when it goes from high to low**

**PxIE Register**

**Port X *Interrupt Enable* Register (X = 1, 2, 3, 4, 5, 6, 7, 8, 9 or 10)**

**Figure 10-11. PxIE Register**





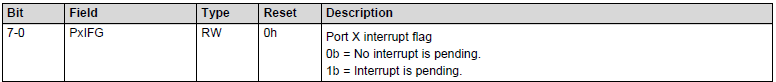
**Example**

**P1 -> IE |= BIT1 |BIT4; // Set interrupt on for p1.1 and 1.4**

**PxIFG Register**

**Port X Interrupt Flag Register (X = 1, 2, 3, 4, 5, 6, 7, 8, 9 or 10)**





**Example interrupt on switch P1.1 toggle the LED on PORT1 pin 0.**

**If (P1->IFG & BIT1)**

**P1->OUT ^= BIT0; // Toggle the LED**

**P1->IFG &= ~BIT1; // always clear the flag before you exit the interrupt ISR**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**P1.1 is configured as an input. Pressing the button connected to P1.1 service an interrupt service routine ISR results in toggling an LED on PORT3 pin5 in device waking up and servicing the Port1 ISR.**

**MCLK = SMCLK = 3MHz**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**#include "msp.h"**

**int main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Hold the watchdog**

**Port1\_init ( );**

**Port3\_init ( );**

**NVIC->ISER[1] = 1 << ((PORT1\_IRQn) & 31); // Enable Port 1 interrupt on the NVIC**

**\_\_enable\_interrupt ( ); // Enable global interrupt**

**while ( 1 )**

**{**

**; // Hardware is doing all the work, so nothing more to do in this while loop.**

**}**

**}**

**void Port1\_init (void);**

**{**

**P1->SEL0 &=~ BIT1;**

**P1->SEL1 &=~ BIT1;**

**P1->DIR &=~ BIT1;**

**P1->REN = BIT1; // Enable pull-up resistor (P1.1 output high)**

**P1->OUT = BIT1;**

**P1->IES = BIT1; //Set pin interrupt to trigger when it goes from high to low**

**P1->IE = BIT1; // Enable interrupt for P1.1**

**P1->IFG = 0; // Clear all P1 interrupt flags**

**}**

**void Port3\_init (void)**

**{**

**P3->SEL0 &=~ BIT5;**

**P3->SEL1 &=~ BIT5;**

**P3->DIR |= BIT5;**

**}**

**void PORT1\_IRQHandler(void) // Port1 ISR**

**{**

**If ( P1->IFG & BIT1 ) // If P1.1 had an interrupt (going from high to low**

**P3->OUT ^= BIT5; // Toggling the output on the LED**

**P1->IFG &= ~BIT1; // Reset the interrupt flag**

**}**

**Example:**

**Sets up P1.1 to toggle Green LED on falling edge PORT3 pin0 // with interrupts**

**Sets up P1.4 to toggle blue LED on falling edge PORT3 pin1 // with interrupts**

**#include "msp.h"**

**void SetupLEDs (void);**

**SetupPort1Interrupts (void);**

**void main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**SetupLEDs ( ); //Sets up the LEDs as outputs**

**SetupPort1Interrupts ( ); //Initializes the buttons that interrupt the program**

**NVIC->ISER[1] = 1 << ((PORT1\_IRQn) & 31); // Enable Port 1 interrupt on the NVIC**

**\_\_enable\_interrupt ( ); //Enable all interrupts that are turned on**

**while (1)**

**{**

**; // Hardware is doing all the work, so nothing more to do in this while loop**

**}**

**}**

**void PORT1\_IRQHandler(void)**

**{**

**int status = P1->IFG; //Record all flags of all interrupts on this port**

**If (status & BIT1) //If P1.1 had an interrupt (comparing the status with the BIT)**

**P2->OUT ^= BIT1; //Turn Green On**

**If (status & BIT4) //If P1.4 had an interrupt (comparing the status with the BIT)**

**P2->OUT ^= BIT2; //Turn Blue On**

**P1->IFG = 0; //Clear all flags**

**}**

**void SetupLEDs(void)**

**{**

**P3->SEL0 &=~ BIT0; // Setup the LED2 on the Launchpad as GPIO, Output, Off**

**P3->SEL1 &=~ BIT0;**

**P3->DIR |= BIT0;**

**P3->OUT &=~ BIT0;**

**P3->SEL0 &=~ BIT1; // Setup the LED2 on the Launchpad as GPIO, Output, Off**

**P3->SEL1 &=~ BIT1;**

**P3->DIR |= BIT1;**

**P3->OUT &=~ BIT1;**

**}**

**void SetupPort1Interrupts ( )**

**{**

**P1->SEL0 &=~ BIT1; // Setup the P1.1 on the Launchpad as Input, Pull Up Resistor**

**P1->SEL1 &=~ BIT1;**

**P1->DIR &=~ BIT1;**

**P1->REN |= BIT1;**

**P1->OUT |= BIT1;**

**P1->IES |= BIT1; //Set pin interrupt to trigger from high to low (starts high due to pull up resistor)**

**P1->IE |= BIT1; //Set interrupt on for P1.4**

**P1->SEL0 &=~ BIT4; // Setup the P1.4 on the Launchpad as Input, Pull Up Resistor**

**P1->SEL1 &=~ BIT4;**

**P1->DIR &=~ BIT4;**

**P1->REN |= BIT4;**

**P1->OUT |= BIT4;**

**P1->IES |= BIT4; //Set pin interrupt to trigger from high to low (starts high due to pull up resistor)**

**P1->IE |= BIT4; //Set interrupt on for P1.1**

**P1->IFG = 0; //Clear all interrupt flags**

**}**

**SysTick Timer**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Offset** | **Acronym** | **Register name** | **Type** | **Reset** | **Section** |
| **10h** | **STCSR** | **SysTick Control and Status Register** | **r/w** | **00000004h** | **2.4.4.1** |
| **14h** | **STRVR** | **SysTick Reload Value Register** | **r/w** | **Undefined** | **2.4.4.2** |
| **18h** | **STCVR** | **SysTick Current Value Register** | **r/w** |  | **2.4.4.3** |
| **1Ch** | **STCR** | **SysTick Calibration Value** | **r/w** |  | **2.4.4.4** |

**SysTick Registers**

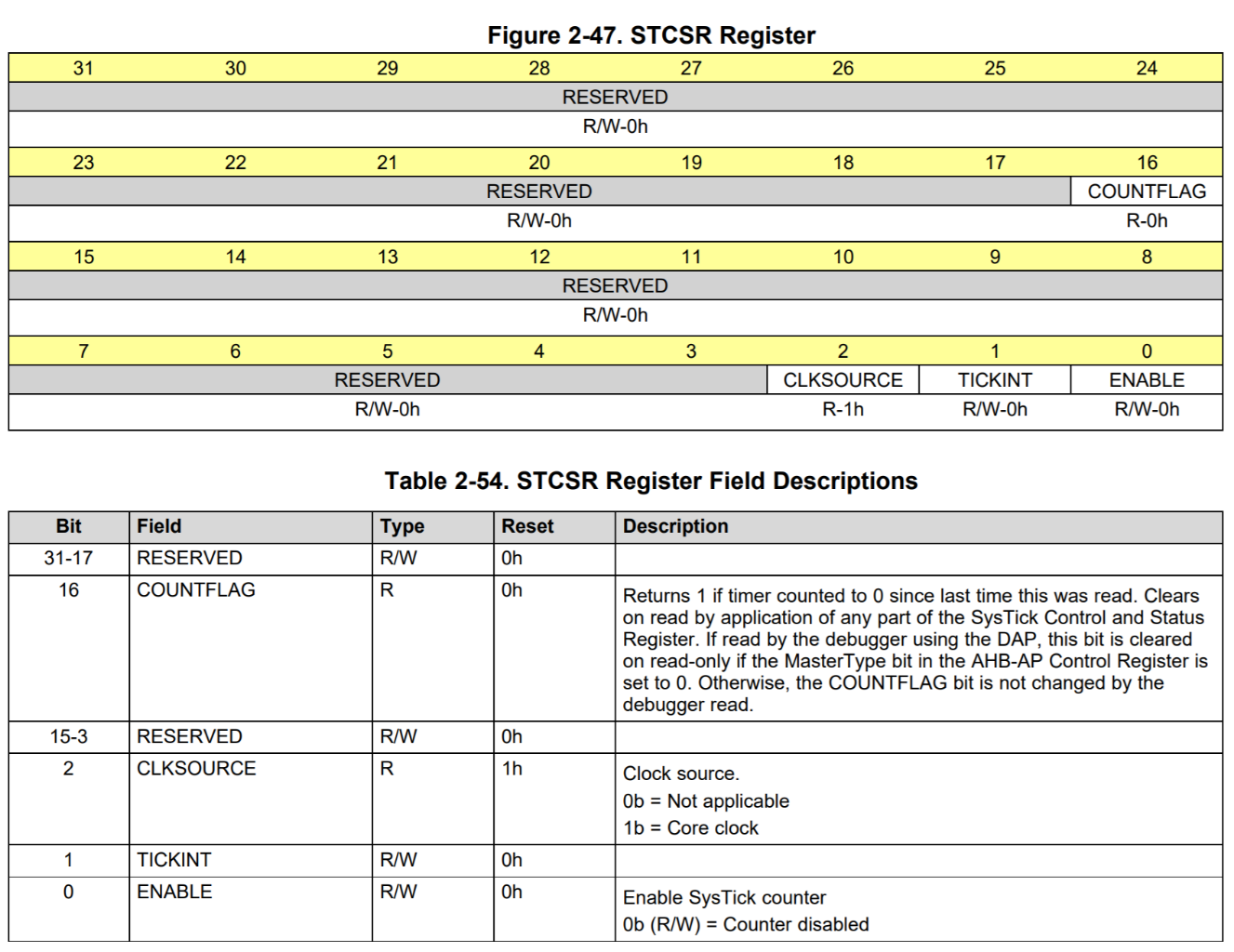
**SysTick -> CTRL (to access the Control Status Register STCSR)**

**SysTick -> LOAD (to access the Reload Value Register STRVR)**

**SysTick -> VAL (to access the Current Value Register STCVR**

**STCSR Register [SysTick -> CTRL]**

**SysTick Control and Status Register. Use the SysTick Control and Status Register to enable the SysTick features.**

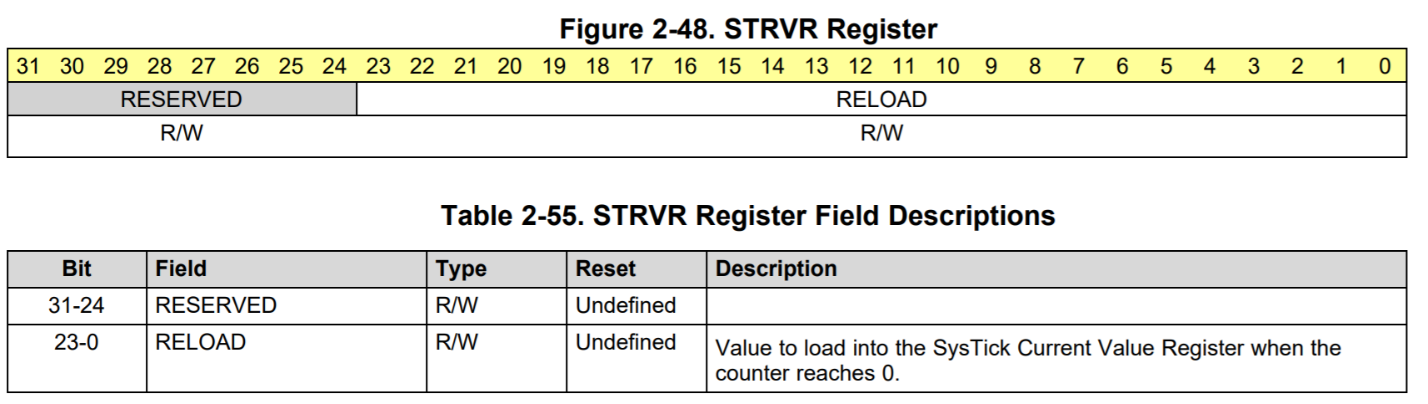


***SysTick -> CTRL = 0x00000005; // enable SysTick, 3MHz, No Interrupts***

***SysTick -> CTRL |= BIT0 | BIT2;***

**STRVR Register [SysTick -> LOAD]**

**Specify the start value to load into the current value register when the counter reaches 0. It can be any value between 1 and 0x00FFFFFF. A start value of 0 is possible, but has no effect because the SysTick interrupt and COUNTFLAG are activated when counting from 1 to 0.**

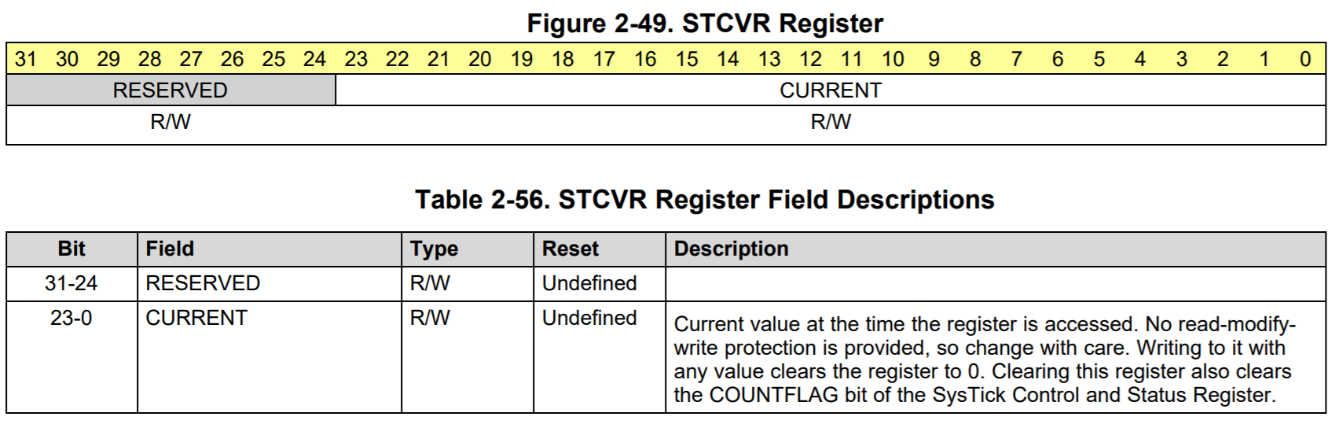


***SysTick -> LOAD = 0x00FFFFFF; // maximum reload value***

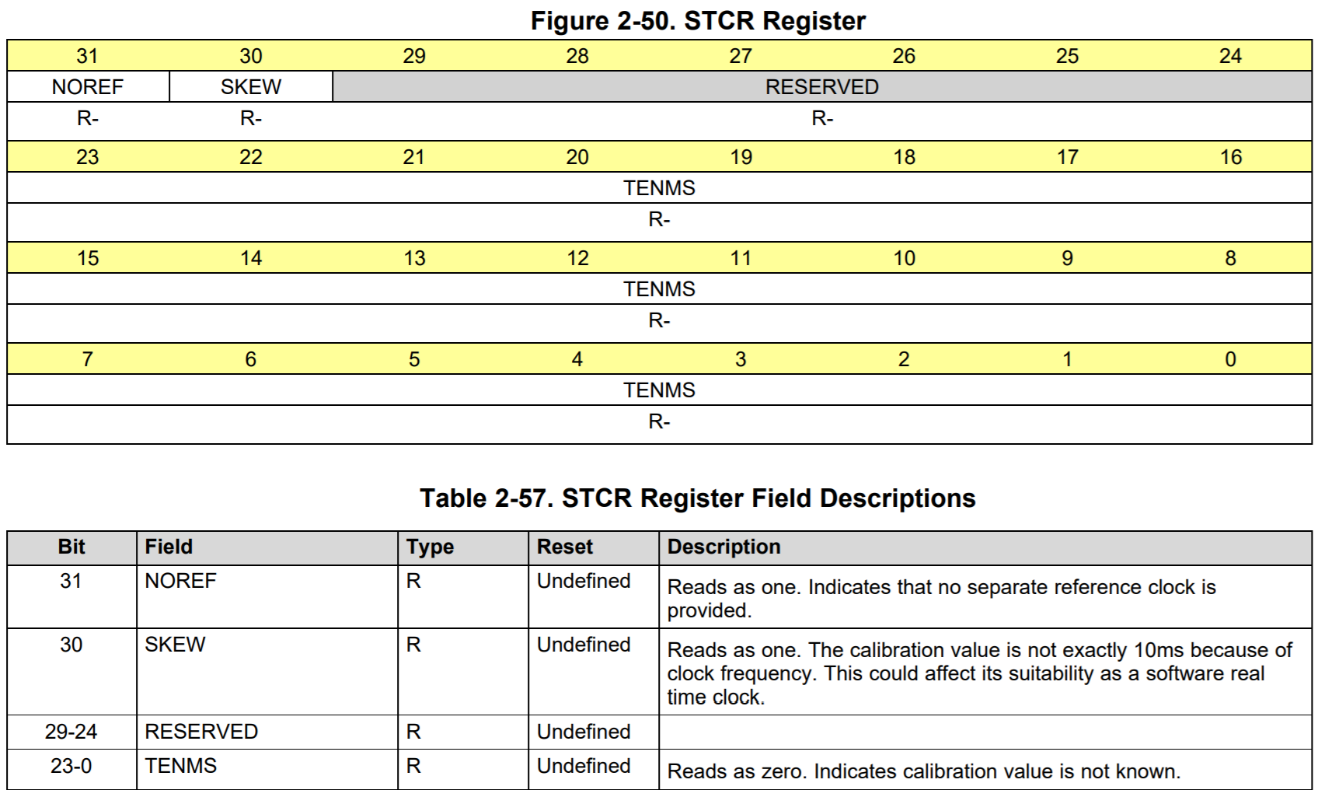
**STCVR Register [SysTick -> VAL]**

**Use the SysTick Current Value Register to find the current value in the register.**

**Writing to it with *any value* clears the register to 0**



***SysTick -> VAL = 0; // any write to current clears it***



***Example***

**SysTick -> CTRL (to access the Control Status Register STCSR)**

**SysTick -> LOAD (to access the Reload Value Register STRVR)**

**SysTick -> VAL (to access the Current Value Register STCVR**

**void *SysTick\_Init (void)***

**{ *//initialization of SysTick timer***

***SysTick -> CTRL = 0; // disable SysTick during step***

***SysTick -> LOAD = 0x00FFFFFF; // max reload value***

***SysTick -> VAL = 0; // any write to current clears it***

***SysTick -> CTRL = 0x00000005; // enable SysTick, 3MHz, No Interrupts***

**}**

**void SysTick\_msdelay (uint16\_t msdelay) // SysTick delay function**

**{**

**SysTick -> LOAD = ((delay \* 3000) - 1); //delay for 1 ms\* delay value**

**SysTick -> VAL = 0; // any write to CVR clears it**

**while ( (SysTick - > CTRL & 0x00010000) == 0); // wait for flag to be SET**

**}**

**void SysTick\_usdelay (uint16\_t usdelay) // SysTick delay function**

**{**

**SysTick -> LOAD = ((delay \* 3) - 1); //delay for 1 us \* delay value**

**SysTick -> VAL = 0; // any write to CVR clears it**

**while ( (SysTick - > CTRL & 0x00010000) == 0); // wait for flag to be SET**

**}**

**For 3 MHz frequency**

**Period T = 1/f**

**T = 1 / 3000000**

**T = 0.333 us = 333.333 ns so each cycle = 1 period = 0.333 us**

**1 ms = 1x103 / 0.333 us**

**1 ms = 3000 cycles**

**With Interrupts**

**Example:**

**Toggle pin 4 PORT 2 every 1 second Using SysTick and interrupts**

**We will use a reload value of 500 \* 3000 (1 ms) to measure 0.5 second**

**volatile uint32\_t timeout ;**

**int main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Halting the Watchdog**

**P2->SEL0 &=~ BIT4 ; // Set pin 4 GPIO**

**P2->SEL1 &=~ BIT4 ; // Set pin 4 GPIO**

**P2->DIR |= BIT4 ; // Set pin 4 OUTPUT**

**SysTick\_Init\_interrupt ( ); // initialize SysTick with interrupt**

**\_\_enable\_irq ( ); // enable global interrupts**

**while (1)**

**{**

**if ( timeout )**

**{**

**P2->OUT ^= BIT4; // Toggle pin 4 every second**

**timeout = 0;**

**}**

**}**

**}**

**void *SysTick\_Init\_interrupt (void)***

**{ *//initialization of SysTick timer***

***SysTick -> CTRL = 0; // disable SysTick during step***

***SysTick -> LOAD = 1500000; // Value for 0.5s interrupts interrupt***

***SysTick -> VAL = 0; // any write to current clears it***

***SysTick -> CTRL = 0x00000007; // enable SysTick, 3MHz,* with Interrupt**

**}**

**void SysTick\_Handler (void) //SysTick handler interrupt**

**{**

**timeout = 1; // set flag =1, reset in main**

**}**

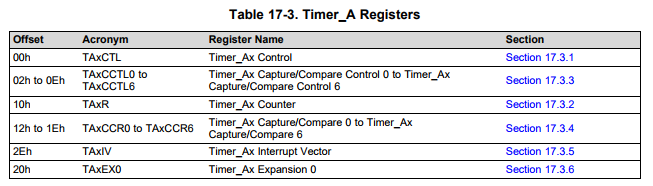
**void SysTick\_Handler (void) //SysTick handler interrupt**

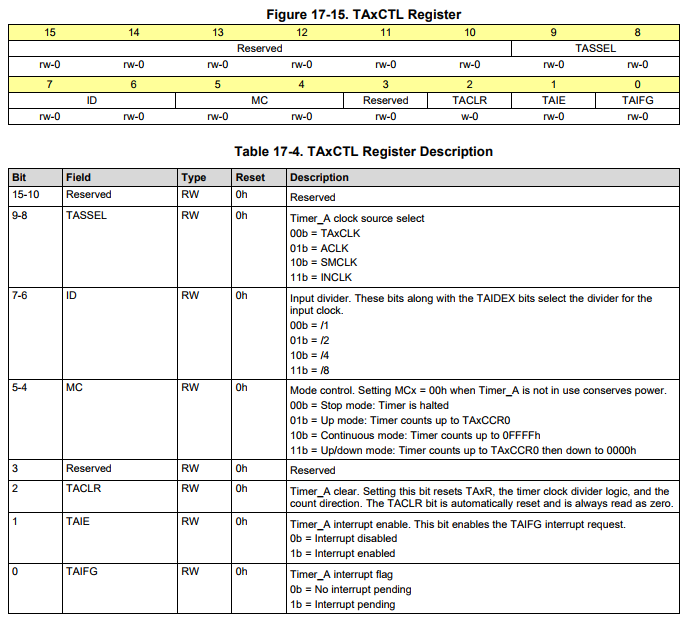
**{**

**count++; //increment count… set as global variable before main**

**}**

**Timer A Registers**





***#define* TIMER\_A\_CTL\_TASSEL\_0 // TAxCLK *#define* TIMER\_A\_CTL\_ID\_0 // divide by 1**

***#define* TIMER\_A\_CTL\_TASSEL\_1 // ACLK *#define* TIMER\_A\_CTL\_ID\_1 // divide by 2**

***#define* TIMER\_A\_CTL\_TASSEL\_2 // SMCLK *#define* TIMER\_A\_CTL\_ID\_2 // divide by 4**

***#define* TIMER\_A\_CTL\_TASSEL\_3 // INCLK *#define* TIMER\_A\_CTL\_ID\_3 // divide by 8**

***#define* TIMER\_A\_CTL\_SSEL\_\_TACLK // TAxCLK**

***#define* TIMER\_A\_CTL\_SSEL\_\_ACLK // ACLK**

***#define* TIMER\_A\_CTL\_SSEL\_\_SMCLK // SMCLK**

***#define* TIMER\_A\_CTL\_SSEL\_\_INCLK // INCLK**

***#define* TIMER\_A\_CTL\_MC\_0 // Stop mode: Timer is halted**

***#define* TIMER\_A\_CTL\_MC\_1 // Up mode: Timer counts up to TAxCCR0**

***#define* TIMER\_A\_CTL\_MC\_2 // Continuous mode: Timer counts up to 0FFFFh**

***#define* TIMER\_A\_CTL\_MC\_3 // Up/down mode: Timer counts up to TAxCCR0 then**

***#define* TIMER\_A\_CTL\_MC\_\_STOP // Stop mode: Timer is halted**

***#define* TIMER\_A\_CTL\_MC\_\_UP // Up mode: Timer counts up to TAxCCR0**

***#define* TIMER\_A\_CTL\_MC\_\_CONTINUOUS // Continuous mode: Timer counts up to 0FFFFh**

***#define* TIMER\_A\_CTL\_MC\_\_UPDOWN // Up/down mode: Timer counts up**

**TIMER\_A0->CTL=TIMER\_A\_CTL\_** **TASSEL\_2 |// SMCLK**

**TIMER\_A\_CTL\_MC\_1|// Up Mode ... Count up**

**TIMER\_A\_CTL\_CLR;// clear TA0R Register**

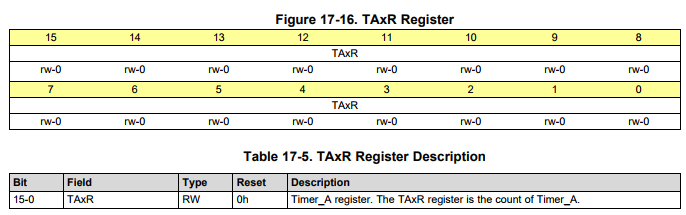
**TIMER\_A0->CTL |= TASSEL\_2 | MC\_1 | TACLR; // SMCLK, Up Mode (Counts to TA0CCR0)**

**TA0CTL |= TASSEL\_2 | MC\_1 | TACLR; // SMCLK, Up Mode (Counts to TA0CCR0)**

**TIMER\_A0->CTL |= 0b0000 0010 0001 0100; // SMCLK, Up Mode (Counts to TA0CCR0)**

**TIMER\_A0->CTL |= 0x0214; // SMCLK, Up Mode (Counts to TA0CCR0**

**Timer/counter register**

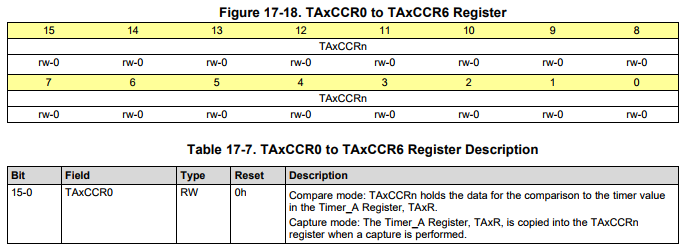


**16-Bit Timer Counter**

**The 16-bit timer/counter register, TAxR, increments or decrements (depending on mode of operation) with each rising edge of the clock signal. TAxR can be read or written with software. Additionally, the timer can generate an interrupt when it overflows.**

**TAxR may be cleared by setting the TACLR bit. Setting TACLR also clears the clock divider and count direction for up/down mode.**

**Timer\_A Capture/Compare 0 Register**



**Changing Period Register TAxCCR0**

**When changing TAxCCR0 while the timer is running, if the new period is greater than or equal to the old period or greater than the current count value, the timer counts up to the new period. If the new period is less than the current count value, the timer rolls to zero. However, one additional count may occur before the counter rolls to zero.**

**Example of a period 1000;**

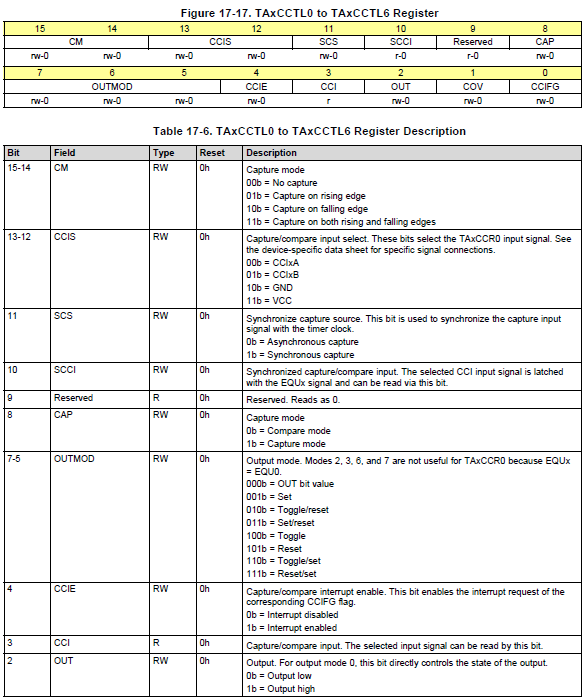
**TIMER\_A0-> CCR[0] = 1000; // PWM Period**

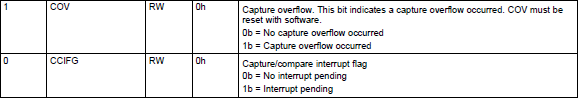
***Or***

**TA0CCR0 |= 1000; // PWM Period**

**Timer\_Ax Capture/Compare Control 0 Register to Timer\_Ax Capture/Compare Control 6**

**CCTL**





***#define* TIMER\_A\_CCTLN\_OUTMOD\_0 // OUT bit value *#define* TIMER\_A\_CCTLN\_CAP\_OFS // Compare mode**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_1 // Set *#define* TIMER\_A\_CCTLN\_CAP // Capture mode**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_2 // Toggle/reset**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_3 // Set/reset**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_4 // Toggle**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_5 // Reset**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_6 // Toggle/set**

***#define* TIMER\_A\_CCTLN\_OUTMOD\_7 // Reset/set**

**Example Reset/Set mode, capture mode**

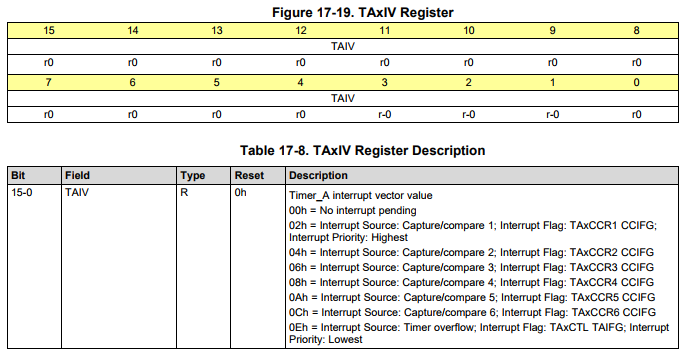
**TIMER\_A0-> CCTL[1] = TIMER\_A\_CCTLN\_OUTMOD\_7; // Reset/set**

**TA0-> CCTL1 |= OUTMOD\_7; // TA0CCR1 output mode = reset/set**

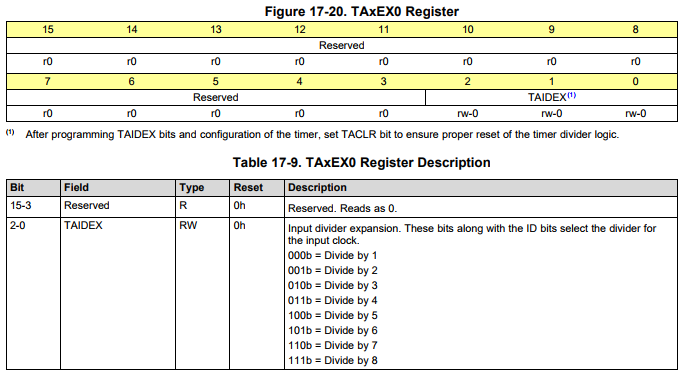
**TIMER\_A0->CCTL[1] = 0b0000 0000 1110 0000; // Reset/set capture compare mode**

**TIMER\_A0->CCTL[1] = 0x00E0; // Reset/set**

**Timer\_Ax Interrupt Vector Register**



**Timer\_Ax Expansion 0 Register**



**MSP432 TimerA Example**

**PWM example using both TimerA\_0 and TimerA\_1 \*/**

**#include< msp.h>**

**void port\_init (void);**

**void TimerA\_init (void);**

**int main (void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**port\_init ( );**

**TimerA\_init ( );**

**while (1)**

**{**

**; //Timer A is connected to 2.4 and 7.7, if you connect an LED to these pins they will**

**// have different intensities**

**}**

**}**

**void port\_init (void) // GPIO Set-Up**

**{**

**P2->SEL0 |= BIT4; // P2.4 selected TA0**

**P2->SEL1 &= ~BIT4; // P2.4 selected TA0**

**P2->DIR |= BIT4; // P2.4 set as ou**

**P7->SEL0 |= BIT7; // P7.7 selected TA1**

**P7->SEL1 &=~ BIT7; // P7.7 selected TA1**

**P7->DIR |= BIT7; // P7.7 set as output**

**}**

**void TimerA\_init (void) //Timer0\_A Set-Up**

**{**

**TIMER\_A0->CCR[0] |= 200 - 1; // PWM Period**

**TIMER\_A0->CCTL[1] |= OUTMOD\_7; // TA0CCR1 output mode = reset/set**

**TIMER\_A0->CCR[1] |= 100; // TA0CCR1 PWM 50% duty cycle**

**TIMER\_A0->CTL |= TASSEL\_2 | MC\_1 | TACLR; // SMCLK, Up mode, Clear**

**TIMER\_A1->CCR[0] |= 1000 - 1; // PWM Period**

**TIMER\_A1->CCTL[1] |= OUTMOD\_7; // TA1CCR1 output mode = reset/set**

**TIMER\_A1->CCR[1] |= 500; // TA1CCR1 PWM duty cycle**

**TIMER\_A1->CTL |= TASSEL\_2 | MC\_1 | TACLR; // SMCLK, Up Mode, Clear**

**}**

**OR you can write in Binary form**

**TIMER\_A0->CTL |= 0b 0000 0010 0001 0100; // SMCLK, Up mode, Clear**

**TIMER\_A1->CTL |= TASSEL\_2 | MC\_1 | TACLR; // SMCLK, Up Mode, Clear**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\***

**\* Nabeeh Kandalaft**

**\* Connect an LED to pin P2.4 and control the intensity of the LED by**

**\* using TIMERA0 PWM**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

***#include* "driverlib.h"**

**voidPWM\_init(void);**

**volatileint *period* =64000;//variable to set the 20 ms period ....*3200 ==>* *1 ms on 3MHz clock***

**volatileint *duty\_cycle* =3200; // variable to set the ON Time for 5% Duty Cycle**

**intmain(void)**

**{**

**WDT\_A->CTL=WDT\_A\_CTL\_PW|WDT\_A\_CTL\_HOLD;// stop watchdog timer**

**PWM\_init();**

**while (1)**

**{**

**TIMER\_A0->CCR[1] = duty\_cycle; //Set breakpoint here**

**}**

**}**

**voidPWM\_init(void)**

**{**

**P2->SEL0|=BIT4;// P2.4 selected TA0**

**P2->SEL1&=~BIT4;// P2.4 selected TA0**

**P2->DIR|=BIT4;// P2.4 set as output**

**TIMER\_A0->CCR[0]=period; // PWM period**

**TIMER\_A0->CCR[1] = duty\_cycle;// Initial duty cycle**

**TIMER\_A0->CCTL[1]=TIMER\_A\_CCTLN\_OUTMOD\_7;// Reset/set**

**TIMER\_A0->CTL=TIMER\_A\_CTL\_TASSEL\_2|// SMCLK**

**TIMER\_A\_CTL\_MC\_1|// Up Mode ... Count up**

**TIMER\_A\_CTL\_CLR;// clear TA0R Register**

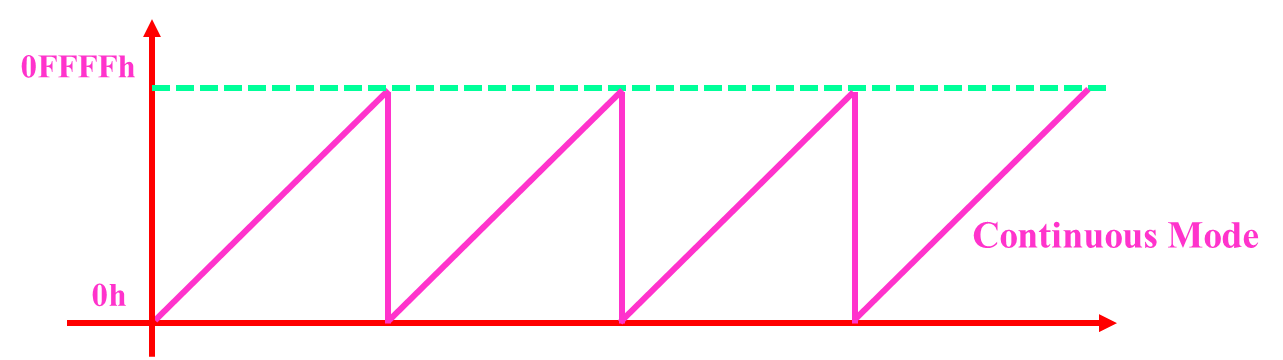
**}**

**The continuous mode can be used to generate independent time intervals and output frequencies. Each time an interval is completed, an interrupt is generated. The next time interval is added to the TAxCCRn register in the interrupt service routine**

**This is controlled in the CCTL register**

**For the over flow 0XFFFF interrupt when it reaches the maximum count**

**It is controlled in CTL register**

****

**Toggle P1.0 using software and the Timer0\_A overflow ISR.**

**In this example an ISR triggers when TA overflows. (Hint…this means using the CTL interrupt flag)**

**Inside the ISR P1.0 is toggled. Toggle rate is exactly 0.5Hz which is 2 seconds.**

**Hint…this means using the ACLK clock 32768 Hz**

**T = 1/ 32768**

**T = 30.51us …… over flow occur at 65535 \* 30.51us …… 2 seconds which is 0.5 Hz**

**#include "msp.h"**

**int main(void)**

**{ WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**void Port\_init ( ); // LED Initialization on PORT1 pin 0**

**TIMER\_A0\_init ( ); // TIMER\_A0 initialization**

**NVIC->ISER[0] = 1 << ((TA0\_0\_IRQn) & 31); // Enable interrupt in NVIC vector**

**\_\_enable\_irq ( ); // Enable global interrupt**

**while (1)**

**{**

**; // Hardware is doing all the work, so nothing more to do in this while loop**

**}**

**}**

**void Port\_init (void)**

**{**

**P1->SEL0 &=~ BIT0;**

**P1->SEL1 &=~ BIT0;**

**P1->DIR |= BIT0;**

**P1->OUT &=~ BIT1;**

**}**

**void TIMER\_A0\_init (void)**

**{**

**TIMER\_A0->CTL = TIMER\_A\_CTL\_SSEL\_\_ACLK | // ACLK 32768 Hz**

**TIMER\_A\_CTL\_MC\_\_CONTINUOUS | // Continuous Mode MC\_2**

**TIMER\_A\_CTL\_CLR | // Clear TAR**

**TIMER\_A\_CTL\_IE; // Enable timer overflow interrupt**

**}**

**void TA0\_0\_IRQHandler(void)**

**{**

**P1->OUT ^= BIT0; // Toggle P1.0 LED**

**TIMER\_A0->CTL[0] &= ~ TIMER\_A\_CTL\_TAIFG; // Clear the flag in CCTL[0]**

**}**

**Toggle P1.0 using software and TA\_0 Interrupt Service Routine.**

**Timer0\_A is configured for continuous mode, thus the timer overflows when TAR counts to CCR0.**

**In this example, CCR0 is loaded with 50000. MCLK = SMCLK = 3MHz**

**#include "msp.h"**

**int main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**void Port\_init ( ) // LED Initialization on PORT1 pin 0**

**TIMER\_A0\_init ( ); // TIMER\_A0 initialization**

**NVIC->ISER[0] = 1 << ((TA0\_0\_IRQn) & 31); // Enable interrupt in NVIC vector**

**\_\_enable\_irq ( ); // Enable global interrupt**

**while (1)**

**{**

**; // Hardware is doing all the work, so nothing more to do in this while loop**

**}**

**}**

**void TIMER\_A0\_init (void)**

**{**

**TIMER\_A0->CCR[0] = 50000;**

**TIMER\_A0->CTL = TIMER\_A\_CTL\_SSEL\_\_SMCLK | // SMCLK**

**TIMER\_A\_CTL\_MC\_\_CONTINUOUS; // Continuous Mode MC\_2**

**TIMER\_A0->CCTL[0] = TIMER\_A\_CCTLN\_CCIE; // TACCR0 interrupt enabled**

**}**

**void Port\_init (void)**

**{**

**P1->SEL0 &=~ BIT0;**

**P1->SEL1 &=~ BIT0;**

**P1->DIR |= BIT0;**

**P1->OUT &=~ BIT1;**

**}**

**void TA0\_0\_IRQHandler(void)**

**{**

**P1->OUT ^= BIT0; // Toggle P1.0 LED**

**TIMER\_A0->CCTL[0] &= ~ TIMER\_A\_CCTLN\_CCIFG; // Clear the flag in CCTL[0]**

**}**

**Example: Scott Zuidema**

**Sets up A0.1 as timer output to P2.4. Sets up A0.4 as timer input (capture on P2.7) on both rising/falling edges**

**If P2.4 and P2.7 are connected the same TIMER\_A0, LED1 will flash twice as fast as LED2. LED2 turns on/off with each Timer compare match.**

**(When the Timer compare matches) and P2.4 on (which happens on Timer A0 overflow)**

**#include "msp.h"**

**void main(void) {**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**Port\_init ( );**

**Timer\_A0\_init ( );**

**NVIC->ISER[0] = 1 << ((TA0\_0\_IRQn) & 31); // Enable interrupt in NVIC vector**

**\_\_enable\_irq ( ); // Enable global interrupt**

**while (1)**

**; // Hardware is doing all the work, so nothing more to do in this while loop.**

**}**

**// Clock is 3 MHz, the setup makes timer A0 have a clock of 3 MHz / 64 = 46.875 kHz.**

**// Timer A0 counts to 65535 which means it overflows once every 1.39 seconds. Each step is 0.00002133 s or 21.33 us.**

**void Timer\_A0\_init (void)**

**{**

**TIMER\_A0->CTL = 0b0000001011010100; // SMCLK, Divide by 8, Count Up, Clear to start**

**TIMER\_A0->CCR[0] = 65535; // A0 count to 2^16 - 1. Set to 46875 for a 1 second flash rate.**

**TIMER\_A0->CCR[1] = 32767; // Duty cycle to 50% to start (1/2 of CCR0).**

**TIMER\_A0->CCTL[1] = 0b0000000011110100; // CCR1 reset/set mode 7, with interrupt.**

**TIMER\_A0->CCTL[4] = 0b1100100100010100; // Capture on rising/falling edge. \*Compared to CCIA (ignore)**

**// synchronized, interrupt enabled**

**//TIMER\_A0-> CCTL[4] = 0b0100100100010100; // Alternative. This set it up for only capture on rising edge.**

**TIMER\_A0->EX0 = 0b0000000000000111; // divide the clock by 1+EX0, in this case, a divider of 8 .**

**TIMER\_A0->CCTL[1] &=~ BIT0; // Clear Timer A0.1 interrupt flag**

**TIMER\_A0->CCTL[1] &=~ BIT1; // Clear Timer A0.1 overflow flag**

**TIMER\_A0->CCTL[4] &=~ BIT0; // Clear Timer A0.4 interrupt flag**

**TIMER\_A0->CCTL[4] &=~ BIT1; // Clear Timer A0.4 overflow flag**

**}**

**// Breaking down the setup of CCTL [4] by bit**

**// Bits 15 - 14 = 11 capture interrupt happens on both rising and falling edges.**

**// Bits 13 - 12 = 00 CCIxA is a reference. Data sheet states that P2.7 is connected to CCIA, so that is selected**

**// Bit 11 = 1 Synchronize. But if things don't occur on clock edges, bad things may happen in future programs. // Bit 10 = 0 A little fuzzy on this, but apparently if things are synchronized, I can read it here later**

**// Bit 9 = 0 Reserved. Either TI uses it when testing, there is a feature on other processors but not**

**// Bit 8 = 1 the magic bit to enable capture (instead of compare)**

**// Bit 7 - 5 = 000 Use these when doing a compare, not for capture**

**// Bit 4 = 1 Enabled the interrupt!**

**// Bit 3 = 0 Read only. You can read the current state of P2.7**

**// Bit 2 = 1 Also for compare. Doesn't do much here. Does the same thing for 0 or 1.**

**// Bit 1 = 0 Clear the overflow flag**

**// Bit 0 = 0 Clear the interrupt flag (don't want it to be on until we actually get a real interrupt).**

**void TA0\_N\_IRQHandler ( ) { // Timer A0 interrupt happened**

***if (TIMER\_A0->CCTL[4] & BIT0) {* // T0.4 was the cause. Successful capture means blink LED1.**

**TIMER\_A0->CCTL[4] &=~ BIT0; // Clear Timer A0.4 interrupt flag**

**TIMER\_A0->CCTL[4] &=~ BIT1; // Clear Timer A0.4 overflow flag**

**P1->OUT ^= BIT0; // Blinking LED1**

**}**

***if (TIMER\_A0->CCTL[1] & BIT0) {* // T0.1 was the cause. Successful compare means blink LED2.**

**TIMER\_A0->CCTL[1] &=~ BIT0; // Clear Timer A0.1 interrupt flag**

**TIMER\_A0->CCTL[1] &=~ BIT1; // Clear Timer A0.1 overflow flag**

**P2->OUT ^= BIT0; // Blinking LED2**

**}**

**}**

**void Port\_init (void)**

**{**

**P1->SEL0 &=~ BIT0; // Setup GPIO**

**P1->SEL1 &=~ BIT0;**

**P1->DIR |= BIT0; // Setup P1.0 as an OUTPUT, initialize to 0**

**P1->OUT &=~ BIT0;**

**P2->SEL0 &=~ BIT0; // Setup GPIO**

**P2->SEL1 &=~ BIT0;**

**P2->DIR |= BIT0; // Setup P2.0 as an OUTPUT, initialize to 0**

**P2->OUT &=~ BIT0;**

**P2->SEL0 |= BIT4;**

**P2->SEL1 &=~ BIT4;**

**P2->DIR |= BIT4; // Setup P2.4 as an TimerA0 controlled OUTPUT, SEL = 01**

**P2->SEL0 |= BIT7;**

**P2->SEL1 &=~ BIT7;**

**P2->DIR &=~ BIT7; // Setup P2.7 an INPUT connected to Timer A0.4 for Capture! SEL = 01**

**}**

**Analog to digital conversion**

**ADC**

**The ADC14 module supports fast 14-bit analog-to-digital conversions. The module implements a 14-bit in SAR core, sample select control, and up to 32 independent conversion-and-control buffers.**

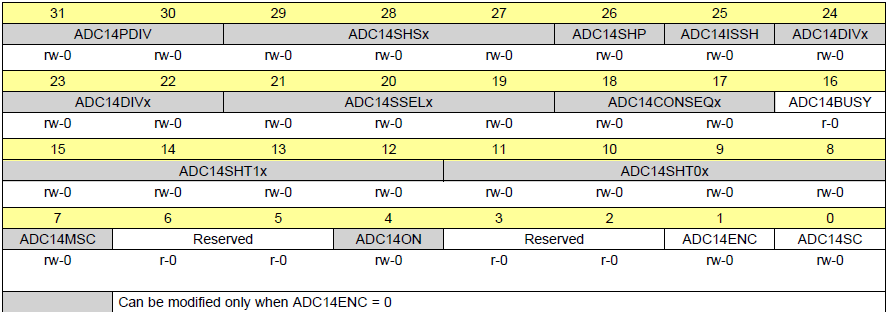
* **The ADC14 core is configured by two control registers, ADC14CTL0 and ADC14CTL1*.***
* **The core is reset when ADC14ON = 0.**
* **When ADC14ON = 1, reset is removed and the core is ready to power up when a valid conversion is triggered.**
* **The ADC14 can be turned off when not in use to save power.**
* **If during a conversion the ADC14ON bit is set to 0, the conversion is abruptly exited, ADC is powered down.**
* **With few exceptions, the ADC14 control bits can be modified only when ADC14ENC = 0.**
* **ADC14ENC must be set to 1 before any conversion can take place.**
* **The conversion results are always stored in binary unsigned format.**
* **For differential inputs, this means that the result has an offset of 8192 added to it to make the number positive. The data format bit ADC14DF in ADC14CTL1 allows the user to read the conversion results as binary unsigned or signed binary (2s complement).**
* **ADC14CLK is used for the conversion clock and to generate sampling period in pulse mode.**
* **The ADC14 source clock is selected using the ADC14SSELx bit.**
* **The input clock can be divided by 1, 4, 32, or 64 using the ADC14PDIV bits**
* **And then subsequently divided by 1 to 8 using the ADC14DIV bits.**
* **Possible ADC14CLK sources are MODCLK, SYSCLK, ACLK, MCLK, SMCLK, HSMCLK**
* **The analog-to-digital conversion requires 9, 11, 14 and 16 ADC14CLK cycles for 8-bit, 10-bit, 12-bit, and 14-bit resolution modes respectively**
* **There are 32 ADC14MEMx conversion memory registers to store conversion results.**
* **Each ADC14MEMx is configured with an associated ADC14MCTLx control register.**
* **The ADC14VRSEL bits define the voltage reference and the ADC14INCHx and ADC14DIF bits select the input channels.**

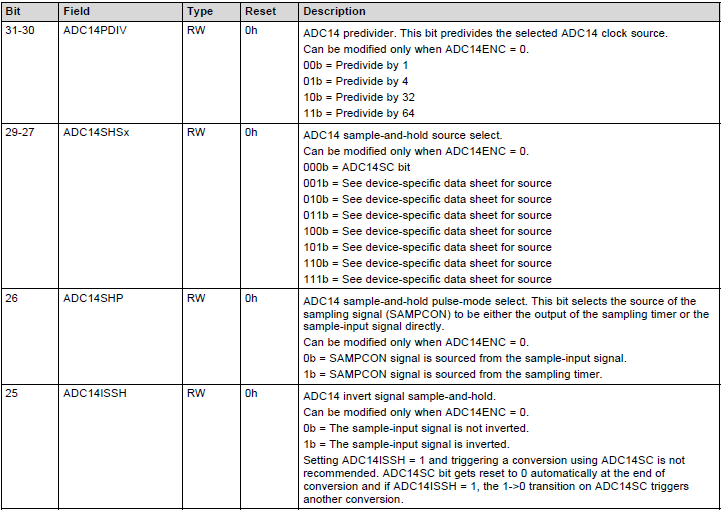
**When conversion results are written to a selected ADC14MEMx, the corresponding flag in the ADC14IFGRx register is set.**

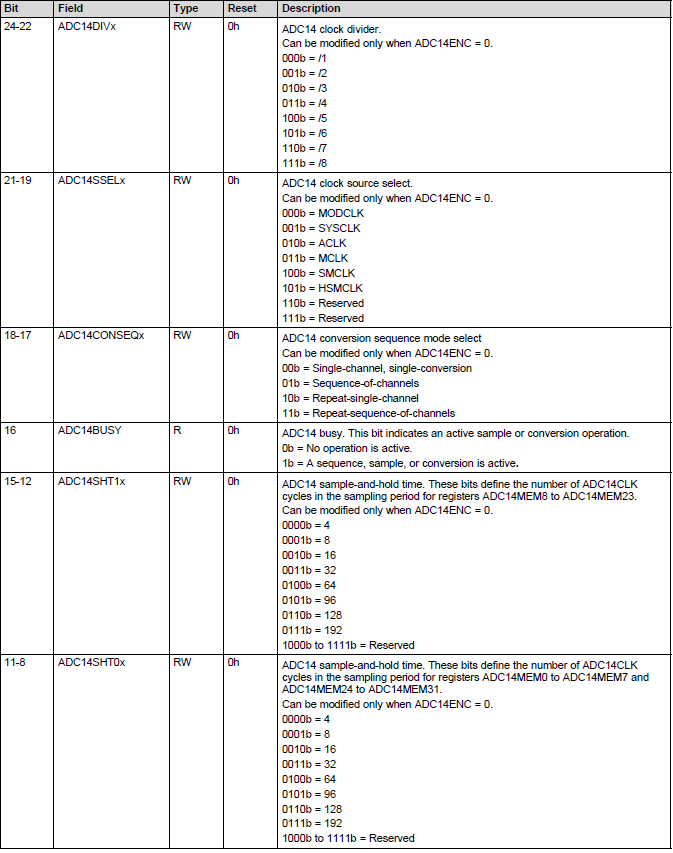
**ADC14CTL0 Register**

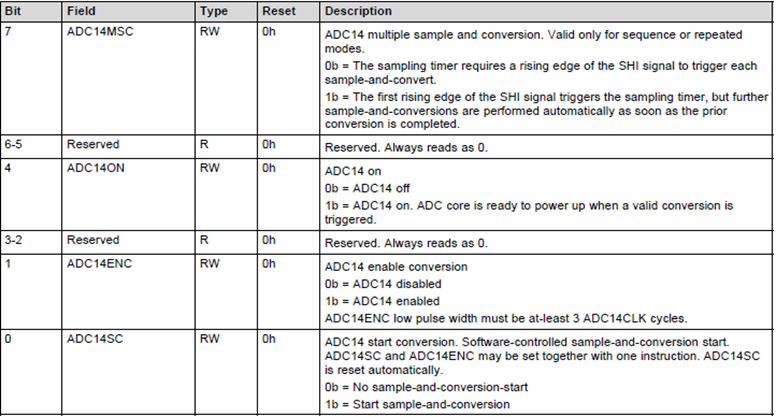
**ADC14 Control 0 Register**

**Figure 20-12. ADC14CTL0 Register**





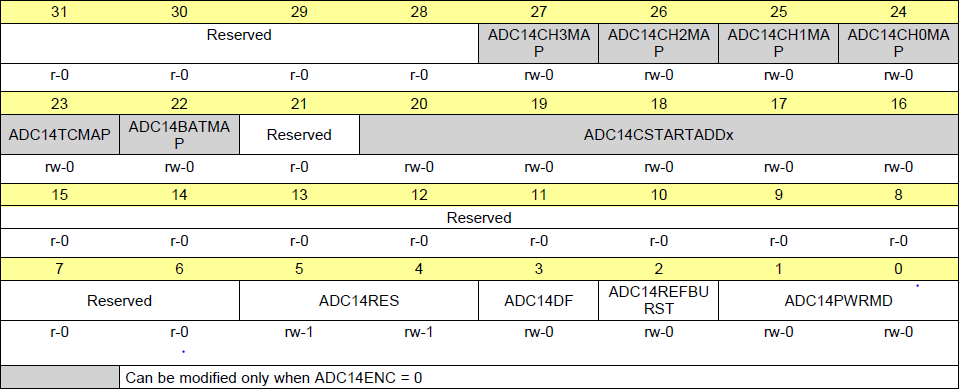


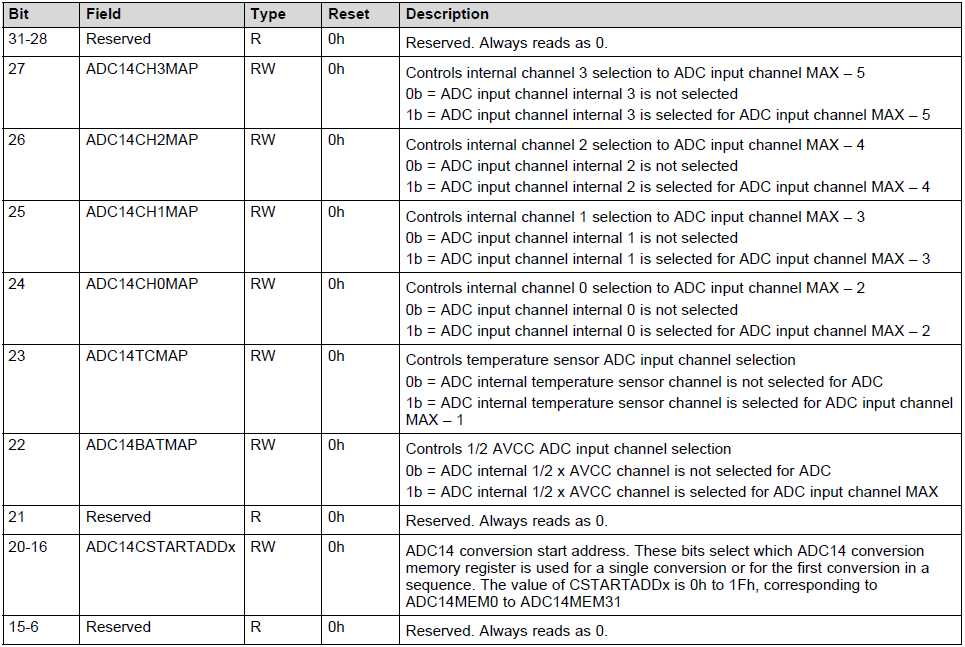
****

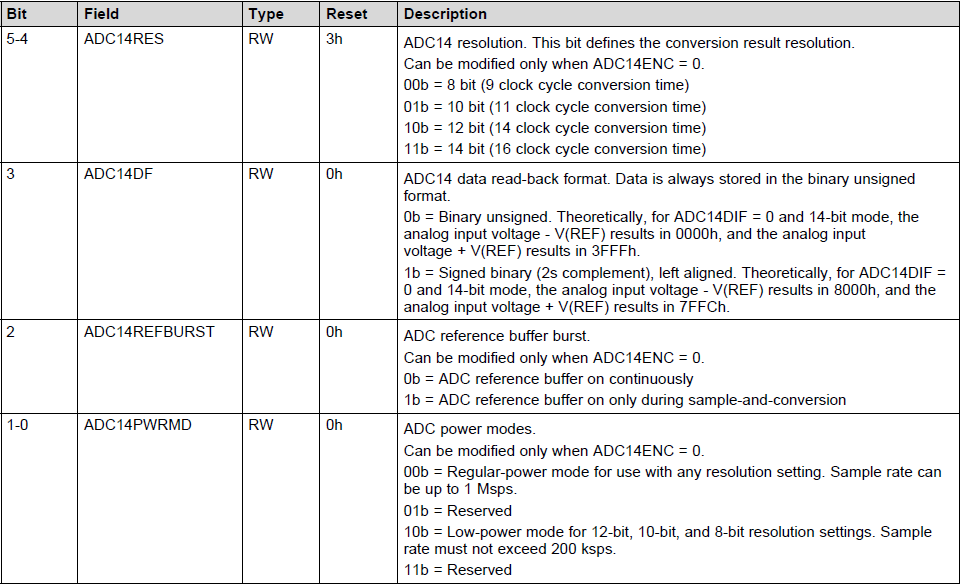
**ADC14CTL1 Register**

**ADC14 Control 1 Register**

**Figure 20-13. ADC14CTL1 Register**



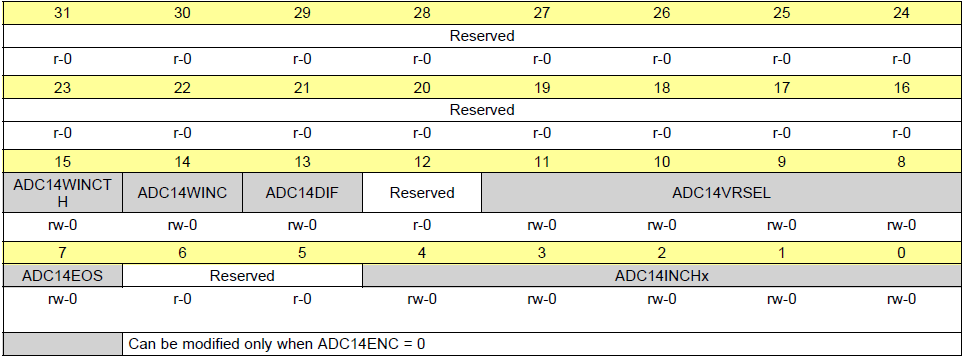


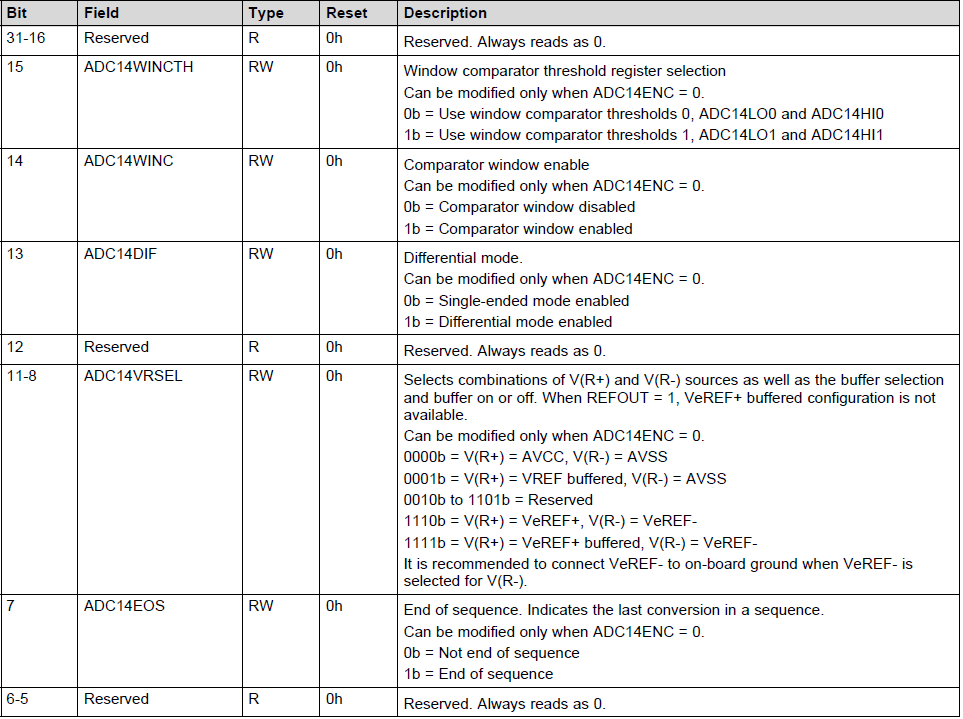


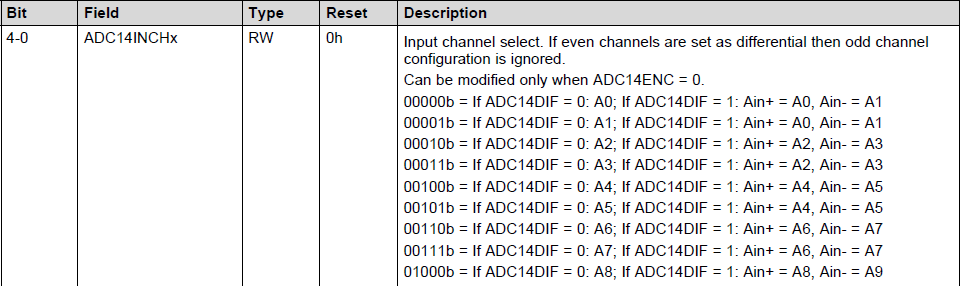
**ADC14MCTL0 to ADC14MCTL31 Register**

**ADC14 Conversion Memory Control x Register (x = 0 to 31)**

**Figure 20-18. ADC14MCTL0 to ADC14MCTL31 Register**



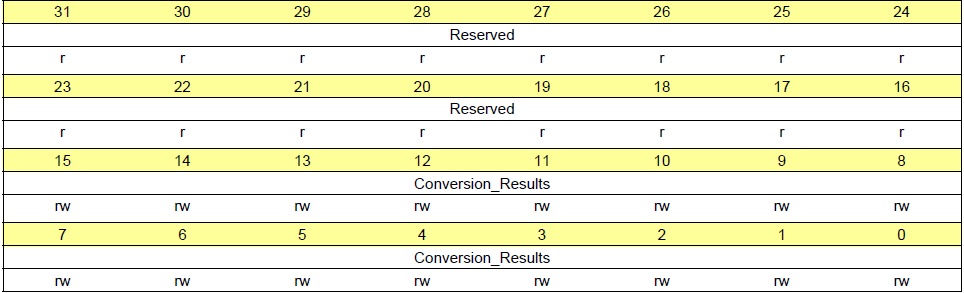


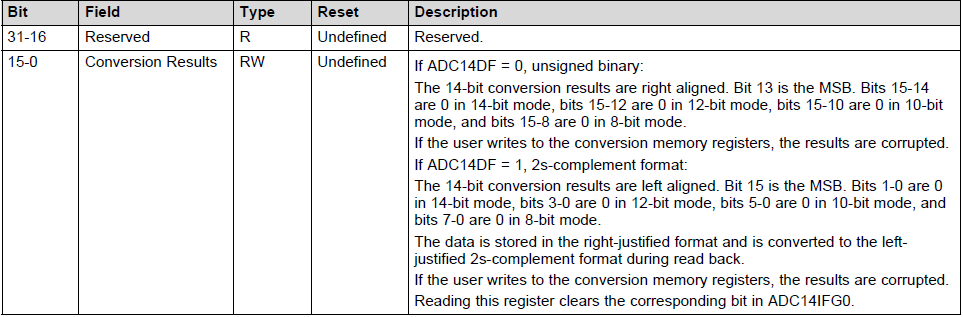


**ADC14MEM0 to ADC14MEM31 Register**

**ADC14 Conversion Memory x Register (x = 0 to 31)**

**Figure 20-19. ADC14MEM0 to ADC14MEM31 Register**

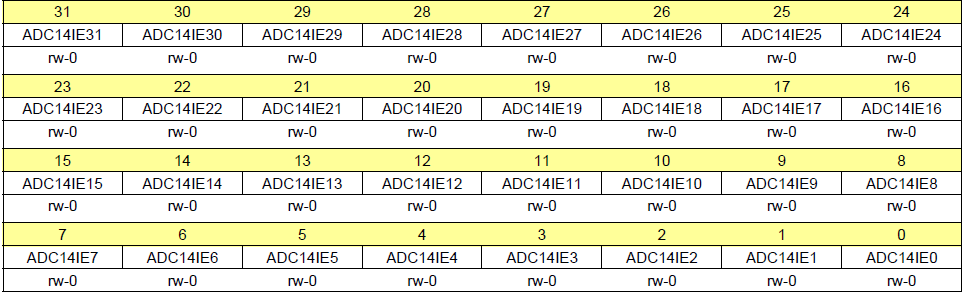


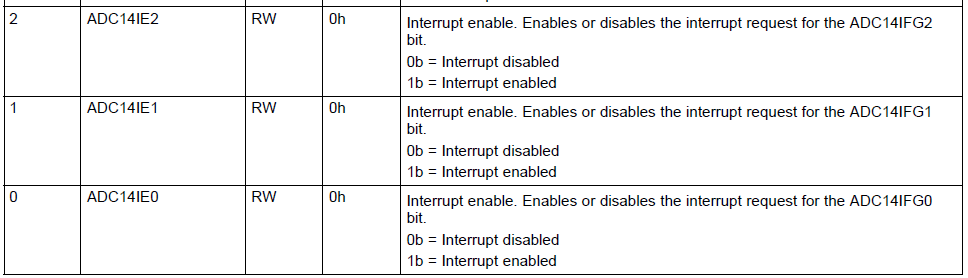


**ADC14IER0 Register**

**ADC14 Interrupt Enable 0 Register**

**Figure 20-20. ADC14IER0 Register**

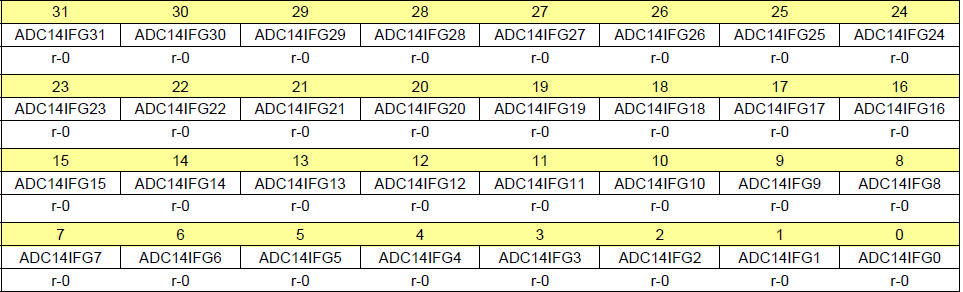




**ADC14IFGR0 Register (offset = 144h) [reset = 00000000h]**

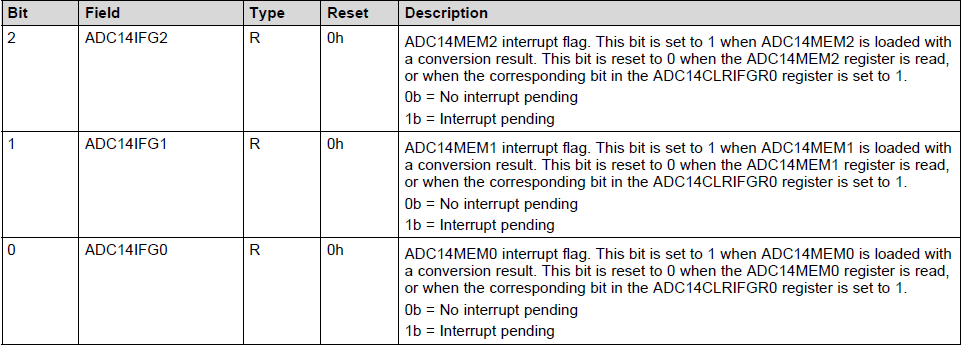
**ADC14 Interrupt Flag 0 Register**

**Figure 20-22. ADC14IFGR0 Register**



**The ADC14IFGx bits are set when their corresponding ADC14MEMx memory register**

**is loaded with a conversion result**

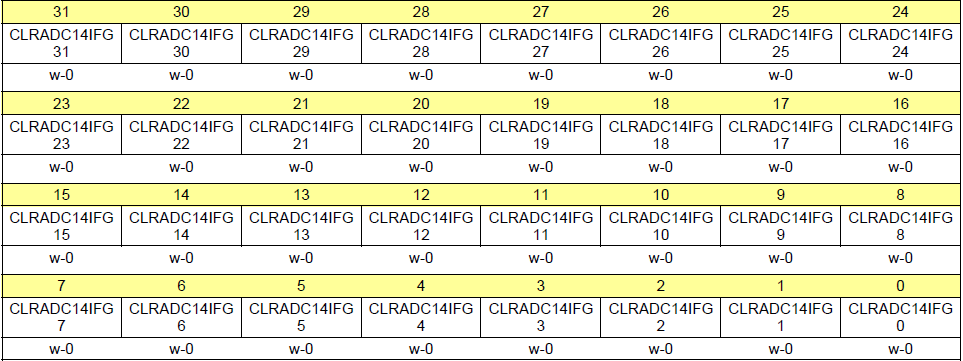


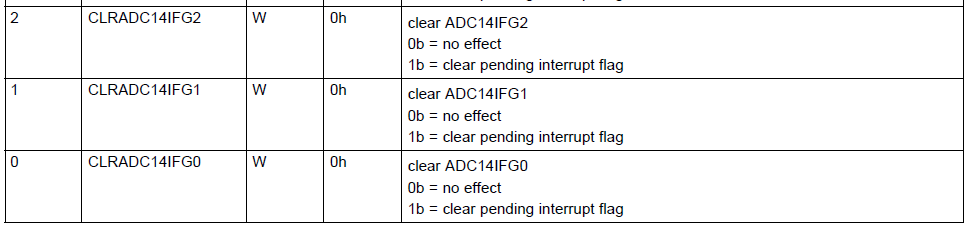
**The rest of the flag bits can be found in the technical reference manual**

**ADC14CLRIFGR0 Register**

**ADC14 Clear Interrupt Flag 0 Register**

**Figure 20-24. ADC14CLRIFGR0 Register**

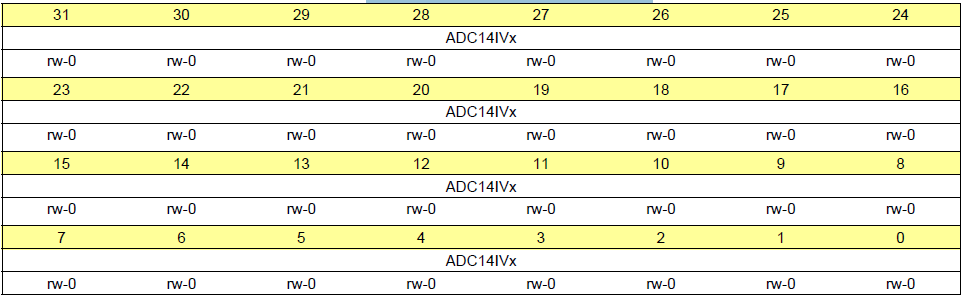


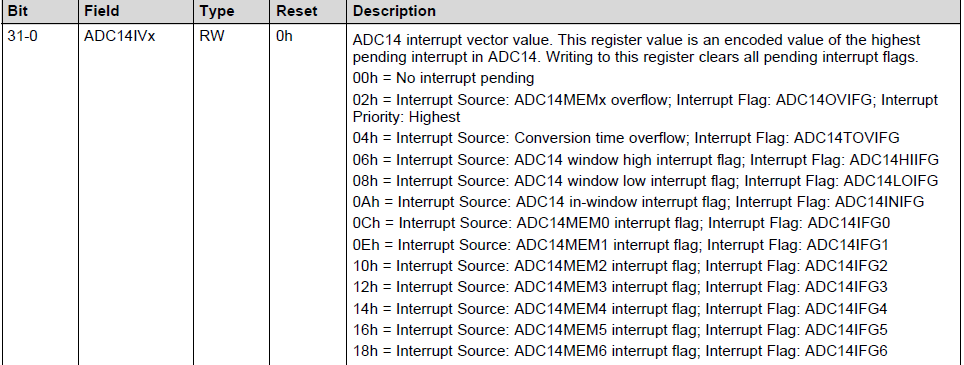


**ADC14IV Register**

**ADC14 Interrupt Vector Register**

**Figure 20-26. ADC14IV Register**





**ADC Examples**

**Connect a photo conductive cell (Photo Diode) in a simple analog circuit designed to create a voltage that increases with increased light level. This voltage will be displayed on the console**

**Use A0 ADC Pin5.5, 14 bit ADC, SMCLK for clock source, and 16 sample clocks**

***#include* "msp.h"**

**voiddelaysetup(void);**

**voiddelayms(uint16\_tdelay);**

**voidADC14\_init (void);**

**main(void)**

**{**

**staticvolatileuint16\_t *result*;**

**float *nADC*;**

**delaysetup ( );**

**ADC14int ( );**

**WDT\_A->CTL=WDT\_A\_CTL\_PW|WDT\_A\_CTL\_HOLD;// stop watchdog timer**

**while(1)**

**{**

**ADC14->CTL0|=1;//start conversation**

**while(!ADC14->IFGR0);//wait for conversation to complete**

**result=ADC14->MEM[0];// get the value from the ADC**

**nADC=(result\*3.3)/16384;**

**printf("Value is:\n\t%d\n\t%f\n",result, nADC);**

**delayms(2000);**

**}**

**}**

**voiddelaysetup (void)// sets up the countdown timer**

**{**

**SysTick->CTRL=0;**

**SysTick->LOAD=0x00FFFFFF;**

**SysTick->VAL=0;**

**SysTick->CTRL=0x00000005;**

**}**

**voiddelayms(uint16\_tdelay)// timer function set to run a 1 ms increments**

**{**

**SysTick->LOAD=((delay\*3000)-1);**

**SysTick->VAL=0;**

**while((SysTick->CTRL&0x00010000)==0);**

**}**

**voidADC14\_init (void)**

**{**

**P5SEL0|=0X20;// configure pin 5.5 for A0 input**

**P5SEL1|=0X20;**

**ADC14->CTL0 *&*=~0x00000002;// disable ADC14ENC during configuration**

**ADC14->CTL0 |= *0x04200210*;// S/H pulse mode, SMCLK, 16 sample clocks**

**ADC14->CTL1 =0x00000030;// 14 bit resolution**

**ADC14->CTL1 |=0x00000000;// Selecting ADC14CSTARTADDx mem0 REGISTER**

**ADC14->MCTL[0]=0x00000000;//** **ADC14INCHx = 0 for mem[0]**

**// ADC14->MCTL[0] = ADC14\_MCTLN\_INCH\_0;**

**ADC14->CTL0|=0x00000002;// enable ADC14ENC, starts the ADC after configuration**

**}**

**Example**

**Measure ambient light intensity with the photocell and microcontroller ADC and use that information to control the light intensity of the LED such that when the ambient light decreases (it gets darker) the intensity of the LED also decreases (it gets dimmer). Use the same parameters from last example. Connect the LED to pin P2.7 for TIMER\_A0**

***#include* "msp.h"**

**voiddelaysetup (void);**

**voiddelayms (uint16\_tdelay);**

**voidADC14\_int (void);**

**voidport\_init (void);**

**voidPWMsetup (void);**

**main(void)**

**{**

**staticvolatileuint16\_t *result*;**

**float *nADC*, *bright*;**

**delaysetup ( );**

**ADC14int ( );**

**pintsetup ( );**

**PWMsetup();**

**WDT\_A->CTL=WDT\_A\_CTL\_PW|WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**while(1)**

**{**

**ADC14->CTL0|=ADC14\_CTL0\_SC;//start conversation or write *ADC14->CTL0 |= 1;***

**while ( ( !ADC14->IFGR0 & BIT0 ) ); //wait for conversion to complete**

**result=ADC14->MEM[0]; // get the value from the ADC**

**nADC=(result\*3.3)/16384;**

**delayms ( 500 );**

**printf ("Value is:\n\t%d\n\t%f\n", result, nADC );**

**bright=nADC/3\*50000; // 3 was the highest voltage from the cell not 3.3.**

**//Always replace by the highest value**

**TIMER\_A0->CCR[4]=bright;**

**}**

**}**

**voiddelaysetup (void) { //sets up the countdown timer**

**SysTick->CTRL=0;**

**SysTick->LOAD=0x00FFFFFF;**

**SysTick->VAL =0;**

**SysTick->CTRL=0x00000005;**

**}**

**voiddelayms (uint16\_tdelay) { //timer function set to run a 1 ms increments**

**SysTick->LOAD=((delay\*3000)-1);**

**SysTick->VAL =0;**

**while((SysTick->CTRL&0x00010000)==0);**

**}**

**voidADC14\_init (void) {**

**P5SEL0 |=0X20;// configure pin 5.5 for A0 input**

**P5SEL1 |=0X20;**

**ADC14->CTL0 *&*=~0x00000002;// disable ADC14ENC during configuration**

**ADC14->CTL0 |= 0x04200210;// S/H pulse mode, SMCLK, 16 sample clocks**

**ADC14->CTL1 =0x00000030;// 14 bit resolution**

**ADC14->CTL1 |=0x00000000;// Selecting ADC14CSTARTADDx mem0 REGISTER**

**ADC14->MCTL[0]=0;//** **ADC14INCHx = 0 for mem[0]**

**ADC14->CTL0|=0x00000002;// enable ADC14ENC, starts the ADC after configuration**

**}**

**voidPort\_init (void)** **{**

**P2->SEL0 |= BIT7; // TIMER\_A\_4 will be connected to led on pin p2.7**

**P2->SEL1&=~BIT7;**

**P2->DIR |=BIT7;**

**}**

**voidPWMsetup (void)** **{**

**TIMER\_A0->CCR[0]=50000-1;//period 🡺 f = 60Hz Ambient light**

**TIMER\_A0->CCR[4] =500;//duty cycle 100% = 50000 500 = 1%**

**TIMER\_A0->CCTL[4]=0xE0; //CCR4 reset/set mode TIMER\_A\_CCTLN\_OUTMOD\_7**

**TIMER\_A0->CTL=0x0214; //use SMCLK, count up, clear TAOR register**

**}**

**Example:**

* **A single sample is made on A1 with reference to AVcc. (A1 Is P5.4 LED is on P1.0)**
* **Software sets ADC14\_CTL0\_SC to start sample and conversion**
* **ADC14\_CTL0\_SC automatically cleared at EOC (End of conversion).**
* **ADC14 internal oscillator times sample (16x) and conversion.**
* **MSP432 waits in LPM0 (Low Power Mode) to save power until ADC14 conversion complete**
* **ADC14\_ISR will force exit from LPM0 in Main loop**
* **If A1 > 0.5 \* AVcc, P1.0 set, else reset.**

**#include "msp.h“**

**int main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Stop WDT**

**Port \_init ( );**

**ADC\_init ( );**

**NVIC->ISER[0] = 1 << ((ADC14\_IRQn) & 31); // Enable ADC interrupt in NVIC module**

**\_\_enable\_irq ( ); // Enable global interrupt**

**while (1)**

**{ // you put a small delay here**

**ADC14->CTL0 |= ADC14\_CTL0\_SC; // Start conversion**

**\_\_sleep ( ); // Go to low power mode**

**}**

**}**

**void Port\_init (void)**

**{**

**P1->SEL0 &=~ BIT0; // Set GPIO**

**P1->SEL1 &=~ BIT0;**

**P1->DIR |= BIT0; // Set P1.0/LED to output**

**P1->OUT &=~ BIT1; // Clear LED to start**

**P5->SEL0 |= BIT4; // Set ADC GPIO**

**P5->SEL1 |= BIT4; // Configure P5.4 for ADC**

**}**

**void ADC\_init (void)**

**{**

**ADC14 -> CTL0 & =~ ADC\_CTL0\_ENC; // disable ADC converter during initialization // ADC14 ON S&H=16, Use sampling timer**

**ADC14->CTL0 = ADC14\_CTL0\_ON | ADC14\_CTL0\_SHT0\_2 | ADC14\_CTL0\_SHP ;**

**ADC14->CTL1 = ADC14\_CTL1\_RES\_3; // 14-bit conversion results**

**ADC14->MCTL[1] |= ADC14\_MCTLN\_INCH\_1; // A1 ADC input select; Vref = AVCC**

**ADC14->IER0 |= ADC14\_IER0\_IE1; // Enable ADC conv complete interrupt**

**ADC14 -> CTL0 |= ADC\_CTL0\_ENC; // Enable ADC converter**

**}**

**void ADC14\_IRQHandler (void)**

**{**

**if (ADC14->MEM[1] >= 0x2000 ) // ADC12MEM1 = A1 > 0.5AVcc?**

**P1->OUT |= BIT0; // P1.0 LED ON**

**else**

**P1->OUT ^= BIT0; // P1.0 LED OFF**

**}**

**Example**

**Changing the period on a Piezo Buzzer using ADC and TIMERA\_0**

**Done by Karl Brokara**

**#include "msp.h"**

**#include <stdio.h>**

**#define ADC\_PORT P5 //ADC Input Port**

**#define ADC\_PIN BIT5 //ADC Input Pin**

**#define ADC\_INST 0 //Which instance of ADC**

**#define SPEAKER\_PORT P2 Speaker PWM Port**

**#define SPEAKER\_PIN BIT4 //Speaker PWM Pin**

**#define SPEAKER\_INST 1 //TIMER A0 Instance for speaker**

**#define SYSTICK\_TICKS 30000 //Which instance of**

**void ADC14init(void);**

**void SysTickInit(void);**

**void TimerA0config(void);**

**void SpeakerConfig(void);**

**uint16\_t adc\_value = 1023; //ADC Measurement set to adc**

**void main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**\_\_disable\_irq();**

**SysTickInit(); //Init SysTick**

**SpeakerConfig(); //Set PWM Pins to drive Speaker**

**TimerA0config(); //Configure Timer A0 to make tone**

**ADC14init(); //Start ADC to read from potentiometer**

**NVIC->ISER[0] |=1<<ADC14\_IRQn; //Turn on ADC Interrupts in NVIC**

**\_\_enable\_irq();**

**while(1);//Loop forever**

**}**

**void ADC14init(void)**

**{**

**/\* Function will set up the ADC14 to run in single measurement mode and to interrupt**

**\* upon conversion.**

**\* Clock Source: SMCLK**

**\* Clock DIV: 32**

**\* Resolution: 10 bits**

**\*/**

**ADC\_PORT->SEL0 |= ADC\_PIN; //Select ADC Operation**

**ADC\_PORT->SEL1 |= ADC\_PIN;**

**ADC14->CTL0 &=~ADC14\_CTL0\_ENC; //Turn of Enable Conversion**

**ADC14->CTL0 |=ADC14\_CTL0\_PDIV\_\_32 | //Set PDIV to 32**

**ADC14\_CTL0\_SHP | //Sample and Hold off of Sampling timer**

**ADC14\_CTL0\_SSEL\_\_SMCLK | //SMCLK**

**ADC14\_CTL0\_SHT1\_\_32 | //Hold memory register for 32 clock cycles**

**ADC14\_CTL0\_ON; //Turn on the ADC**

**ADC14->CTL1|= ADC14\_CTL1\_RES\_\_10BIT; //Set to 10-bit resolution**

**ADC14->MCTL[ADC\_INST]= 0; //Default configuration for ADC Channel**

**ADC14->IER0 |= ADC14\_IER0\_IE0; //Interrupt on**

**ADC14->CTL0 |= ADC14\_CTL0\_ENC; //Enable Conversion**

**}**

**void SysTickInit(void)**

**{**

**SysTick->LOAD = SYSTICK\_TICKS; //Set Interval**

**SysTick->VAL = 0; //Reset Value**

**SysTick->CTRL = 7; //Set CLK, Set IE, Set run**

**}**

**void TimerA0config(void)**

**{**

**TIMER\_A0->CCR[0] = 0xFFFF; //Set the period of Timer0 PWM**

**TIMER\_A0->CCR[SPEAKER\_INST] = 0; //Set the period of Timer0 PWM**

**TIMER\_A0->CCTL[SPEAKER\_INST] = TIMER\_A\_CCTLN\_OUTMOD\_7; //Set to Reset/set Compare Mode**

**TIMER\_A0->CTL = TIMER\_A\_CTL\_TASSEL\_2 | //SMCLK and Input divider 1**

**TIMER\_A\_CTL\_MC\_\_UP |**

**TIMER\_A\_CTL\_CLR; //Up mode and clear the timer**

**}**

**void SpeakerConfig(void)**

**{**

**SPEAKER\_PORT->SEL0 |= SPEAKER\_PIN; //Config to TIMER\_A0 PWM Control**

**SPEAKER\_PORT->SEL1 &= ~SPEAKER\_PIN;**

**SPEAKER\_PORT->DIR |= SPEAKER\_PIN; //Set to output**

**}**

**void SysTick\_Handler(void)**

**{**

**ADC14->CTL0 |=ADC14\_CTL0\_SC; //Start ADC Conversion**

**}**

**void ADC14\_IRQHandler(void)**

**{**

**if(ADC14->IV == (0x0000000C+2\*ADC\_INST)) //If interrupted by ADC\_INST**

**{**

**TIMER\_A0->CCR[0] = 100\*(600 + 15\*(ADC14->MEM[ADC\_INST]>>4))/100; //Set Period of speaker**

**TIMER\_A0->CCR[SPEAKER\_INST] = TIMER\_A0->CCR[0]/2; //Set duty cycle to 50%**

**}**

**ADC14->IV= 0; //Clear ADC interrupt flags**

**}**

***Enhanced Universal Serial Communication Interface***

***(eUSCI) – UART Mode***

**Initializing or reconfiguring the eUSCI\_A module**

The recommended eUSCI\_A initialization/reconfiguration process is:

***1. Set UCSWRST.***

***2. Initialize all eUSCI\_A registers with UCSWRST = 1 (including UCAxCTL1).***

***3. Configure ports.***

***4. Clear UCSWRST with software.* *Clearing UCSWRST releases the eUSCI\_A for operation.***

***5. Enable* interrupts (optional) with UCRXIE or UCTXIE.**

**Setting a Baud Rate**

For a given **BRCLK** clock source **(Baud rate Clock Source),** the baud rate used determines the required division factor N:

**N = fBRCLK / baud rate**

The division factor N is often a non-integer value, thus, at least one divider and one modulator stage is

Used to meet the factor as closely as possible.

If N is equal or greater than 16, it is recommended to use the oversampling baud-rate generation mode by setting UCOS16.

**Baud-rate settings quick set up**

***To calculate the correct settings for the baud-rate generation, perform these steps:***

1. **Calculate N = fBRCLK / baud rate [if N > 16 *[continue with step 3, otherwise with step 2]***
2. **OS16 = 0 UCBRx = INT(N) *[continue with step 4]***

**3. OS16 = 1 UCBRx = INT (N/16),**

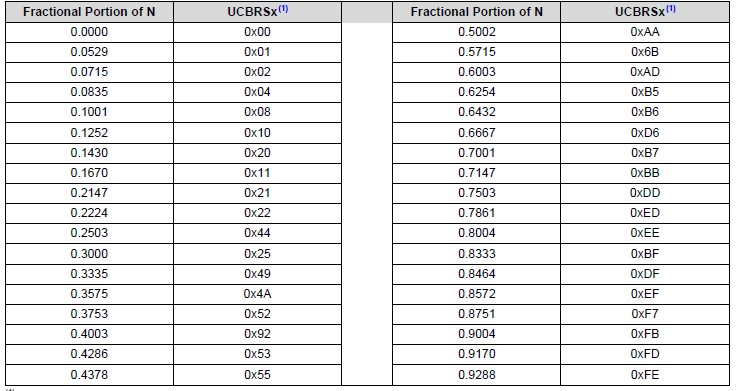
**UCBRFx = INT ([(N/16) – INT (N/16)] × 16)**

**4. UCBRSx can be found by looking up the fractional part of**

**N = N - INT (N) Table 22-4**

**5. If OS16 = 0 was chosen, a detailed error calculation is recommended to be performed**

**Table 22-4. UCBRSx Settings for Fractional Portion of N = fBRCLK/Baud Rate**



**eUSCI\_A has *only one* interrupt vector that is shared for transmission and for reception.**

**UCTXIFG interrupt flag is set by the transmitter to indicate that UCAxTXBUF is ready to accept another character.**

***UCTXIFG is automatically reset if a character is written to UCAxTXBUF.***

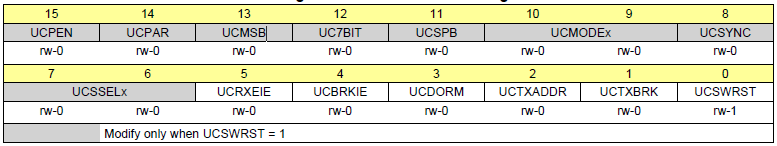
**UCTXIFG is set after a Hard Reset or when UCSWRST = 1.**

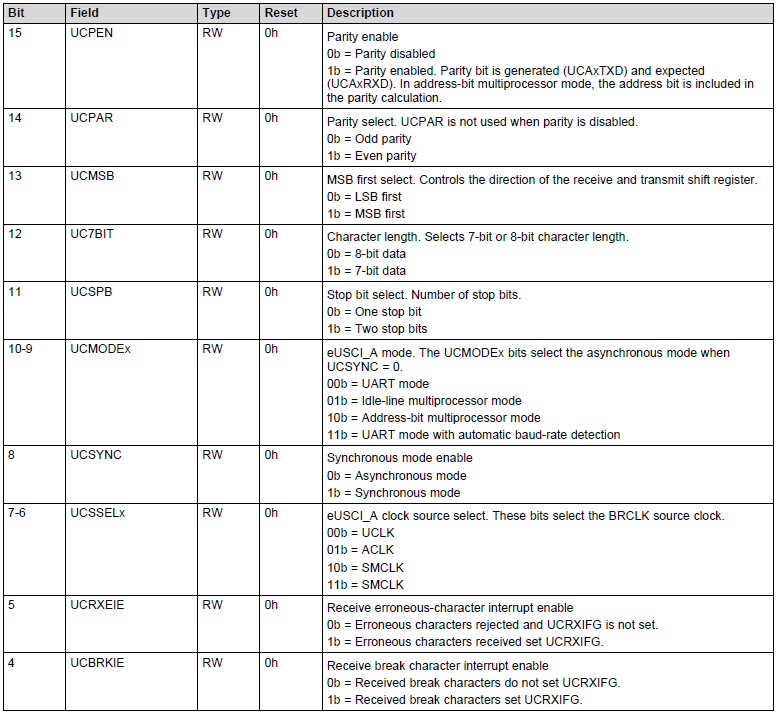
**An interrupt request is generated if UCTXIE is set.**

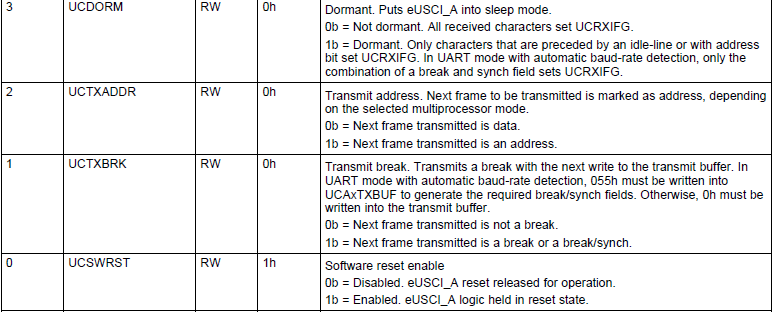
**UCTXIE is reset after a Hard Reset or when UCSWRST = 1.**

**UCAxCTLW0 Register … eUSCI\_Ax Control Word Register 0**

**Figure 22-12. UCAxCTLW0 Register**

****

v



**Example**

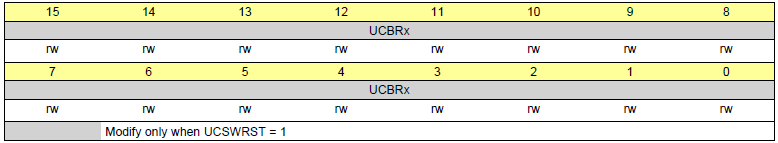
**EUSCI\_A0->CTLW0 = *0x0001;* // hold the USCI module in reset mode**

**EUSCI\_A0->CTLW0 = *0x00C1;***

**EUSCI\_A0->CTLW0 &= *~0x0001;* // enable the USCI module**

**UCAxBRW Register …. eUSCI\_Ax Baud Rate Control Word Register**

**Figure 22-14. UCAxBRW Register**





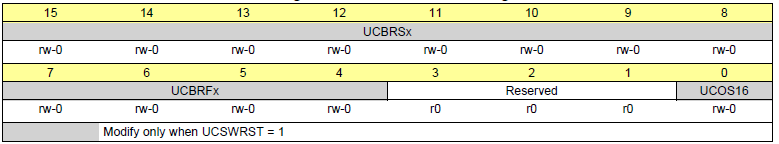
**EUSCI\_A0->BRW = 26; // UCBR = baud rate = int(N) = clock/baud rate**

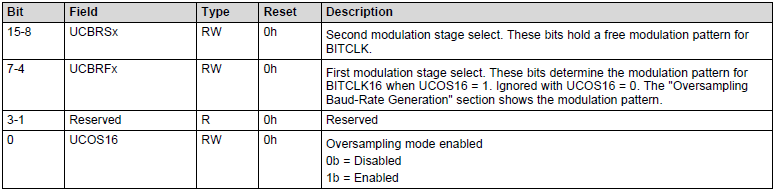
**=clock/baud rate = 3MHz / 115200**

**= 26.0417**

***UCAxMCTLW Register …. eUSCI\_Ax Modulation Control Word Register***

**Figure 22-15. UCAxMCTLW Register**





**EUSCI\_A0->MCTLW &= ~0xFFF1; // clear first and second modulation stage bit fields**

***Example with UCOS16 enabled***

**Baud rate 9600 Frequency 12000000**

**N = 12000000/9600 = >1250**

**OS16 = 1**

**UCBRx = INT(N/16) = INT (12000000/(16\*9600))**

**= 78**

**UCBRFx = INT( [ ( N / 16) – INT(N/16) ] x 16 )**

**= INT( [ (1250 / 16) – INT(71250/16) ] x 16 )**

**= INT( [ 78.125 - 78 ] x 16 )**

**= INT( [ 0 .125 ] x 16 )**

**= 2**

**Fractional portion = 0.125**

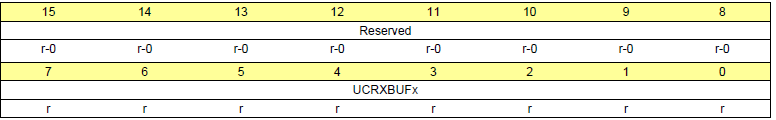
**UCBRSx = 0x10 User'sGuide Table 21-4:**

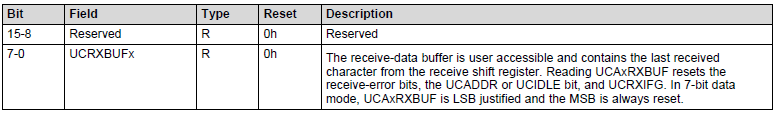
**Code: EUSCI\_A0->MCTLW= 0x1000 | UCOS16 | 0x0020;**

**EUSCI\_A0->BRW = 78;**

***UCAxRXBUF Register ……eUSCI\_Ax Receive Buffer Register***

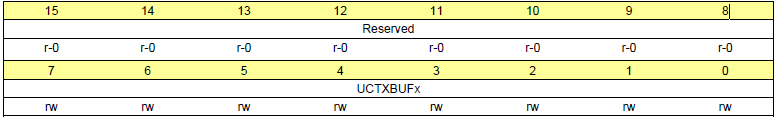
**Figure 22-17. UCAxRXBUF Register**

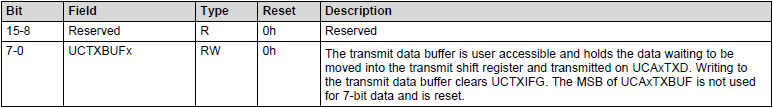




***UCAxTXBUF Register …. eUSCI\_Ax Transmit Buffer Register***

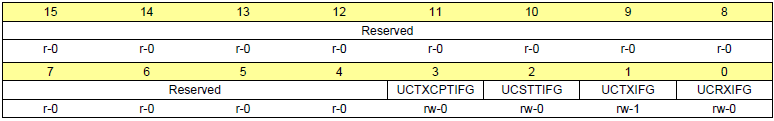
**Figure 22-18. UCAxTXBUF Register**

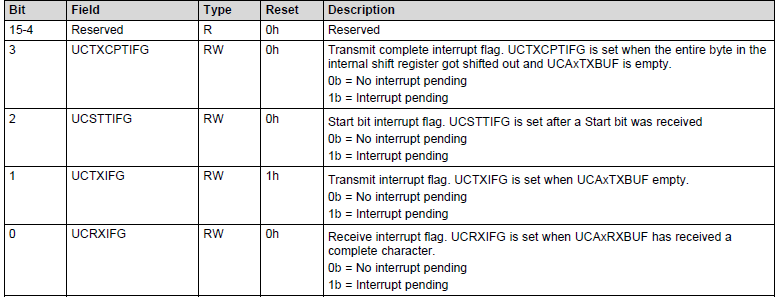




***UCAxIFG Register …. eUSCI\_Ax Interrupt Flag Register***

**Figure 22-22. UCAxIFG Register**



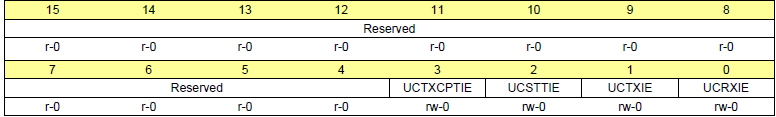


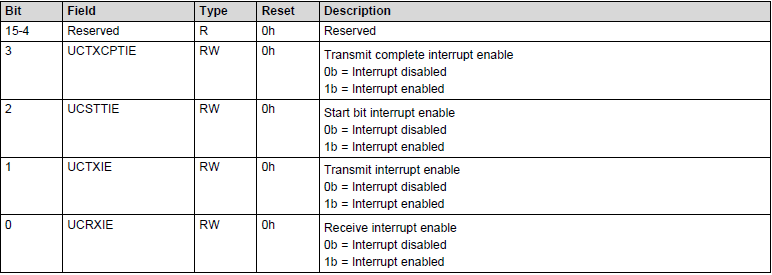
**while ( (EUSCI\_A0 -> IFG&0x01) == 0); // wait till receiving end**

**while ( (EUSCI\_A0 -> IFG&0x02) == 0); // wait till transmitting end**

***UCAxIE Register …. EUSCI\_Ax Interrupt Enable Register***

**Figure 22-21. UCAxIE Register**

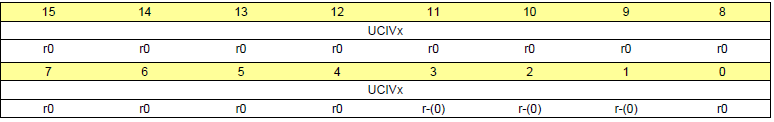


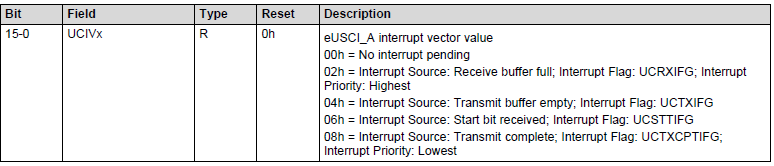


**EUSCI\_A0->IE &= ~0x000F; // disable interrupts**

***UCAxIV Register …. eUSCI\_Ax Interrupt Vector Register***

**Figure 22-23. UCAxIV Register**





**include "msp.h"**

**void UART\_Init(void){**

**EUSCI\_A0->CTLW0 = 0x0001; // hold the USCI module in reset mode**

**// bit15=0, no parity bits**

**// bit14=x, not used when parity is disabled**

**// bit13=0, LSB first**

**// bit12=0, 8-bit data length**

**// bit11=0, 1 stop bit**

**// bits10-8=000, asynchronous UART mode**

**// bits7-6=11, clock source to SMCLK**

**// bit5=0, reject erroneous characters and do not set flag**

**// bit4=0, do not set flag for break characters**

**// bit3=0, not dormant**

**// bit2=0, transmit data, not address (not used here)**

**// bit1=0, do not transmit break (not used here)**

**// bit0=1, hold logic in reset state while configuring**

**EUSCI\_A0->CTLW0 = 0x00C1;**

**EUSCI\_A0->BRW = 26; // UCBR = baud rate = int (N) =26 = clock/baud rate = 26.0417**

**EUSCI\_A0->MCTLW &= ~0xFFF1; // clear first and second modulation stage bit fields,**

**P1SEL0 |= 0x0C;**

**P1SEL1 &= ~0x0C; // configure P1.3 and P1.2 as primary module function**

**EUSCI\_A0->CTLW0 &= ~0x0001; // enable the USCI module**

**EUSCI\_A0->IE &= ~0x000F; // disable interrupts**

**}**

**// Wait for new serial port input, then return byte in receive buffer**

**char UART\_InChar(void){**

**while ( (EUSCI\_A0->IFG&0x01) == 0);**

**return ( (char)(EUSCI\_A0->RXBUF));**

**}**

**// Wait for last byte to be sent, then output character by placing in transmit buffer**

**void UART\_OutChar(char data){**

**while((EUSCI\_A0->IFG&0x02) == 0);**

**EUSCI\_A0->TXBUF = data;**

**}**

**volatile uint8\_t inChar;**

**void main(void)**

**{**

**WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer**

**UART\_Init ( ); // initialize serial port for communication at 115200**

**while(1) {**

**inChar = UART\_InChar(); // wait for character from serial port keyboard emulator**

**UART\_OutChar(inChar); // echo character back to serial port keyboard emulator**

**}**

**}**