University of Strathclyde Engineering

Lecture 2
Timing, Interrupts & Low Power Modes

EE579

Advanced Microcontroller Applications

Dr James Irvine, EEE

DEPARTMENT OF ELECTRONIC & ELECTRICAL ENGINEERING

Setting and Resetting Bits

Set a bit

- OR bit(s) to set with 1, OR other bits with 0
- ? OR 1 = 1, ? OR 0 = ?

```
P1IES |= 0x08;    //Set bit 3 of P1IES
```

Reset a bit

- AND bit(s) to reset with 0, AND others with 1
- ? AND 0 = 0, ? AND 1 = ?

```
P1IES &= ~0x08;   //Reset bit 3 of P1IES
```

Setting and Resetting Bits

Set a bit

- OR bit(s) to set with 1, OR other bits with 0
- ? OR 1 = 1, ? OR 0 = ?

```
    P1IES |= BIT3;    //Set bit 3 of P1IES
```

Reset a bit

- AND bit(s) to reset with 0, AND others with 1
- ? AND 0 = 0, ? AND 1 = ?

```
    P1IES &= ~BIT3;   //Reset bit 3 of P1IES
```

Setting and Resetting Bits

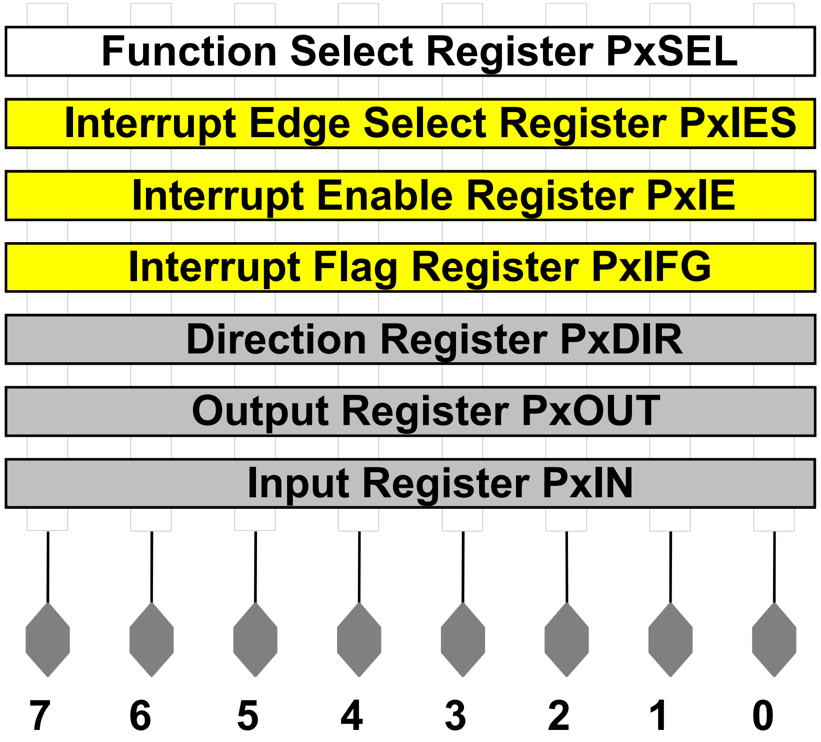
```
/*-------------------------------------------------------------------------
 *   Standard Bits
 *-------------------------------------------------------------------------*/

#define BIT0                (0x0001)
#define BIT1                (0x0002)
#define BIT2                (0x0004)
#define BIT3                (0x0008)
#define BIT4                (0x0010)
#define BIT5                (0x0020)
#define BIT6                (0x0040)
#define BIT7                (0x0080)
#define BIT8                (0x0100)
#define BIT9                (0x0200)
#define BITA                (0x0400)
#define BITB                (0x0800)
#define BITC                (0x1000)
#define BITD                (0x2000)
#define BITE                (0x4000)
#define BITF                (0x8000)
```

Dev Board Switch

- Connected to P1.3 (SW1)
  - Connected between pin and ground
  - **BUT** no physical pull up on the board
  - Must use internal pull up
- When set to interrupt, triggers an interrupt on PORT1_VECTOR

I/O

- 8 bit ports
- 6 to 8 registers per port
- Separate in/out registers
- Each IO line can generate an interrupt
- Interrupts can be rising of falling edge


Function Select Register PxSEL
Interrupt Edge Select Register PxIES
Interrupt Enable Register PxIE
Interrupt Flag Register PxIFG
Direction Register PxDIR
Output Register PxOUT
Input Register PxIN
7
6
5
4
3
2
1
0

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                  // Stop WDT
  P1DIR |= BIT0;                             // P1.0 output
  P1OUT |= BIT3;                             // Select pull up resistor on P1.3
  P1REN |= BIT3;                             // and enable it

  while (1)
  {
    if (!(P1IN & BIT3))                      // Is switched pressed (==0)?
    {
      P1OUT ^= BIT0;                         // Toggle P1.0 to switch LED
      while (!(P1IN & BIT3))                 // Wait for switch to be released
        /* do nothing */ ;
    }
  }
}
```

Starting Point

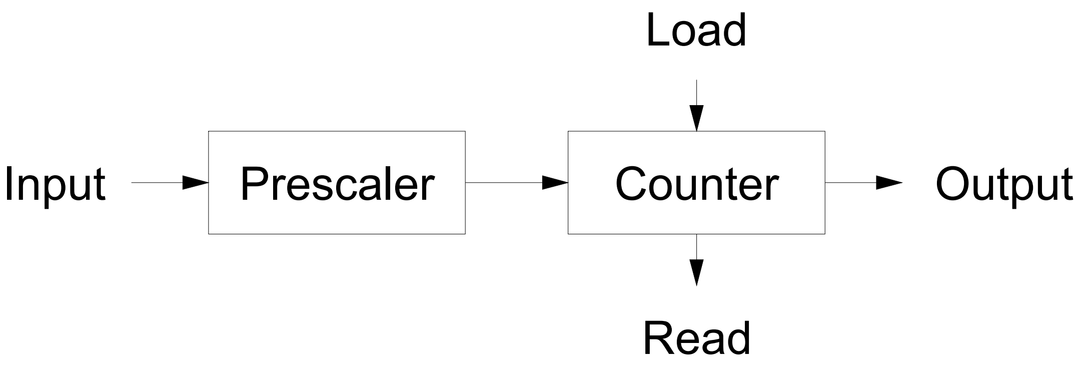
Problem

- We need to flash a light at an appropriate time

Solution

- Use a timer

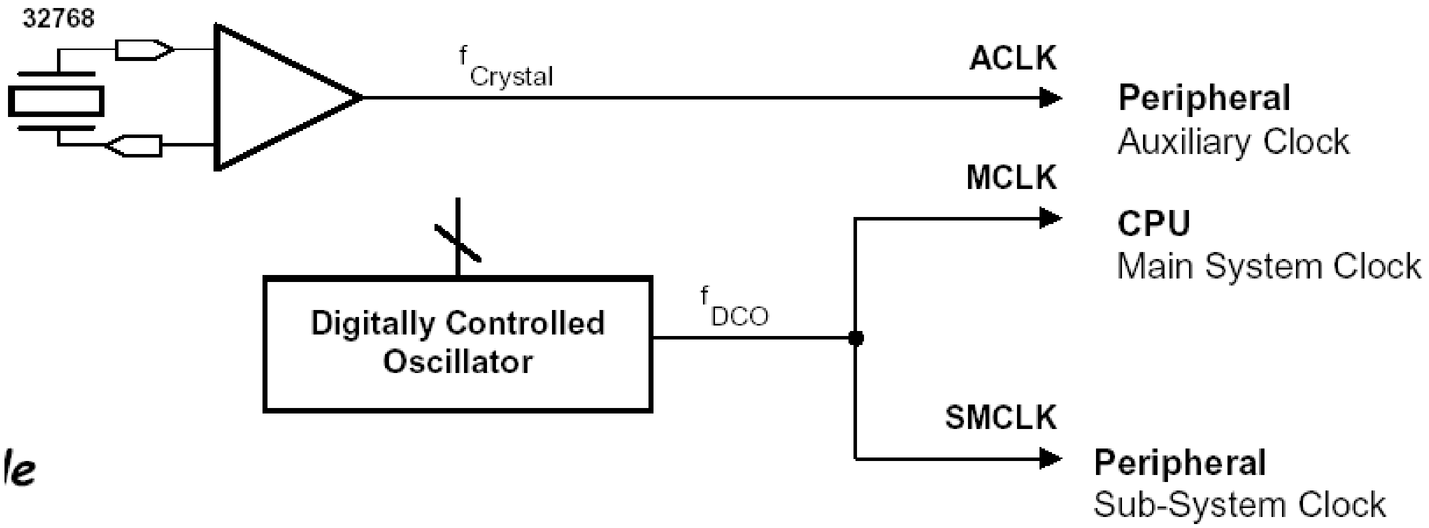
Standard Counter System



- Input
  - Microprocessor clock
  - Secondary clock
  - External signal
- Output
  - Flag
  - Interrupt
  - External signal

Clock Structure

- 3 different clocks on LaunchPad
- ACLK only on LaunchPad if you move the 0R links (initially connected as I/O not crystal)



32768

$f_{Crystal}$

ACLK

Peripheral
Auxiliary Clock

MCLK

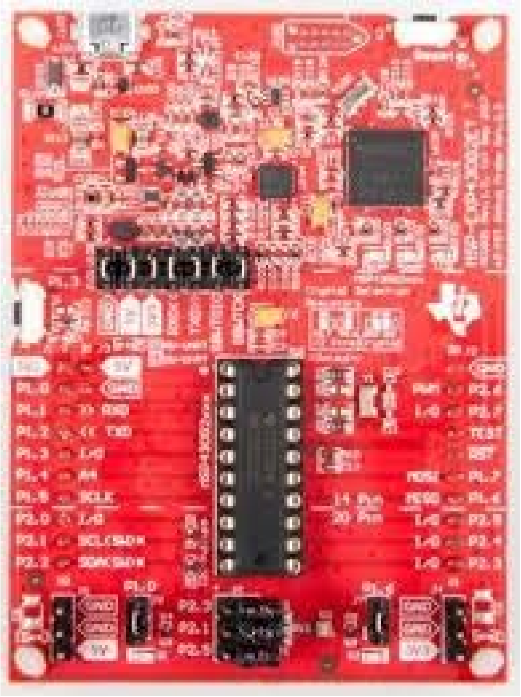
CPU
Main System Clock

Digitally Controlled Oscillator

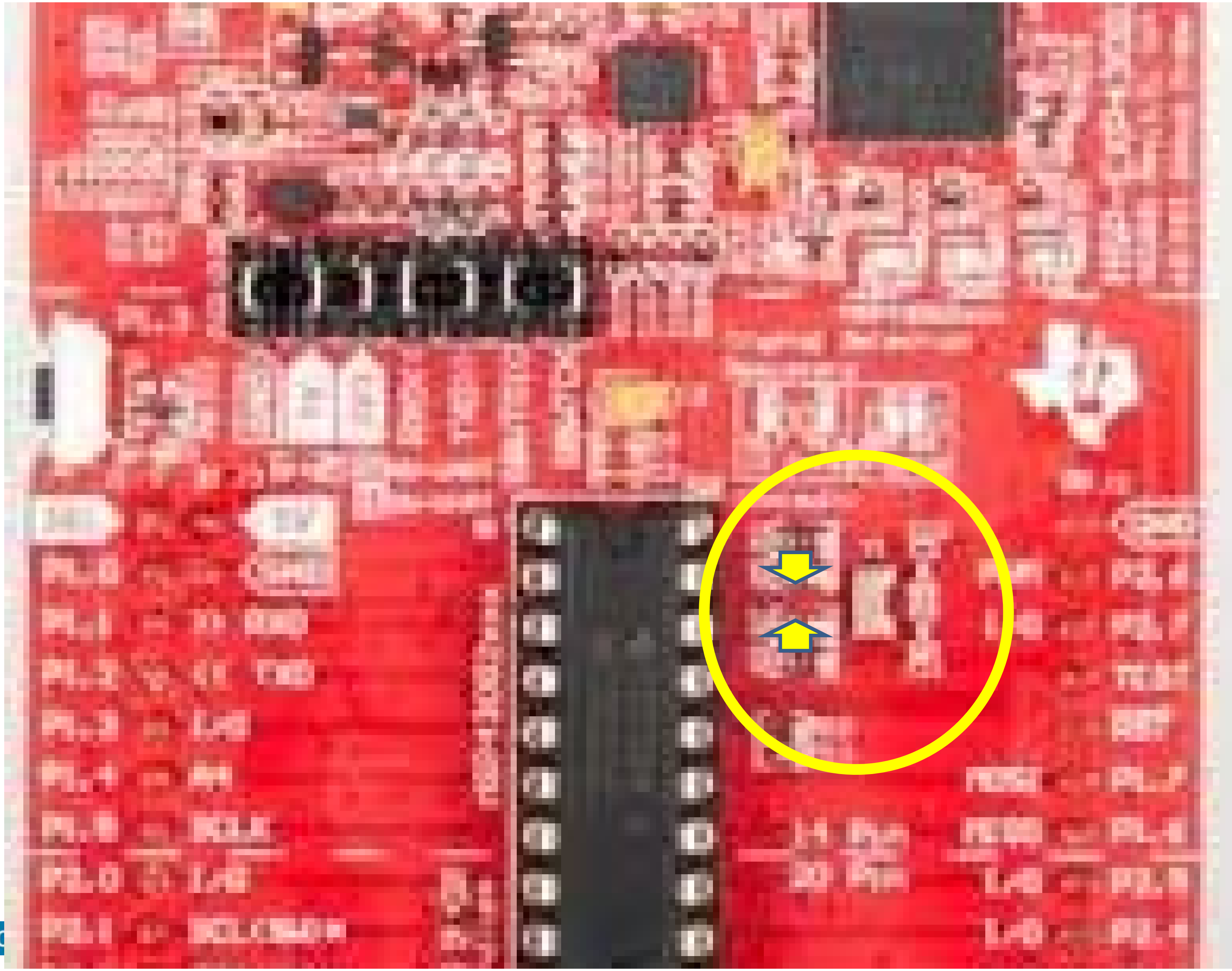
f_{DCO}

SMCLK

Peripheral
Sub-System Clock



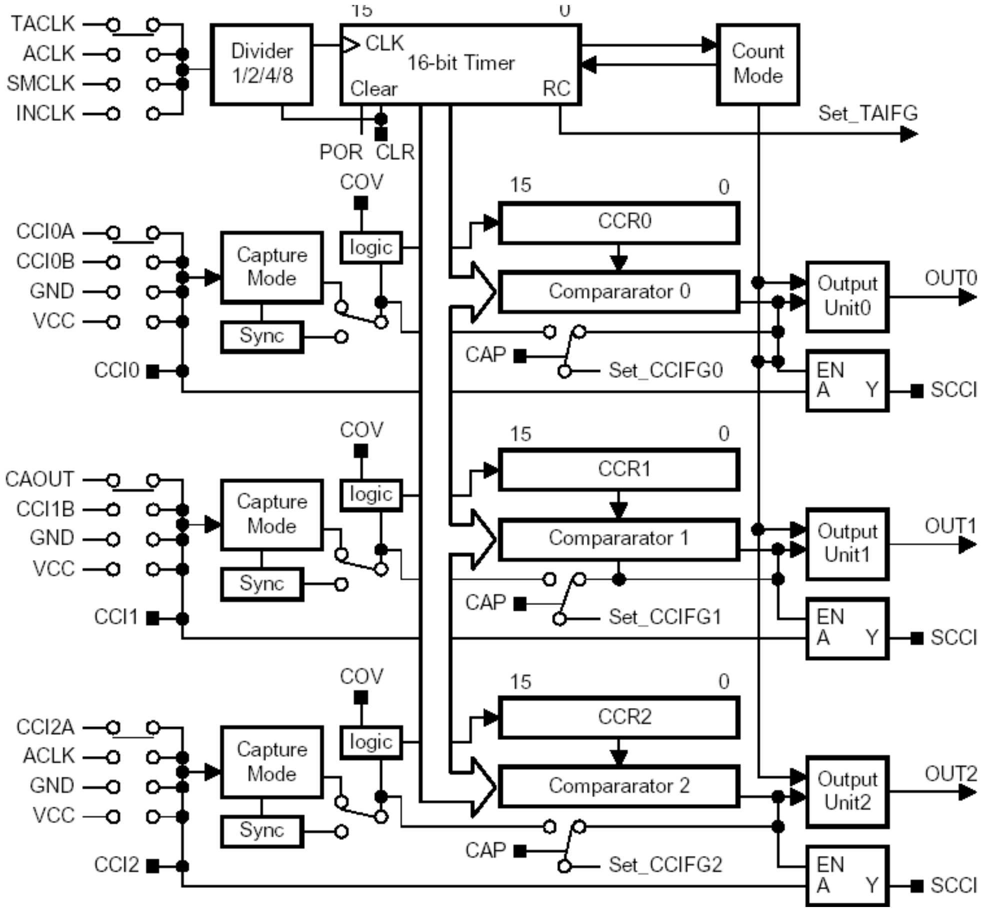
le



Timer_A3

- Flexible 16-bit timer/counter
- Four input clocks, including external signal
  - Prescaler (/1,/2,/4,/8)
- Selectable mode (time/count, capture/ compare)
- Extensive interrupt capability
- Three capture/compare registers (CCR) generate events when value reached
  - Used, for example, for PWM generation
- Can drive external pin
- 2 available on MSP430G2553

Timer_A3



Note timer on MSP430G2231 is Timer_A2 - has only 2 capture/compare sections, not three

Option 1: Wait for a Flag

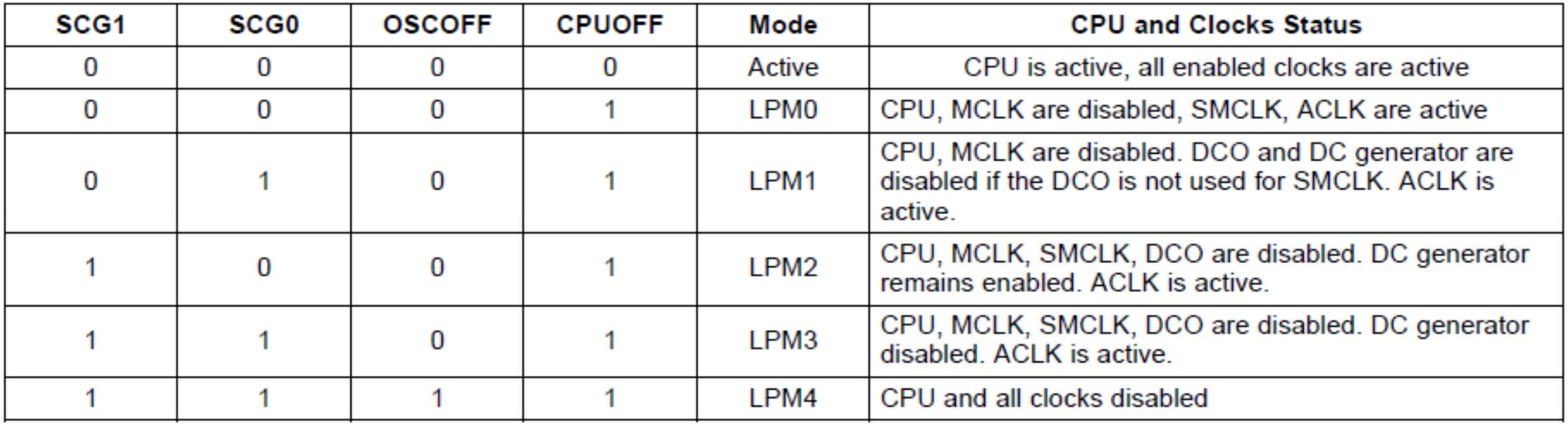
- Since we have nothing else for the processor to do, we can choose the simple option of waiting for a flag to change on the timer
- The comparator status can be read from a bit
  - SCCI (Synchronized capture/compare input) in TAxCCTLn register of the relevant comparator
- Wait for that bit to go high

Option 1: Wait for a Flag

- Inefficient…

- On a simple microcontroller, may be okay
  - What else would you do?
- MPS430 has sophisticated low power modes
- Processor and other peripherals can be switched off to save power

Low Power Modes



SCG1	SCG0	OSCOFF	CPUOFF	Mode	CPU and Clocks Status
0	0	0	0	Active	CPU is active, all enabled clocks are active
0	0	0	1	LPM0	CPU, MCLK are disabled, SMCLK, ACLK are active
0	1	0	1	LPM1	CPU, MCLK are disabled. DCO and DC generator are disabled if the DCO is not used for SMCLK. ACLK is active.
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO are disabled. DC generator remains enabled. ACLK is active.
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO are disabled. DC generator disabled. ACLK is active.
1	1	1	1	LPM4	CPU and all clocks disabled

Low Power Modes

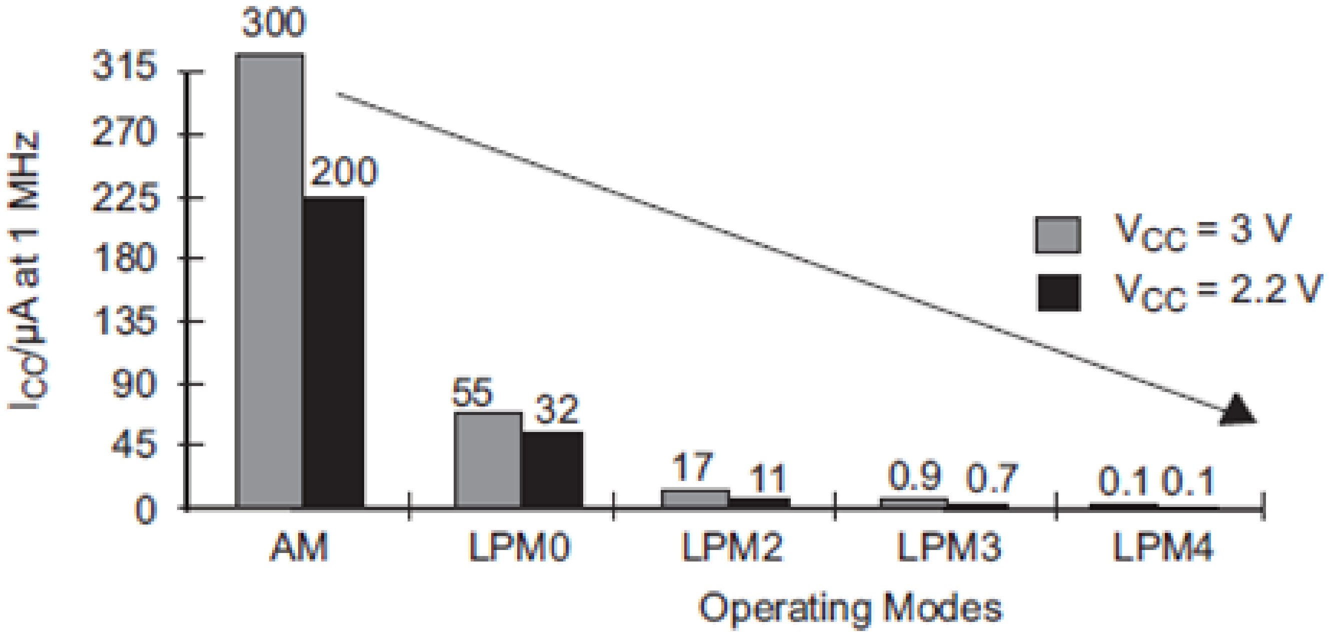
- In all Low Power Modes (LPM0-4) CPU is disabled
- Therefore, **only interrupts can be used to restart**

- LPM3 switches off everything bar the RTC
  - 1/300 power reduction
- LPM4 switches off all clocks, so interrupt can only come from an external source
  - 1/3000 power reduction

Low Power Modes

	CPU	MCLK	DCO	SMCLK	ACLK
LPM0	✗	✗	✓	✓	✓
LPM1	✗	✗	✗	✓	✓
LPM2	✗	✗	✗	✗	✓
LPM3	✗	✗	✗	✗	✓
LPM4	✗	✗	✗	✗	✗

Low Power Modes



Option 2: Interrupts

- Interrupts are good for
  - – infrequent, important events
  - – Precisely timed events (interrupts triggered by a timer)
  - – For processors with a rich range of interrupt priorities, when many different activities with different priorities have to be attended to
  - – Bonus on the MSP430 – allow the use of low power modes
- However:
  - – they are difficult to debug
  - – going to and returning from an interrupt service routine has a significant overhead. Polling may be better for short, very high priority events.

C Language Interface

- Interrupt routines specified by extended keyword '__interrupt'
- Interrupt vector specified by #pragma vector=<whatever> in previous line
- Enable interrupts with __enable_interrupt()
- Disable with __disable_interrupt()
- These are intrinsic functions so need to use #include <intrinsics.h>

Interrupts on the MSP430

```
#define PORT2_VECTOR        (46 * 2u) /* 0xFFE4 Port 2 */
#define PORT1_VECTOR        (47 * 2u) /* 0xFFE6 Port 1 */
#define ADC_VECTOR          (48 * 2u) /* 0xFFE8 ADC */
#define USCI_B0_VECTOR      (49 * 2u) /* 0xFFEA USCI B0 Receive/Transmit */
#define USCI_A0_VECTOR      (50 * 2u) /* 0xFFEC USCI A0 Receive/Transmit */
#define WDT_VECTOR          (51 * 2u) /* 0xFFEE Watchdog Timer */
#define RTC_VECTOR          (52 * 2u) /* 0xFFF0 RTC */
#define TIMER1_A1_VECTOR    (53 * 2u) /* 0xFFF2 Timer1_A3 CC1-2, TA */
#define TIMER1_A0_VECTOR    (54 * 2u) /* 0xFFF4 Timer1_A3 CC0 */
#define TIMER0_A1_VECTOR    (55 * 2u) /* 0xFFF6 Timer0_A3 CC1-2, TA */
#define TIMER0_A0_VECTOR    (56 * 2u) /* 0xFFE8 Timer0_A3 CC0 */
#define UNMI_VECTOR         (57 * 2u) /* 0xFFFA User Non-maskable */
#define SYSNMI_VECTOR       (58 * 2u) /* 0xFFFC System Non-maskable */
#define RESET_VECTOR        (59 * 2u) /* 0xFFFE Reset [Highest Priority] */
```

Interrupts on the MSP430

- Very many interrupt sources for each vector
- Either
  - Only enable one interrupt per vector
- Or
  - Check actual source as first step of interrupt service routine

- Remember to clear the interrupt request in the routine!
  - **MIGHT** be done automatically, but check in the data sheet

Watchdog

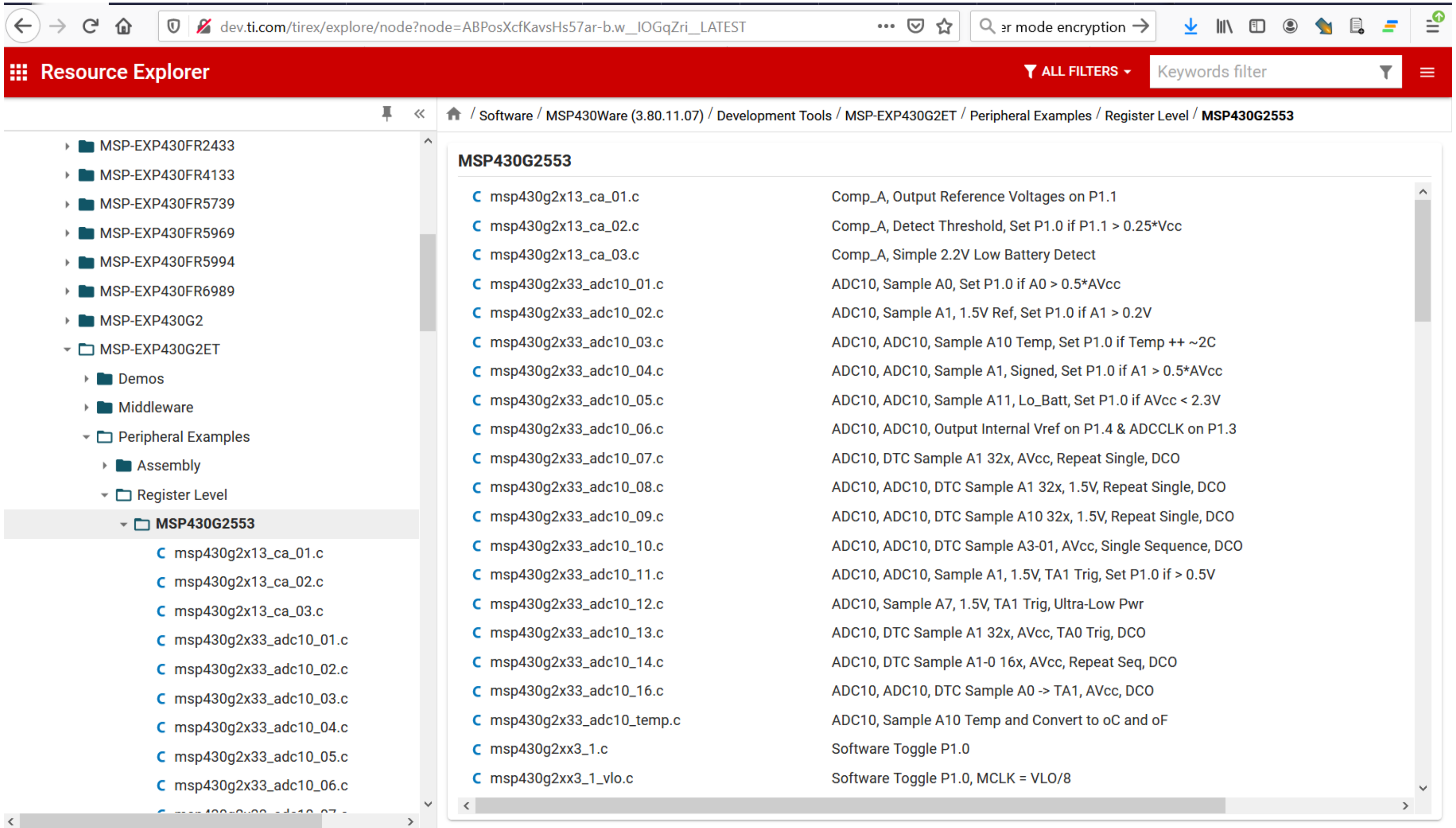
- MSP430s include a watchdog
- Hardware clock, causes interrupt (reset) when times out
- Resets the chip when code fails
  - Your code must regularly update the watchdog
- Can be used as an event timer
  - Generates an interrupt or sets flag on timeout

Watchdog

- Controlled by the WDTCTL register
  - ‘Password protected’ – not writing WDTPW to the high 8 bits will cause a reset
  - Avoids runaway code rewriting the watchdog control
- Clock sources
  - ACLK & SMCLK
  - Also VLOCLK on MSP430FR4133
- **Defaults on**
  - Uses power, so may not use in battery devices, BUT
    - Then you have no reset option if something goes wrong
  - Turn off by including the following code early in program

```
WDTCTL = WDTPW + WDTHOLD;
```

TI Resources – Link on MyPlace



MSP430G2553

msp430g2x13_ca_01.c	Comp_A, Output Reference Voltages on P1.1
msp430g2x13_ca_02.c	Comp_A, Detect Threshold, Set P1.0 if P1.1 > 0.25*Vcc
msp430g2x13_ca_03.c	Comp_A, Simple 2.2V Low Battery Detect
msp430g2x33_adc10_01.c	ADC10, Sample A0, Set P1.0 if A0 > 0.5*AVcc
msp430g2x33_adc10_02.c	ADC10, Sample A1, 1.5V Ref, Set P1.0 if A1 > 0.2V
msp430g2x33_adc10_03.c	ADC10, ADC10, Sample A10 Temp, Set P1.0 if Temp ++ ~2C
msp430g2x33_adc10_04.c	ADC10, ADC10, Sample A1, Signed, Set P1.0 if A1 > 0.5*AVcc
msp430g2x33_adc10_05.c	ADC10, ADC10, Sample A11, Lo_Batt, Set P1.0 if AVcc < 2.3V
msp430g2x33_adc10_06.c	ADC10, ADC10, Output Internal Vref on P1.4 & ADCCLK on P1.3
msp430g2x33_adc10_07.c	ADC10, DTC Sample A1 32x, AVcc, Repeat Single, DCO
msp430g2x33_adc10_08.c	ADC10, ADC10, DTC Sample A1 32x, 1.5V, Repeat Single, DCO
msp430g2x33_adc10_09.c	ADC10, ADC10, DTC Sample A10 32x, 1.5V, Repeat Single, DCO
msp430g2x33_adc10_10.c	ADC10, ADC10, DTC Sample A3-01, AVcc, Single Sequence, DCO
msp430g2x33_adc10_11.c	ADC10, ADC10, Sample A1, 1.5V, TA1 Trig, Set P1.0 if > 0.5V
msp430g2x33_adc10_12.c	ADC10, Sample A7, 1.5V, TA1 Trig, Ultra-Low Pwr
msp430g2x33_adc10_13.c	ADC10, DTC Sample A1 32x, AVcc, TA0 Trig, DCO
msp430g2x33_adc10_14.c	ADC10, DTC Sample A1-0 16x, AVcc, Repeat Seq, DCO
msp430g2x33_adc10_16.c	ADC10, ADC10, DTC Sample A0 -> TA1, AVcc, DCO
msp430g2x33_adc10_temp.c	ADC10, Sample A10 Temp and Convert to oC and oF
msp430g2xx3_1.c	Software Toggle P1.0
msp430g2xx3_1_vlo.c	Software Toggle P1.0, MCLK = VLO/8

```
//  Description: Toggle P1.0 using software and the TA_0 ISR. Timer_A is
//  configured for up mode, thus the the timer overflows when TAR counts
//  to CCR0. In this example, CCR0 is loaded with 1000-1.
//  Toggle rate = 32768/(2*1000) = 16.384Hz
//  ACLK = TACLK = 32768Hz, MCLK = SMCLK = DCO
//  //* An external watch crystal on XIN XOUT is required for ACLK *//
//
//           MSP430G2xx3
//         ---------------
//     /|\|            XIN|-
//      | |               | 32kHz
//      --|RST        XOUT|-
//        |               |
//        |           P1.0|-->LED
//
//  D. Dang
//  Texas Instruments Inc.
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************

#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  CCTL0 = CCIE;                             // CCR0 interrupt enabled
  CCR0 = 1000-1;
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) Timer_A (void)
#else
#error Compiler not supported!
#endif
{
  P1OUT ^= 0x01;                            // Toggle P1.0
}
```

```
//  Description: Toggle P1.0 using software and the TA_0 ISR. Timer_A is
//  configured for up mode, thus the the timer overflows when TAR counts
//  to CCR0. In this example, CCR0 is loaded with 1000-1.
//  Toggle rate = 32768/(2*1000) = 16.384Hz
//  ACLK = TACLK = 32768Hz, MCLK = SMCLK = DCO
//  //* An external watch crystal on XIN XOUT is required for ACLK *//
//
//           MSP430G2xx3
//         ---------------
//     /|\|            XIN|-
//      | |               | 32kHz
//      --|RST        XOUT|-
//        |               |
//        |           P1.0|-->LED
//
//  D. Dang
//  Texas Instruments Inc.
//  December 2010
//   Built with CCS Version 4.2.0 and IAR Embedded Workbench Version: 5.10
//******************************************************************************
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  CCTL0 = CCIE;                             // CCR0 interrupt enabled
  CCR0 = 1000-1;
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) Timer_A (void)
#else
#error Compiler not supported!
#endif
{
  P1OUT ^= 0x01;                            // Toggle P1.0
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  CCTL0 = CCIE;                             // CCR0 interrupt enabled
  CCR0 = 1000-1;
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) Timer_A (void)
#else
#error Compiler not supported!
#endif
{
  P1OUT ^= 0x01;                            // Toggle P1.0
}
```

```
#include <msp430.h>

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR |= 0x01;                            // P1.0 output
  CCTL0 = CCIE;                             // CCR0 interrupt enabled
  CCR0 = 1000-1;
  TACTL = TASSEL_1 + MC_1;                  // ACLK, upmode

  __bis_SR_register(LPM3_bits + GIE);       // Enter LPM3 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                            // Toggle P1.0
}
```

Advanced Warning of Week 5 Problem

- Write a program to flash an LED with a period of 1.5 seconds. Once a switch is pressed, the LED will go off, and then *exactly* thirty seconds later the LED will come on again and start flashing at 40 flashes/minute

University of Strathclyde Glasgow