CS4101 Introduction to Embedded Systems

Lab 4: Interrupt

Prof. Chung-Ta King

Department of Computer Science

National Tsing Hua University, Taiwan

Introduction

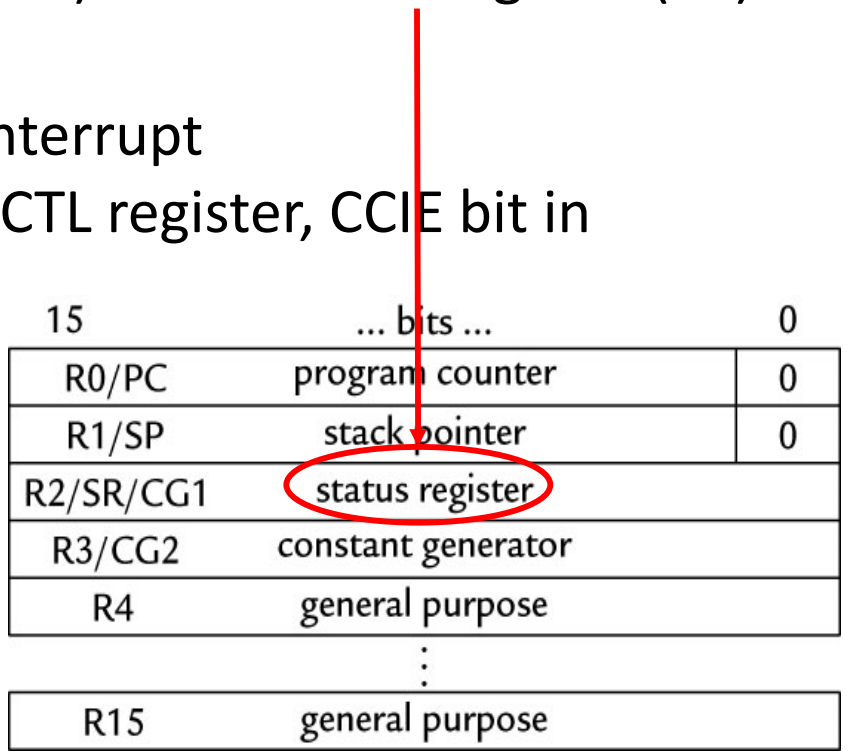
- In this lab, we will learn interrupts of MSP430
  - Handling interrupts in MSP430
  - Handling interrupts of Timer_A in MSP430
  - Handling interrupts of port P1 in MSP430
  - Writing an interrupt service routine

Three Types of Interrupts in MSP430

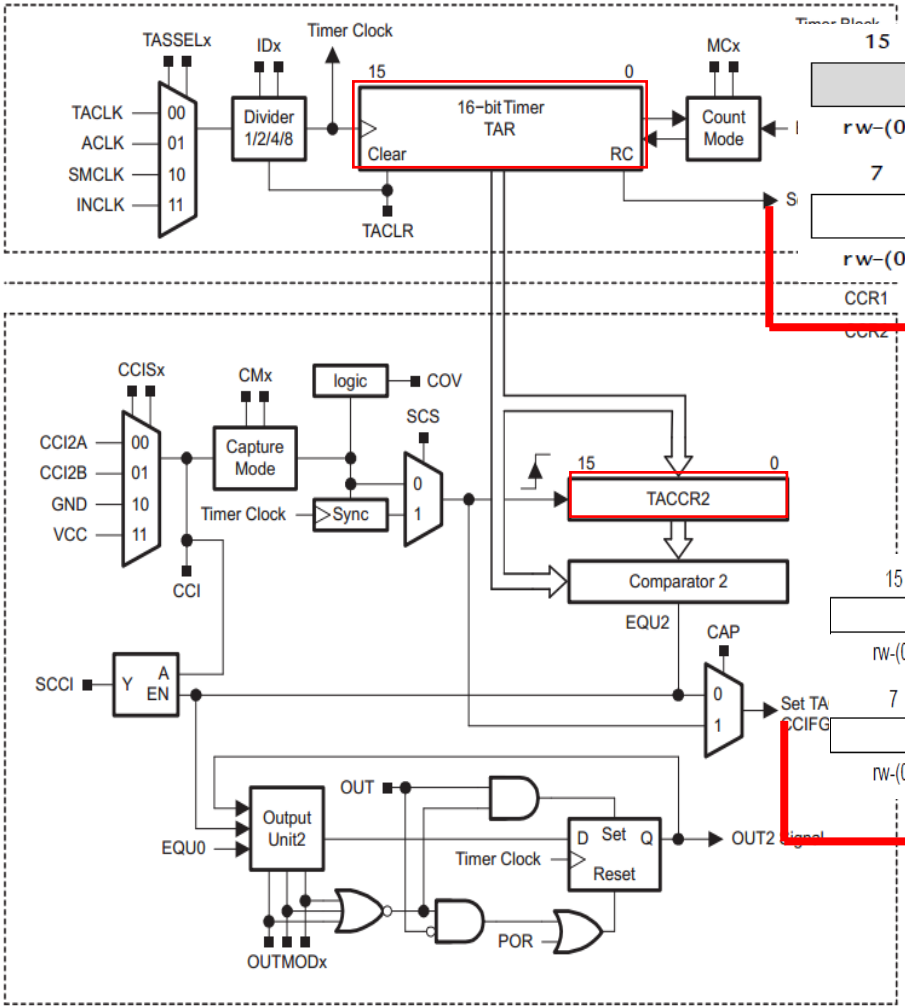
- System reset:
  - Power-up, external reset, Watchdog Timer, flash key violation, PC out-of-range, etc.
  - Always take
- (Non)-maskable interrupt (NMI):
  - RST/NMI pin, oscillator fault, flash access violation
  - Cannot be masked by clearing the GIE bit; but still need bits to be set in special peripheral registers
- Maskable interrupt

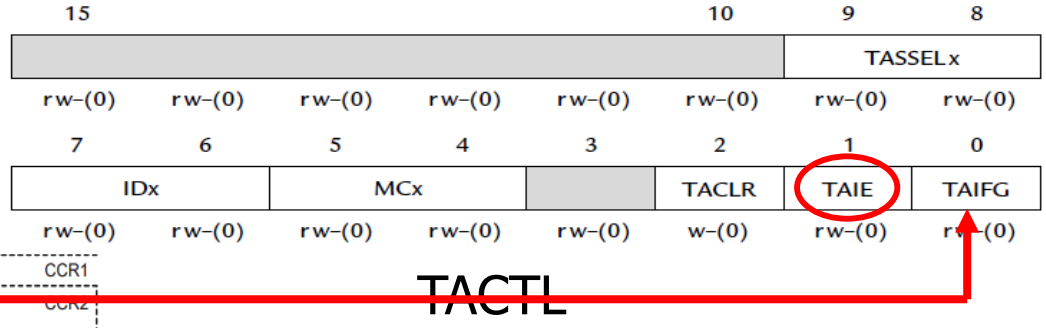
Know When an Interrupt Occurs

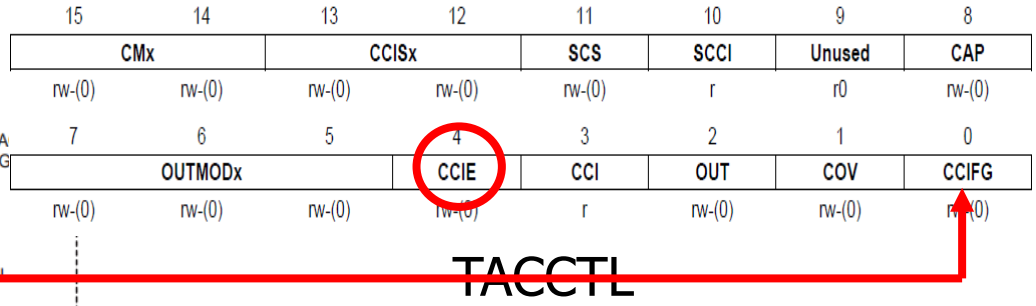
- On MSP430, an interrupt will be detected and serviced if
  - The global interrupt-enable (GIE) bit in Status Register (SR) in CPU is set
  - A peripheral device enables interrupt
    → For Timer_A: TAIE bit in TACTL register, CCIE bit in TACCTLx register
  - The peripheral signals an interrupt
    → For Timer_A: TAIFG, CCIFG



Ex: Timer_A Interrupt Enabling







When an Interrupt Is Requested

- Any currently executing instruction is completed. MCLK is started if the CPU was off.
- The PC, which points to the next instruction, is pushed onto the stack.
- The SR is pushed onto the stack.
- The interrupt with the highest priority is selected.
- The interrupt request flag is cleared automatically for vectors that have a single source.
- The SR is cleared, and maskable interrupts are disabled.
- The interrupt vector is loaded into the PC and the CPU starts to execute the ISR at that address.

These operations take about 6 cycles

After an Interrupt Is Serviced

- An interrupt service routine must always finish with the *return from interrupt* instruction **`reti`**:
  - The SR pops from the stack. All previous settings of GIE and the mode control bits are now in effect.
    → enable maskable interrupts and restores the previous low-power mode if there was one.
  - The PC pops from the stack and execution resumes at the point where it was interrupted. Alternatively, the CPU stops and the device reverts to its low-power mode before the interrupt.

Where to Find ISRs?

- The MSP430 uses *vectored interrupts*
  - Each ISR has its own vector, which is stored at a predefined address in a *vector table* at the end of the program memory (addresses 0xFFC0 ~ 0xFFFF).
  - The vector table is at a fixed location, but the ISRs themselves can be located anywhere in memory.

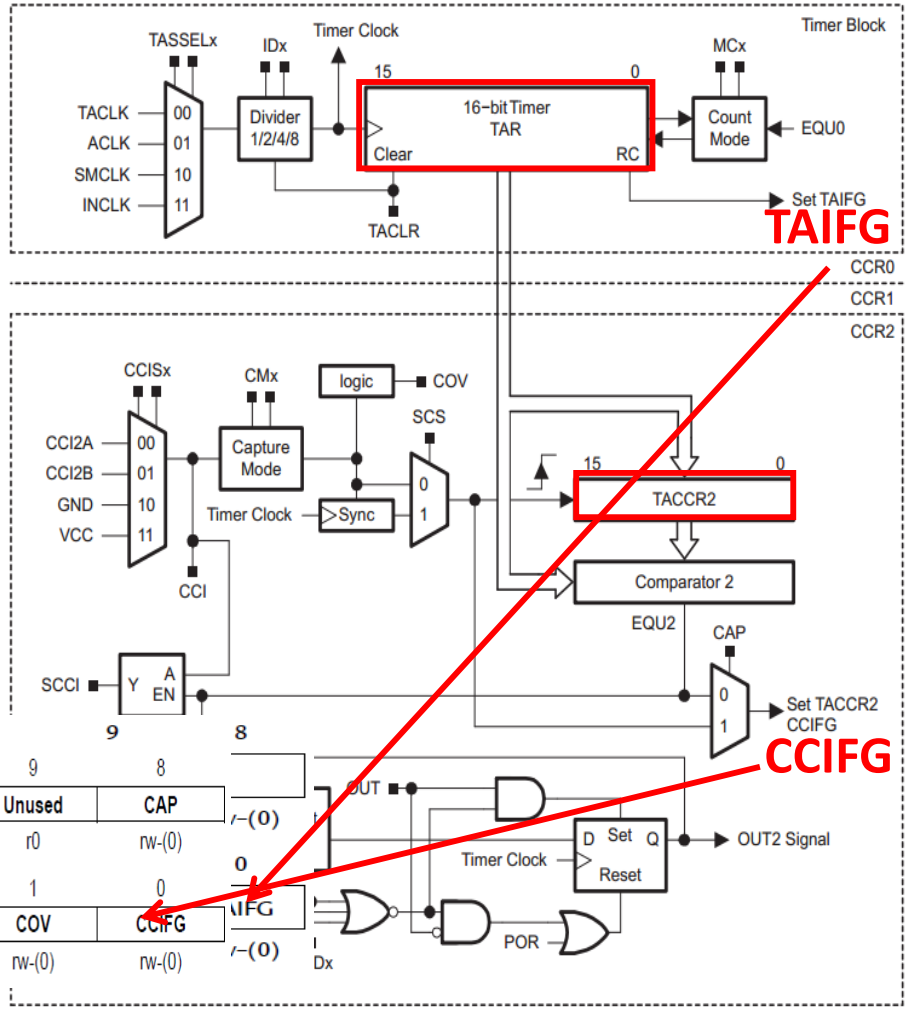
Interrupt Source	Interrupt Flag	System Interrupt	Word Address	Priority
Power-up/external reset/Watchdog Timer+/flash key viol./PC out-of-range	PORIFG RSTIFG WDTIFG KEYV	Reset	0FFFEh	31 (highest)
NMI/Oscillator Fault/ Flash access viol.	NMIIFG/OFIFG/ ACCVIFG	Non-maskable	0FFFCh	30
Timer1_A3	TA1CCR0 CCFIG	maskable	0FFFAh	29
Timer1_A3	TA1CCR1/2 CCFIG, TAIFG	maskable	0FFF8h	28
Comparator_A+	CAIFG	maskable	0FFF6h	27
Watchdog Timer+	WDTIFG	maskable	0FFF4h	26
Timer0_A3	TA0CCR0 CCIFG	maskable	0FFF2h	25
Timer0_A3	TA0CCR1/2 CCIFG, TAIFG	maskable	0FFF0h	24
			0FFEEh	23
			0FFECh	22
ADC10	ADC10IFG	maskable	0FFEAh	21
			0FFE8h	20
I/O Port P2 (8)	P2IFG.0 to P2IFG.7	maskable	0FFE6h	19
I/O Port P1 (8)	P1IFG.0 to P1IFG.7	maskable	0FFE4h	18
			0FFE2h	17
			0FFE0h	16
Unused			0FFDEh 0FFCDh	15 - 0

Outline

- Handling interrupts in MSP430
- Handling interrupts of Timer_A in MSP430
- Handling interrupts of port P1 in MSP430
- Writing an interrupt service routine

Interrupts from Timer_A

- Interrupts can be generated by the timer itself (flag TAIFG) and each capture/compare block (flag TACCRx CCIFG)



15 10 9 8

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	r0	rw-(0)
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw-(0)	rw-(0)	rw-(0)	rw-(0)	r	rw-(0)	rw-(0)	rw-(0)

TACCTL

Two Interrupt Vectors for Timer_A

- For TACCR0 CCIFG (high priority):
  - CCIFG0 flag is cleared automatically when serviced
- For all other CCIFG flags and TAIFG
  - In compare mode, any CCIFG flag is set if TAR counts to the associated TACCRx value
  - Flags are not cleared automatically, because need to determine who made the interrupt request
    - Can use software (ISR) to poll the flags → slow
    - Use hardware: Timer_A interrupt vector register (TAIV)

TAIV

- On an interrupt, TAIV contains a number indicating highest priority enabled interrupt
  - Any access of TAIV resets the highest pending interrupt flag. If another interrupt flag is set, another interrupt is immediately generated

TAIV Contents	Interrupt Source	Interrupt Flag	Interrupt Priority
00h	No interrupt pending	-	
02h	Capture/compare 1	TACCR1 CCIFG	Highest
04h	Capture/compare 2(1)	TACCR2 CCIFG	
06h	Reserved	-	
08h	Reserved	-	
0Ah	Timer overflow	TAIFG	
0Ch	Reserved	-	
0Eh	Reserved	-	Lowest

Sample Code for Timer_A Interrupt

- Toggle LEDs using interrupts from Timer_A in up mode

```
#include <io430x11x1.h> // Specific device
#include <intrinsics.h> // Intrinsic functions
#define LED1 BIT0
void main(void) {
  WDTCTL = WDTPW|WDTHOLD; // Stop watchdog timer
  P1OUT = LED1;   P1DIR = LED1;
  TACCR0 = 49999; // Upper limit of count for TAR
  TACCTL0 = CCIE; // Enable interrupts
  TACTL = MC_1|ID_3|TASSEL_2|TACLR;
  // Up mode, divide clock by 8, clock from SMCLK, clear
  __enable_interrupt(); // Enable interrupts (intrinsic)
  for (;;) { } // Loop forever doing nothing
}
// Interrupt service routine for Timer_A
#pragma vector = TIMER0_A0_VECTOR
__interrupt void TA0_ISR (void){
  P1OUT ^= LED1; // Toggle LED
}
```

Sample Code Explained

- **`#pragma`** line associates the function with a particular interrupt vector
- **`__interrupt`** keyword names the function
  - Compiler will generate code to store address of the function in the vector and to use **`reti`** rather than **`ret`** at the end of the function
- An intrinsic function, **`__enable_interrupt()`** sets the GIE bit and turn on interrupts
  - It is declared in **`intrinsics.h`**

Outline

- Handling interrupts in MSP430
- Handling interrupts of Timer_A in MSP430
- Handling interrupts of port P1 in MSP430
- Writing an interrupt service routine

Interrupts on Port 1

- Ports P1 and P2 can request an interrupt when the value on an input pin changes
- Registers of P1 for interrupt:
  - **Port P1 interrupt enable, P1IE:** enables interrupts when the value on an input pin changes, by setting appropriate bits of P1IE to 1; off (0) by default
  - **Port P1 interrupt edge select, P1IES:** can generate interrupts either on a positive edge (0), when the input goes from low to high, or on a negative edge (1)
  - **Port P1 interrupt flag, P1IFG:** a bit is set when the selected transition has been detected on the input, and an interrupt is requested if it has been enabled.

Interrupts on Port 1

- A single vector for the port
  - The user must check P1IFG to determine the bit that caused the interrupt.
  - This bit must be cleared explicitly

Sample Code for P1

- A hi/low transition on P1.4 triggers P1_ISR to toggles P1.0

```
void main(void) {
  WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
  P1DIR = 0x01;               // P1.0 output, else input
  P1OUT = 0x10;               // P1.4 set, else reset
  P1REN |= 0x10;              // P1.4 pullup
  P1IE |= 0x10;               // P1.4 interrupt enabled
  P1IES |= 0x10;              // P1.4 Hi/lo edge
  P1IFG &= ~0x10;             // P1.4 IFG cleared
  _BIS_SR(GIE);               // Enter interrupt
  while(1);                   // Loop forever
}
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void) {
  P1OUT ^= 0x01;               // P1.0 = toggle
  P1IFG &= ~0x10;              // P1.4 IFG cleared
}
```

Lab 4: Basic 1

- Flash green LED at 1 Hz based on the interrupt from Timer_A, which is driven by SMCLK sourced by VLO.
  - *Hint: For different devices, the "Interrupt Vectors name" may be different; please check the header file for the correct Interrupt Vectors name.*
- Write an ISR for the button, and whenever the button is pushed, make the blinking LED into red LED and vice versa.
  - *Hint: PORT1_VECTOR as interrupt type, P1IFG as interrupt flag.*
- List the assembly code and observe the location of ISR.

Lab 4: Basic 2

- Flash green LED at 0.2 Hz (e.g. turn on for 2.5 sec, turn off 2.5 sec, and so on) based on the interrupt from Timer_A, which is driven by SMCLK sourced by VLO.
- Record TAR in ISR of Timer_A.
- In ISR of P1, record TAR second time and use expression window to see the difference between first and second records.
- Set a break point in the end of P1 ISR and see how fast you can push the button after the LED is off!

Lab 4 Bonus

- Flash both red and green LEDs at 1 Hz. The green LED should be on for 0.5 sec and off for 0.5 sec. The red LED should be on for 0.2 sec and off for 0.8 sec.
- After button pushed, switch the behavior of two LED (e.g. Red LED on for 0.5 sec and off for 0.5 sec and green LED on for 0.2 sec and off for 0.8 sec) using the interrupt.

Lab 4 Bonus hint

- The name of the interrupt vector for TACCR1, TACCR2, and TAR is TIMER0_A1_VECTOR. All three interrupts will cause the CPU to run the same ISR at TIMER0_A1_VECTOR.
- To detect whether it is TACCR1, TACCR2, or TAR that causes the interrupt, please check the register TA0IV. Note that TA0IV will be reset automatically when you read it. Thus, you need to read TA0IV into a local variable first before you check its bits.