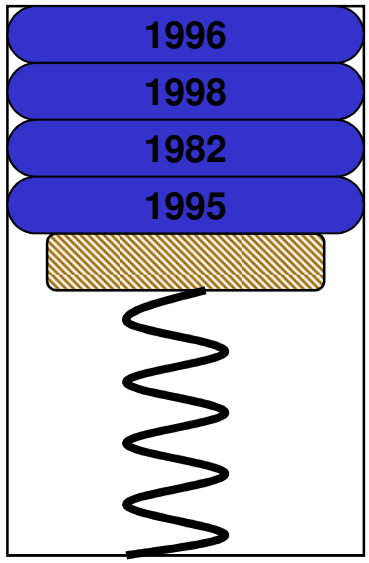
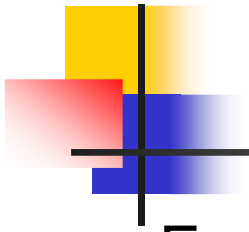
Chapter 9 – Interrupts



Topics to Cover…

- Interrupts
- Interrupt Service Routines (ISR's)
- Watchdog Timer
  - Example 9.4 – Watchdog ISR
- Energy Consumption
- Processor Clocks
  - Example 9.2 – Master Clock
- Low Power Modes
  - Example 9.3 – Low-Power Mode
- Timers
  - Example 9.4 – Interrupts w/Timer_A
- Pulse Width Modulation (PWM)
  - Example 9.5 – LED PWM w/Timer_A
- Speaker (Transducer)
  - Example 9.6 – Speaker PWM w/Watchdog

Interrupt Service Routines

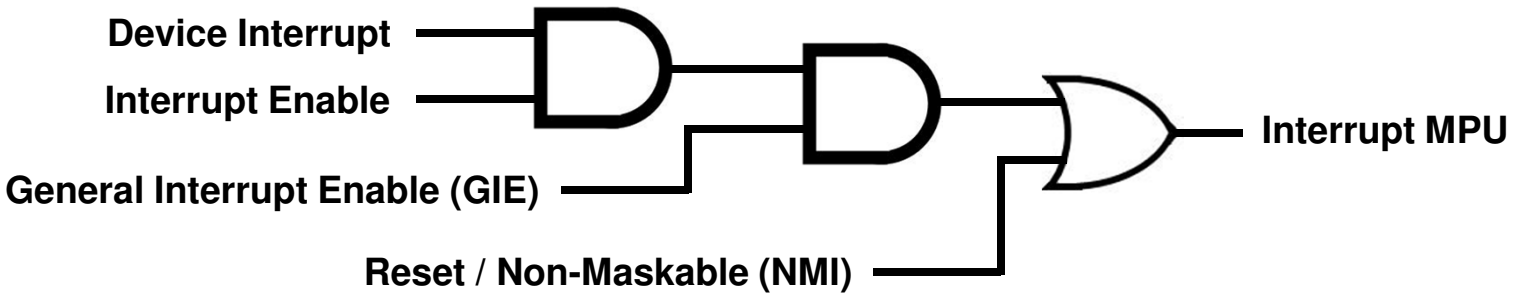


Interrupts

- Execution of a program normally proceeds predictably, with *interrupts* being the exception.
- An *interrupt* is an asynchronous signal indicating something needs attention.
  - Some event has occurred
  - Some event has completed
- The processing of an interrupt subroutine uses the stack.
  - Processor stops with it is doing,
  - stores enough information on the stack to later resume,
  - executes an *interrupt service routine* (ISR),
  - restores saved information from stack (**RETI**),
  - and then resumes execution at the point where the processor was executing before the interrupt.

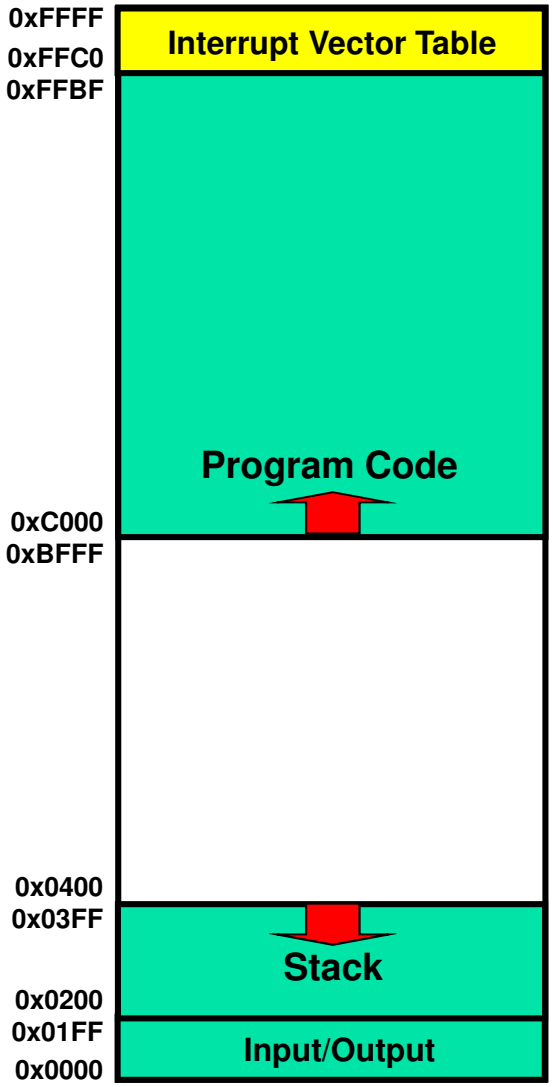
Interrupt Flags

- Each interrupt has a flag that is raised (set) when the interrupt is pending.
- Each interrupt flag has a corresponding enable bit – setting this bit allows a hardware module to request an interrupt.
- Most interrupts are ***maskable***, which means they can only interrupt if
  1) Individually enabled and
  2) general interrupt enable (GIE) bit is set in the status register (SR).
- Reset and Non-Maskable Interrupts (NMI) are reserved for system interrupts such as power-up (PUC), external reset, oscillator fault, illegal flash access, watchdog, and illegal instruction fetch.



Interrupt Vectors

- **The CPU must know where to fetch the next instruction following an interrupt.**
- The address of an ISR is defined in an *interrupt vector*.
- The MSP430 uses *vectored interrupts* where each ISR has its own vector stored in a *vector table* located at the end of program memory.
- Note: The *vector table* is at a fixed location (defined by the processor data sheet), but the ISRs can be located anywhere in memory.



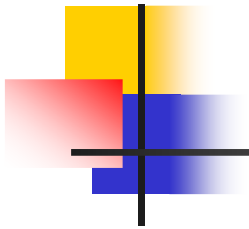
MSP430 Interrupt Vectors

INTERRUPT SOURCE	INTERRUPT FLAG		ADDRESS	SECTION	PRIORITY
Power-up External reset Watchdog	PORIFG RSTIFG WDTIFG		0xFFFE	.reset	15, highest
NMI Oscillator fault Flash memory violation	NMIIFG OFIFG ACCDVIFG	Non-maskable	0xFFFC	.int14	14
Timer_B3	TBCCR0 CCIFG		0xFFFA	.int13	13
Timer_B3	TBCCR1 CCIFG TBCCR2 CCIFG, TBIFG		0xFFF8	.int12	12
			0xFFF6	.int11	11
Watchdog Timer	WDTIFG	Maskable	0xFFF4	.int10	10
Timer_A3	TACCR0 CCIFG	Maskable	0xFFF2	.int09	9
Timer_A3	TACCR1 CCIFG, TACCR2 CCIFG, TAIFG	Maskable	0xFFF0	.int08	8
USCI_A0/USCI_B0 Rx	UCA0RXIFG, USB0RXIFG	Maskable	0xFFEE	.int07	7
USCI_Z0/USCI_B0 Tx	UCA0TXIFG, UCB0TXIFG	Maskable	0xFFEC	.int06	6
ADC10	ADC10IFG		0xFFEA	.int05	5
			0xFFE8	.int04	4
I/O Port P2	P2IFG.0 – P2IFG.7	Maskable	0xFFE6	.int03	3
I/O Port P1	P1IFG.0 – P1IFG.7	Maskable	0xFFE4	.int02	2
			0xFFE2	.int01	1
			0xFFE0	.int00	0

Non-Maskable Interrupts

Timers

Ports



Processing an Interrupt…

1. Processor completes execution of current instruction.
2. Master Clock (MCLK) started (if CPU was off).
3. Processor pushes Program Counter (PC) on stack.
4. Processor pushes Status Register (SR) on stack.
5. Interrupt w/highest priority is selected.
6. Interrupt request flag cleared (if single sourced).
7. Status Register is cleared:
   - Disables further maskable interrupts (GIE cleared)
   - Terminates low-power mode
8. Processor fetches interrupt vector and stores it in the program counter.
9. User ISR must do the rest!

Return From Interrupt

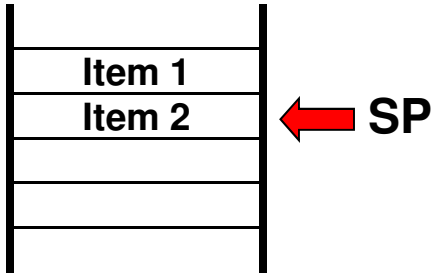
- Single operand instructions:

Mnemonic	Operation	Description
PUSH (.B or .W) src	SP-2→SP, src→@SP	Push byte/word source on stack
CALL dst	dst→tmp ,SP-2→SP, PC→@SP, tmp→PC	Subroutine call to destination
RETI	TOS→SR, SP+2→SP TOS→PC, SP+2→SP	Return from interrupt

- Emulated instructions:

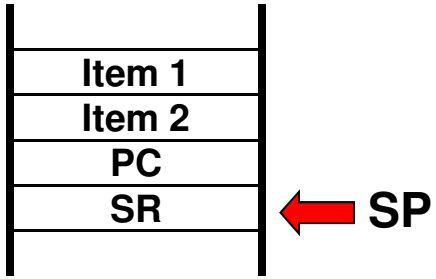
Mnemonic	Operation	Emulation	Description
RET	@SP→PC SP+2→SP	MOV @SP+,PC	Return from subroutine
POP (.B or .W) dst	@SP→temp SP+2→SP temp→dst	MOV(.B or .W) @SP+,dst	Pop byte/word from stack to destination

Interrupt Stack



Prior to Interrupt

```
        add.w   r4,r7
PC →      jnc   $+4
        add.w   #1,r6

        add.w   r5,r6
```

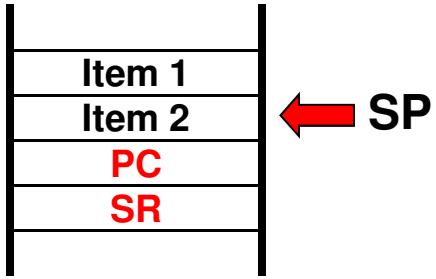


Interrupt (hardware)

- **P**rogram **C**ounter pushed on stack
- **S**tatus **R**egister pushed on stack
- Interrupt vector moved to **PC**
- Further interrupts disabled
- Interrupt flag cleared

```
PC →    xor.b   #1,&P1OUT
        reti
```

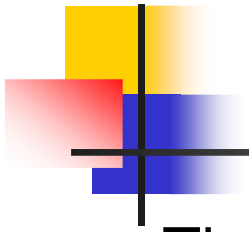
Execute Interrupt Service Routine (ISR)



Return from Interrupt (**reti**)

- **S**tatus **R**egister popped from stack
- **P**rogram **C**ounter popped from stack

```
        add.w   r4,r7
PC →      jnc   $+4
        add.w   #1,r6

        add.w   r5,r6
```



Interrupt Latency

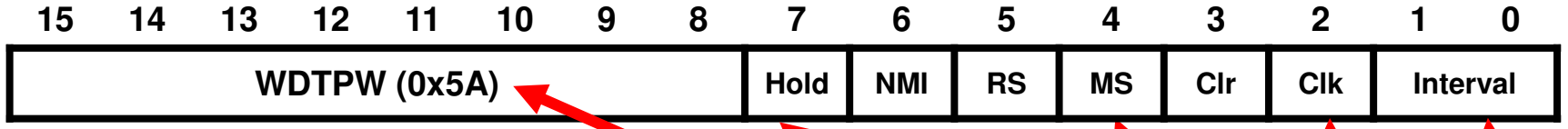
- The time between the interrupt request and the start of the ISR is called latency
  - MSP430 requires 6 clock cycles before the ISR begins executing
  - An ISR may be interrupted if interrupts are enabled in the ISR
- Well-written ISRs:
  - Should be *short* and *fast – get in and get out*
  - Require a *balance* between doing very little – thereby leaving the background code with lots of processing – and doing a lot and leaving the background code with nothing to do
- Applications that use interrupts should:
  - Disable interrupts *as little as possible*
  - *Respond to interrupts* as quickly as possible
  - Communicate w/ISR only through global variables (never through registers!!!)

Watchdog

Watchdog Timer

- The MSP430 watchdog can be configured as a COP (computer operating properly) device or as a timer.
- The primary function of the watchdog timer (WDT+) module is to perform a controlled system restart after a software problem occurs.
  - After a power-up cycle (PUC), the WDT+ module is automatically configured in watchdog mode with an initial 32768 clock cycle reset interval using the DCOCLK.
  - The user must setup or halt the WDT+ prior to the expiration of the initial reset interval, else an unmasked system reset is generated.
- If the watchdog function is not needed in an application, the module can be configured as an interval timer and can generate interrupts at selected time intervals.

WDT+ Control Register (WDTCTL)



15-8	WDTPW	WDT+ password:	0x69 ⇒ Read 0x5A ⇒ Write o[illegible] PUC is generated
7	WDTHOLD	WDT+ hold:	0 ⇒ Enabled 1 ⇒ Stopped
6	WDTNMIES	WDT+ NMI edge select:	0 ⇒ NMI on rising edge 1 ⇒ NMI on falling edge
5	WDTNMI	WDT+ NMI select:	0 ⇒ Reset function 1 ⇒ NMI function
4	WDTTMSEL	WDT+ mode select:	0 ⇒ Watchdog mode 1 ⇒ Interval timer mode
3	WDTCNTCL	WDT+ counter clear:	0 ⇒ No action 1 ⇒ WDTCNT = 0x0000
2	WDTSSEL	WDT+ clock source select:	0 ⇒ SMCLK 1 ⇒ ACLK
1-0	WDTISx	WDT+ interval select:	0 ⇒ WD clock source / 32768 1 ⇒ WD clock source / 8192 2 ⇒ WD clock source / 512 3 ⇒ WD clock source / 64

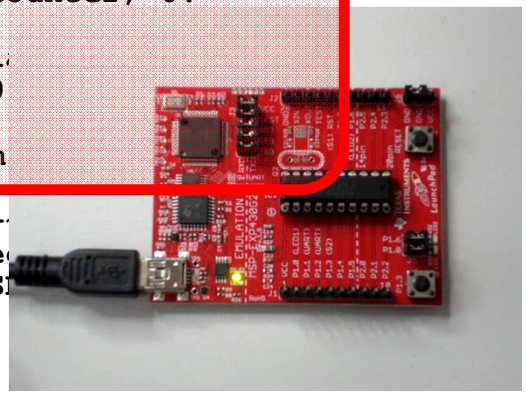
Example 9.1 – Watchdog ISR

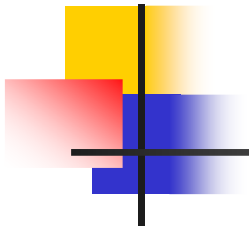
```
            .cdecls C,"msp430.h"
SMCLK       .equ    1200000                 ; 1.2 Mhz clock
WDT_CTL     .equ    WDT_MDLY_8              ; WDT SMCLK, 8 ms (@1 Mhz)
WDT_CPS     .equ    SMCLK/8000              ; WD clocks / second count

; Data Section ------------------------------------------------------------
            .bss    WDTSecCnt,2             ; WDT second c
; Code Section ------------------------------------------------------------
            .text
start:      mov.w   #0x400,SP               ; initialize stack pointer
            mov.w   #WDT_CTL,&WDTCTL        ; set WD timer interval
            mov.w   #WDT_CPS,&WDTSecCnt     ; initialize 1 sec WD counter
            bis.b   #WDTIE,&IE1             ; enable WDT interrupt
            bis.b   #0x01,&P1DIR            ; P1.0 output
            bis.w   #GIE,SR                 ; enable interrupt

loop:       ;<< program >>
            jmp     loop                    ; loop indefinite

; Watchdog ISR ------------------------------------------------------------
WDT_ISR:    dec.w   &WDTSecCnt              ; decrement counter, 0?
              jne   WDT_02                  ; n
            mov.w   #WDT_CPS,&WDTSecCnt     ; y, re-initi
            xor.b   #0x01,&P1OUT            ; toggle P1.0

WDT_02:     reti                            ; return from

; Interrupt Vectors -------------------------------------------------------
            .sect   ".int10"                ; Watchdog Ve
            .word   WDT_ISR                 ; Watchdog IS
            .sect   ".reset"                ; PUC Vector
            .word   start                   ; RESET ISR
```

8 ms (@1 MHz SMCLK)
WDT_CPS = clocks/second

Configure Watchdog as a timer and enable it to interrupt

Watchdog Interrupt Service Routine





Quiz 9.1

1. What conditions must be met before a device can interrupt the computer?

2. What is saved on the stack when an interrupt occurs?

3. Where are interrupt vectors located in memory? ISRs?

4. (T or F) Interrupts are predictable asynchronous events.

Energy Consumption

U.S. Energy Consumption

- The United States is the 2nd largest energy consumer in terms of total use in 2010[1] in the world.
- Not getting better – everyone must do their part.
- How could I lower home energy consumption?
  - Turn off lights
  - Turn down thermostat
  - Unplug appliances not in use
  - Use slower motor speeds
  - Heat/cool at a slower rate
  - Improve insulation
  - Purchase energy efficient appliances
  - Eliminate unnecessary living space
- How could I make computers more energy efficient?

1. Barr, Robert. "China surpasses US as top energy consumer". Associated Press. Retrieved 16 June 2012.

Computer Energy Consumption

- Computers occupy a small but growing percentage of annual U.S. electricity consumption.
  - Estimates vary for 3% to 13% of entire U.S. supply.
  - Hidden costs – every 100 watts consumed by computers requires 50 watts of cooling.
  - Computers are ubiquitous – found in every energy sector.
- Contemporary processors require more electricity than their predecessors.
- Server farms, supercomputers, web hosting, scientific simulations, 3D rendering, search engines,…
  - Google uses enough energy to continuously power 200,000 homes (260 million watts – ¼ output of a nuclear power plant).
  - One Google search is equal to turning on a 60W light bulb for 17 seconds.

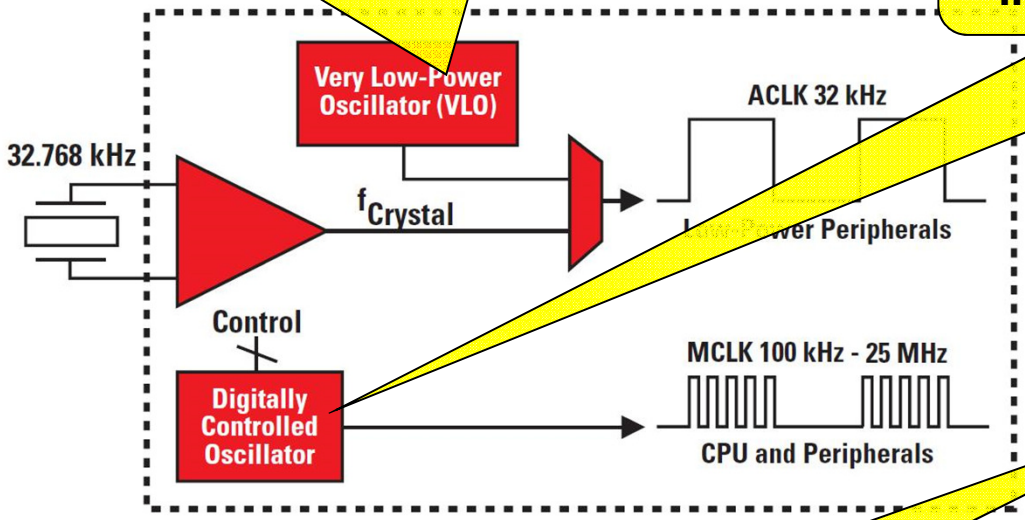
Clocks

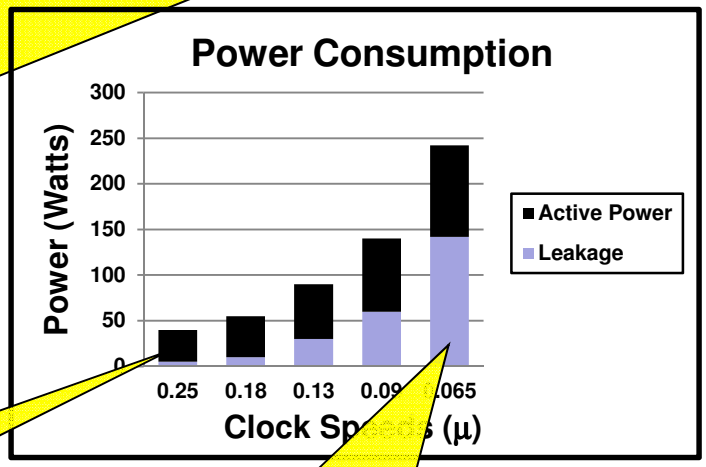
Processor Clock Speeds

- Often, the most important factor for reducing power consumption is slowing the clock down.
  - Faster clock = Higher performance, more power required
  - Slower clock = Lower performance, less power required

```
; Set ACLK to ≈12kHz
mov.w #LFXT1S_2,&BCSCTL3
```

```
;  Set DCO to 8 MHz:
   mov.b   #CALBC1_8MHZ,&BCSCTL1
   mov.b   #CALDCO_8MHZ,&DCOCTL
```





10% loss of power

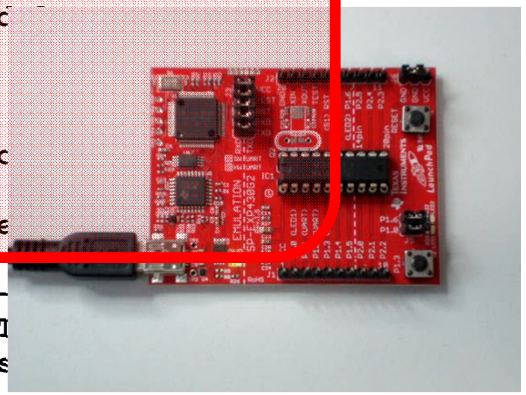
60% loss of power

Example 9.2 – Clock Speed

```
            .cdecls C,"msp430.h"
SMCLK       .equ    1200000                 ; 1.2 Mhz clock
WDT_CTL     .equ    WDT_MDLY_8              ; WDT SMCLK, 8 ms (@1 MHz)
WDT_CPS     .equ    SMCLK/8000              ; WDT clocks / s
MHz8        .set    1                       ; 8 MHz flag

; Code Section -------------------------------------------
            .text
start:      mov.w   #0x0400,SP              ; init stack p
            mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; stop WDT
            .if     MHz8                    ; set DCO to 8
            mov.b   #CALBC1_8MHZ,&BCSCTL1   ; set range
            mov.b   #CALDCO_8MHZ,&DCOCTL    ; set DCO step
            .endif
            bis.b   #0x01,&P1DIR            ; set P1.0 as

mainloop:   xor.b   #0x01,&P1OUT            ; toggle P1.0
            mov.w   #8,r14                  ; use R14 as c

delaylp1:   mov.w   #0,r15                  ; use R15 as c

delaylp2:   dec.w   r15                     ; delay over?
              jne   delaylp2                ; n
            dec.w   r14                     ; y, outer loo
              jne   delaylp1                ; n
            jmp     mainloop                ; y, toggle le

; Interrupt Vectors --------------------------------------
            .sect   ".reset"                ; MSP430 RESET
            .word   start                   ; start addres
            .end
```

Conditionally assemble setting new DCO constants (for 8 MHz)

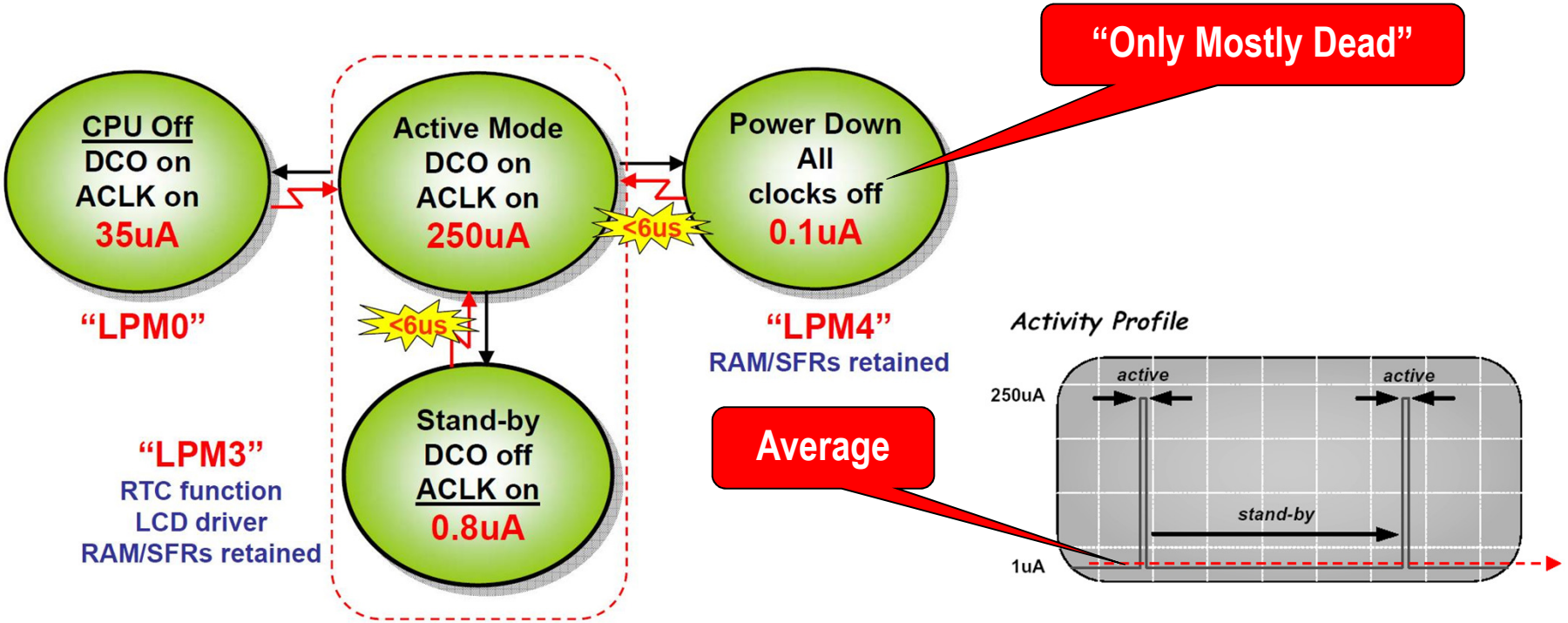
Nested delay loop to blink LED



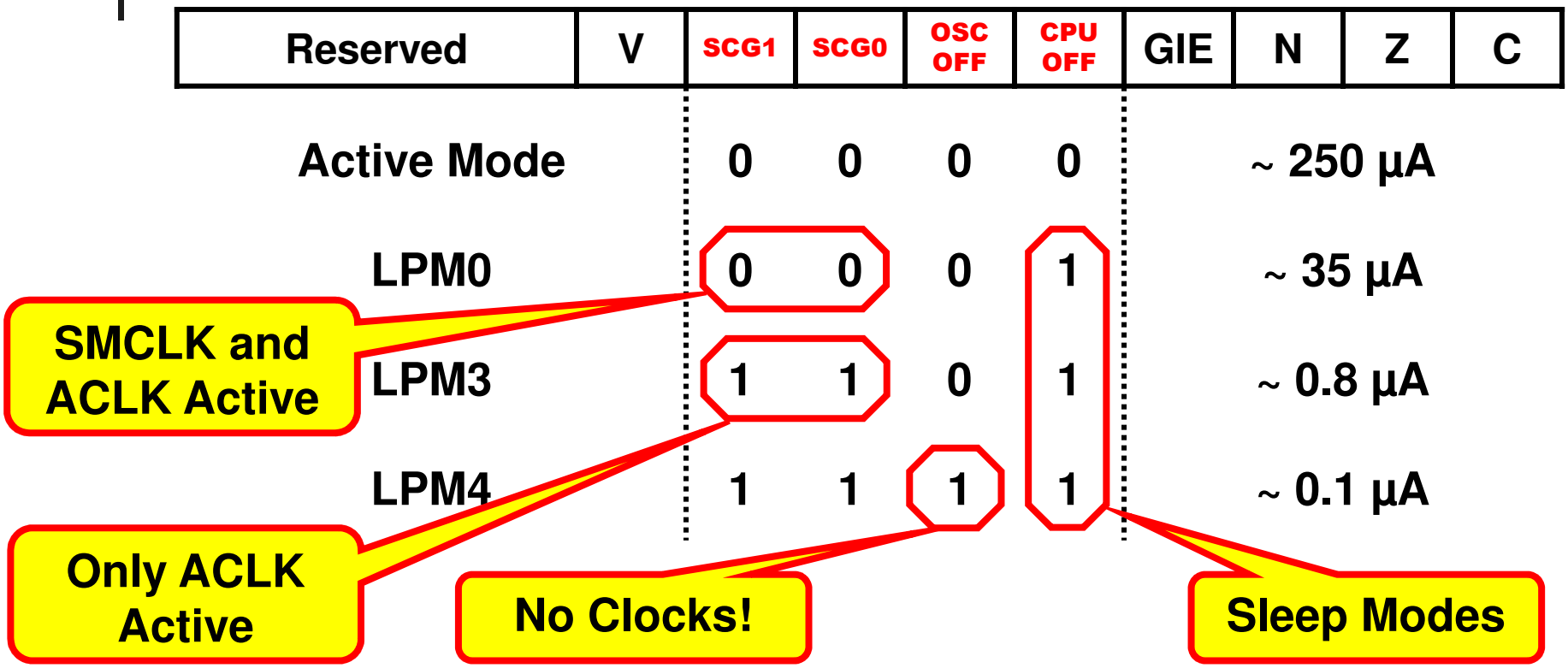
Low-Power Mode

MSP430 Clock Modes

- Another method to reduce power consumption is to turn off some (or all) of the system clocks.
- A device is said to be *sleeping* when in low-power mode; *waking* refers to returning to active mode.



MSP430 Clock Settings



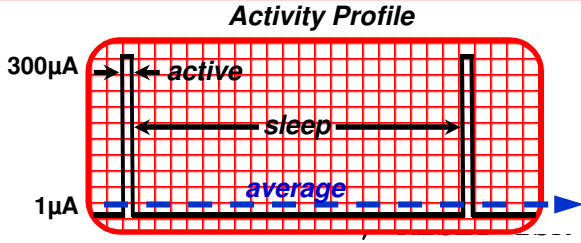

```
;   enable interrupts / enter low-power mode 0
    bis.w   #LPM0+GIE,SR        ; LPM0 w/interrupts
```

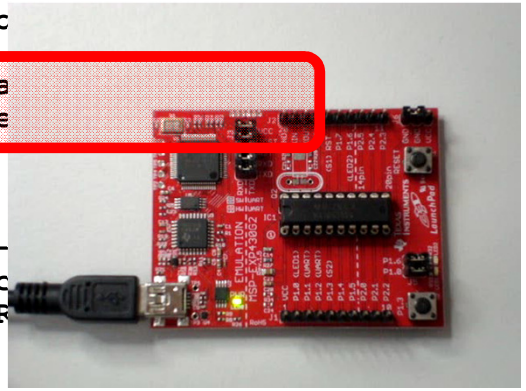
Example 9.3 – Low Power

```
            .cdecls C,"msp430.h"
SMCLK       .equ    1200000                 ; 1.2 Mhz clock
WDT_CTL     .equ    WDT_MDLY_8              ; WDT SMCLK, 8 ms (@1 Mhz)
WDT_CPS     .equ    SMCLK/8000              ; WDT clocks / second count
STACK       .equ    0x0400                  ; top of stack

; Data Section ------------------------------------------------------------
            .bss    WDTSecCnt,2             ; WDT second counter
; Code Section ------------------------------------------
            .text
start:      mov.w   #STACK,SP               ; initi
            mov.w   #WDT_CTL,&WDTCTL        ; set WD
            mov.w   #WDT_CPS,&WDTSecCnt     ; initiali  WD counter
            bis.b   #WDTIE,&IE1             ; enab  WDT interrupt
            bis.b   #0x01,&P1DIR            ; P1.0 output

loop:       bis.w   #LPM0|GIE,SR            ; sleep/enable interrupts
            xor.b   #0x01,&P1OUT            ; toggle P1.0
            jmp     loop                    ; loop indefinitely

; Watchdog ISR ------------------------------------------------------------
WDT_ISR:    dec.w   &WDTSecCnt              ; decrement cc
              jne   WDT_02                  ; n
            mov.w   #WDT_CPS,&WDTSecCnt     ; y, re-initia
            bic.b   #LPM0,0(SP)             ; wakeup proce

WDT_02:     reti                            ;

            .end
```

1. Enable interrupts
2. Goto Sleep (Low-power Mode 0)
3. Blink LED when awakened

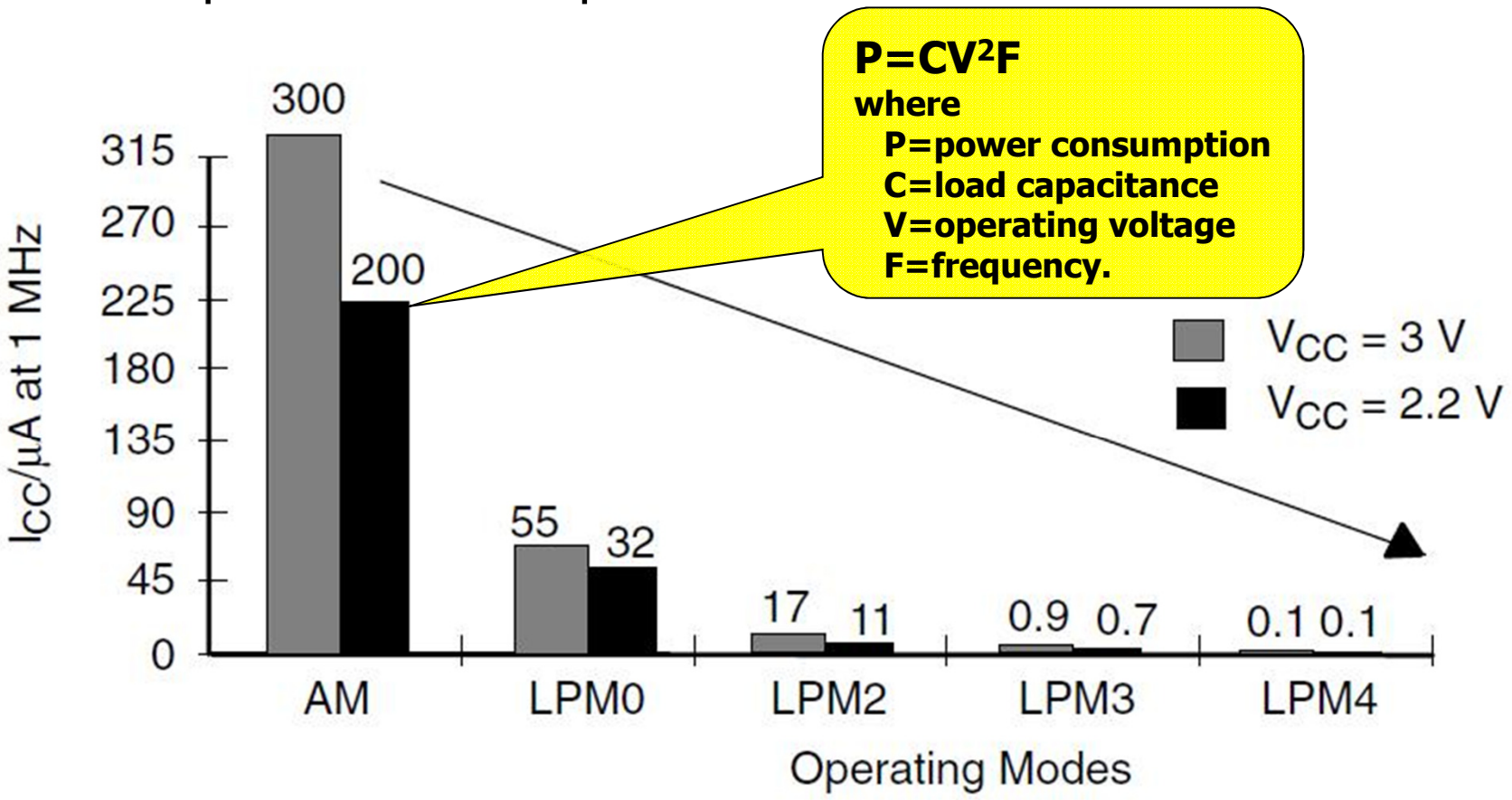
1. Reset counter every second
2. Set Active Mode in saved SR
3. Wakeup processor on RETI





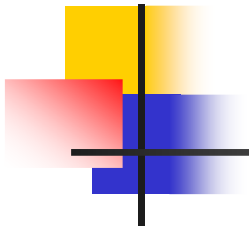
Lower Power Savings

- Finally, powering your system with lower voltages means lower power consumption as well.



Principles of Low-Power Apps

- Maximize the time in low-power modes.
- Sleep as long as possible and use interrupts to wakeup.
- Use the slowest clock while still meeting processing needs.
- Switch on peripherals only when needed (ie, switch off peripherals when not needed).
- Use low-power integrated computer peripherals.
  - Timers: Timer_A and Timer_B for PWM
  - A/D convertors, flash, LCD's
- Use faster software algorithms / programming techniques
  - Calculated branches instead of flag polling.
  - Fast table look-ups instead of iterative calculations.
  - Use in-line code instead of frequent subroutine / function calls.
  - More single-cycle CPU register usage.



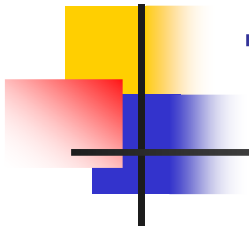
Quiz 9.2

1. Why is low-power usage an important issue?

2. What power mode is used by ISRs?

3. Name 3 ways to reduce power consumption.
   1.
   2.
   3.
4. Approximately what percentage less power is consumed in a given time period by sleeping?

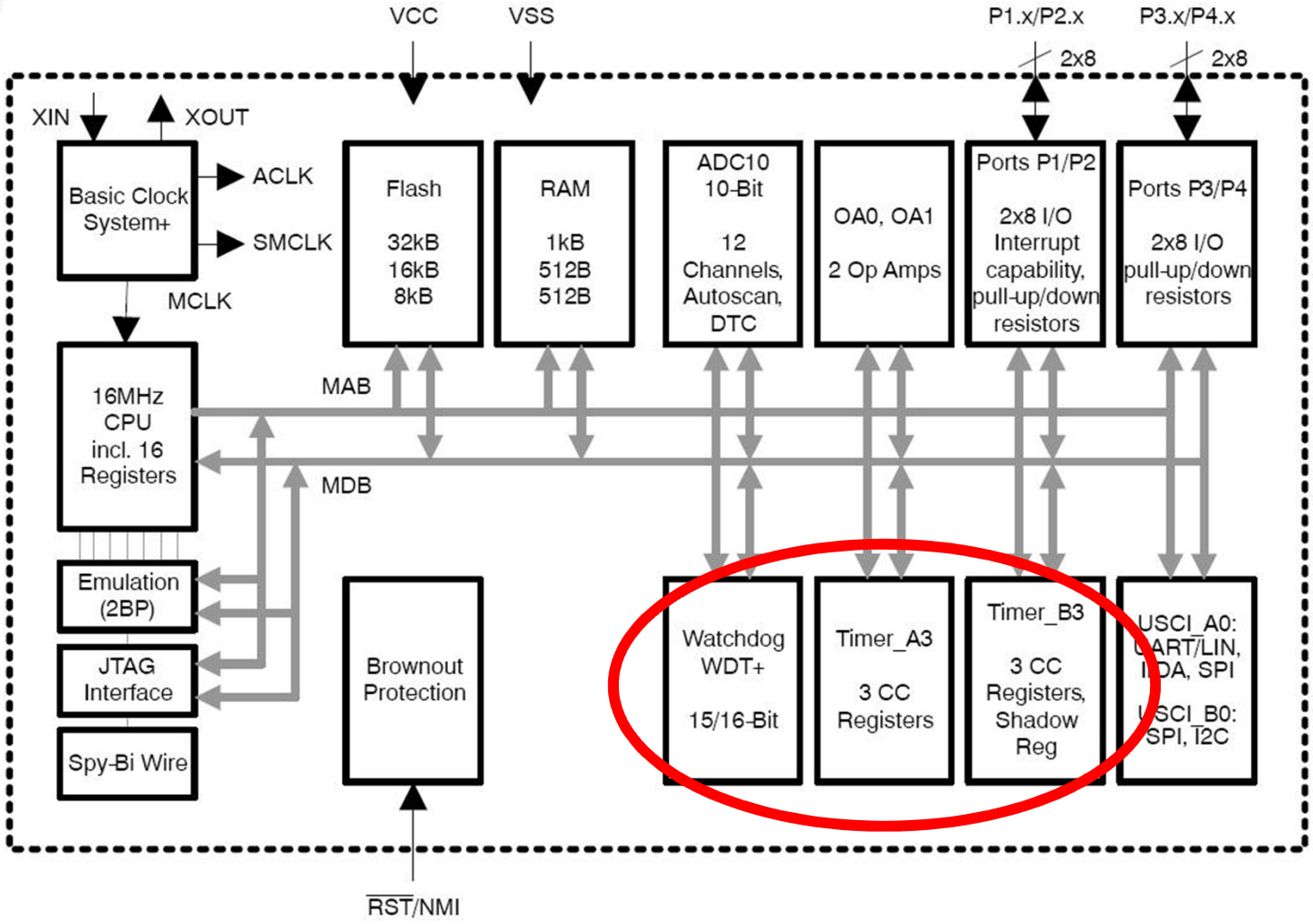
Timers



Timers

- System timing is fundamental for real-time applications
- The main applications of timers are to:
  - generate events of fixed time-period
  - allow periodic wakeup from sleep
  - count transitional signal edges
  - replace delay loops allowing the CPU to sleep between operations, consuming less power
  - maintain synchronization clocks
  - debounce mechanical devices
  - real-time clocks
  - control simulations
  - measure rates
  - Pulse Width Modulation

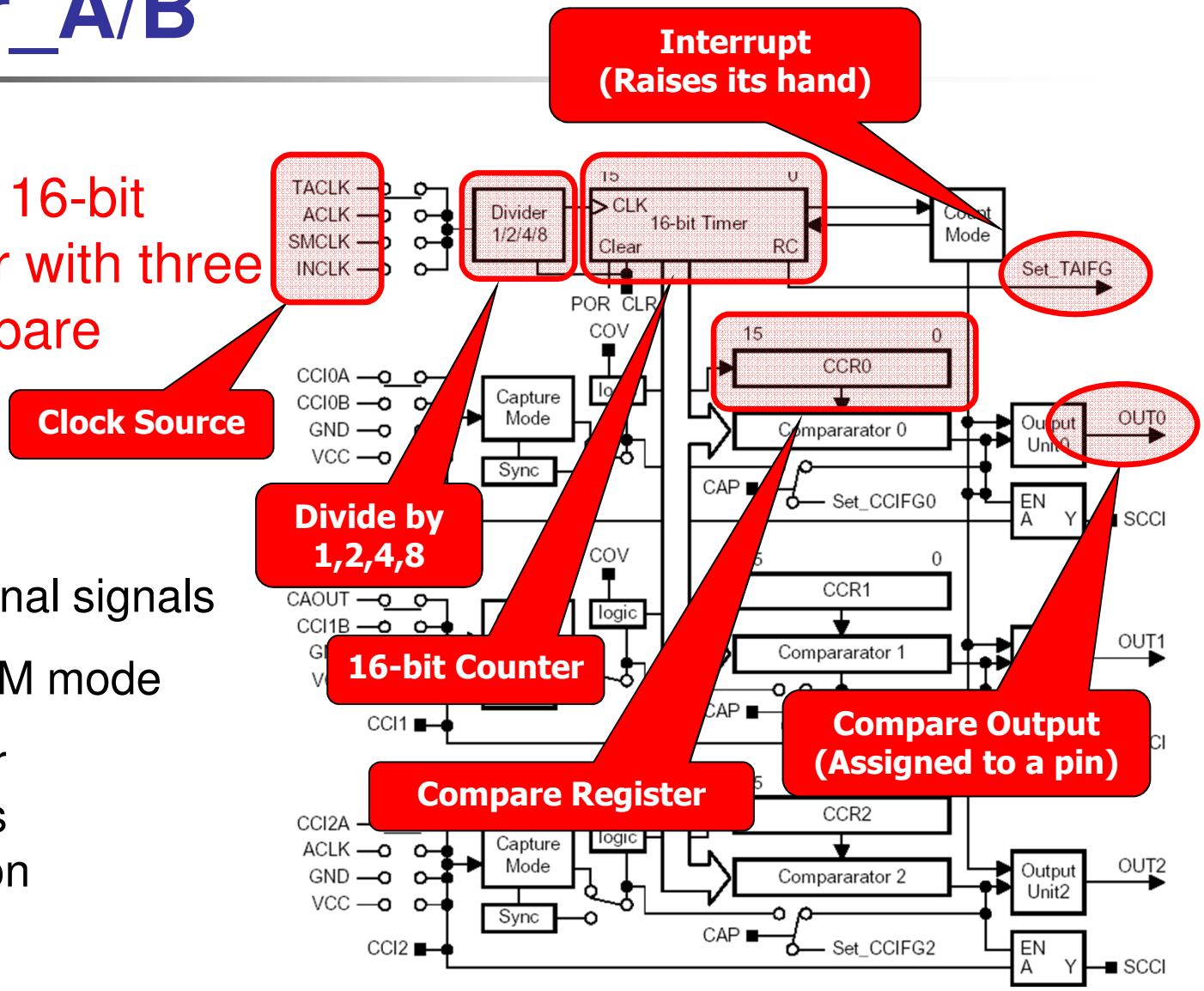
Timers


VCC
VSS
P1.x/P2.x
2x8
P3.x/P4.x
2x8
XIN
XOUT
Basic Clock System+
ACLK
SMCLK
MCLK
Flash
32kB
16kB
8kB
RAM
1kB
512B
512B
ADC10
10-Bit
12
Channels,
Autoscan,
DTC
OA0, OA1
2 Op Amps
Ports P1/P2
2x8 I/O
Interrupt
capability,
pull-up/down
resistors
Ports P3/P4
2x8 I/O
pull-up/down
resistors
16MHz
CPU
incl. 16
Registers
MAB
MDB
Emulation
(2BP)
JTAG
Interface
Spy-Bi Wire
Brownout
Protection
Watchdog
WDT+
15/16-Bit
Timer_A3
3 CC
Registers
Timer_B3
3 CC
Registers,
Shadow
Reg
USCI_A0:
UART/LIN,
IrDA, SPI
USCI_B0:
SPI, I2C
RST/NMI

Timer_A/B

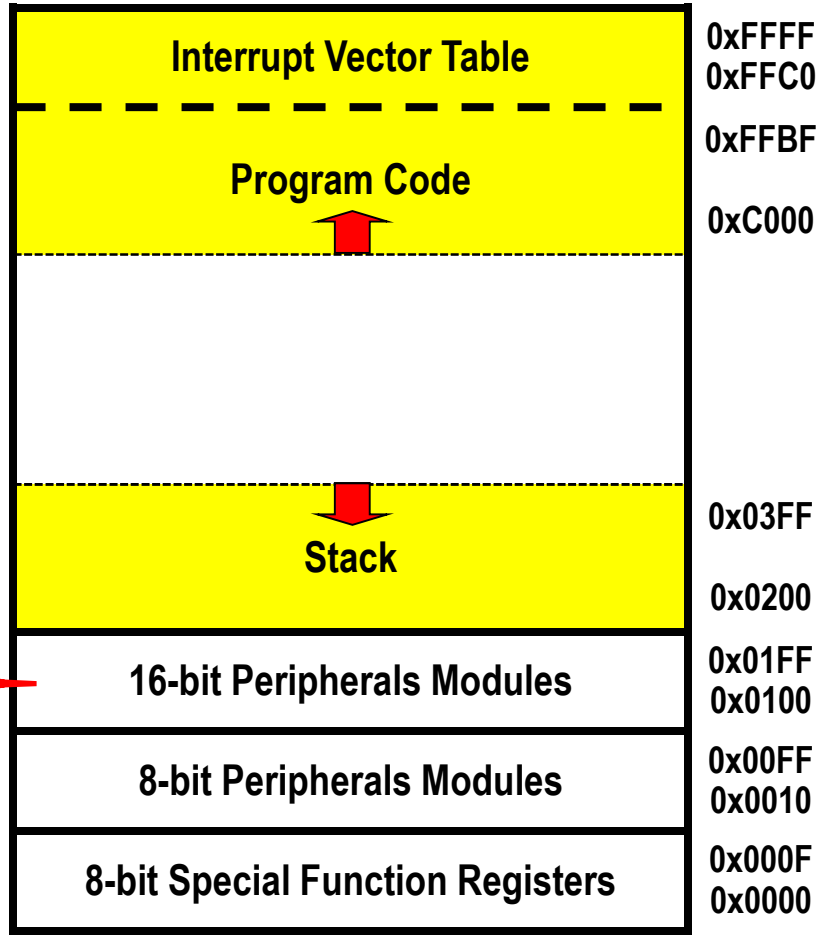
Timer_A is a 16-bit timer/counter with three capture/compare registers

- Capture external signals
- Compare PWM mode
- SCCI latch for asynchronous communication



Timer Registers

	Description		
Timer_B (Not available in F2013)	Capture/compare register	TBCCR2	0x0196
	Capture/compare register	TBCCR1	0x0194
	Capture/compare register	TBCCR0	0x0192
	Timer_B register	TBR	0x0190
	Capture/compare control	TBCCTL2	0x0186
	Capture/compare control	TBCCTL1	0x0184
	Capture/compare control	TBCCTL0	0x0182
	Timer_B control	TBCTL	0x0180
	Timer_B interrupt vector	TBIV	0x011E
Timer_A	Capture/compare register	TACCR2	0x0176
	Capture/compare register	TACCR1	0x0174
	Capture/compare register	TACCR0	0x0172
	Timer_A register	TAR	0x0170
	Capture/compare control	TACCTL2	0x0166
	Capture/compare control	TACCTL1	0x0164
	Capture/compare control	TACCTL0	0x0162
	Timer_A control	TACTL	0x0160
	Timer_A interrupt vector	TAIV	0x012E
Watchdog Timer+	WD Control	WDTCTL	0x0120



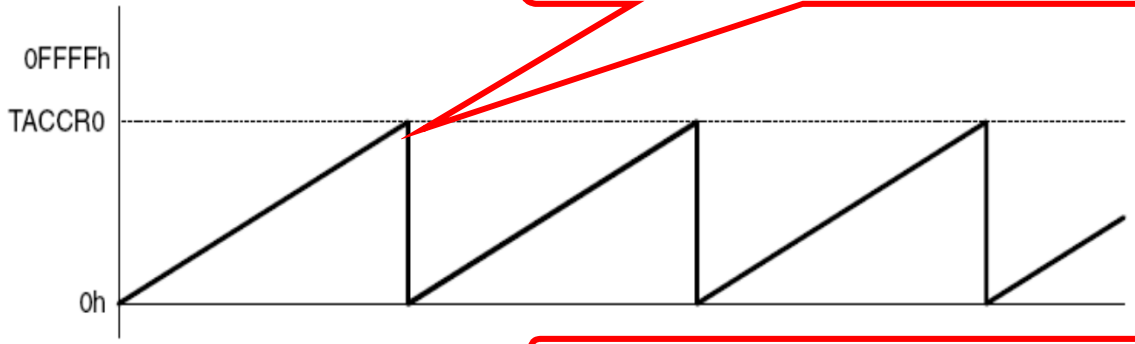
TxCTL Control Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
(Used only by Timer_B)						TxSSELx		IDx		MCx		-	TxCLR	TxIE	TxIFG

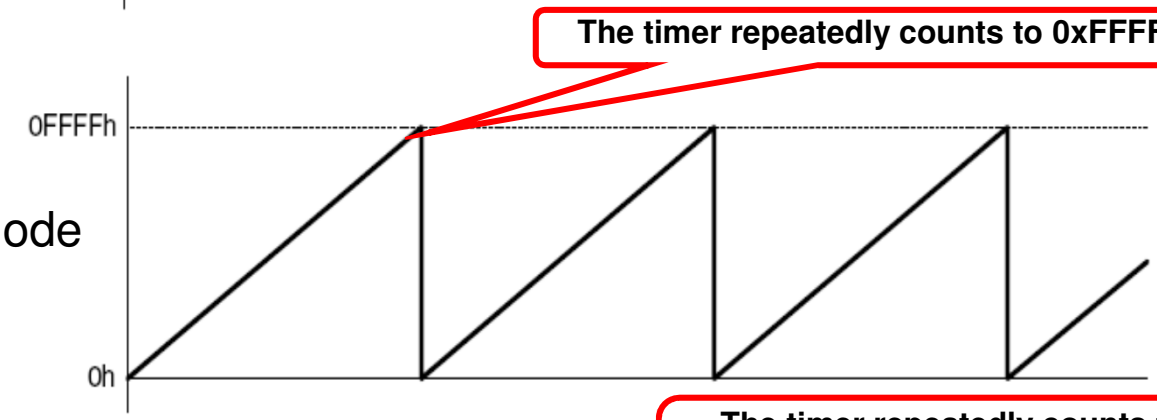
Bit		Description	
9-8	TxSSELx	Timer_x clock source:	0 0 ⇒ TxCLK 0 1 ⇒ ACLK 1 0 ⇒ SMCLK 1 1 ⇒ INCLK
7-6	IDx	Clock signal divider:	0 0 ⇒ / 1 0 1 ⇒ / 2 1 0 ⇒ / 4 1 1 ⇒ / 8
5-4	MCx	Clock timer operating mode:	0 0 ⇒ Stop mode 0 1 ⇒ Up mode 1 0 ⇒ Continuous mode 1 1 ⇒ Up/down mode
2	TxCLR	Timer_x clear when TxCLR = 1	
1	TxIE	Timer_x interrupt enable when TxIE = 1	
0	TxIFG	Timer_x interrupt pending when TxIFG = 1	

Timer Modes

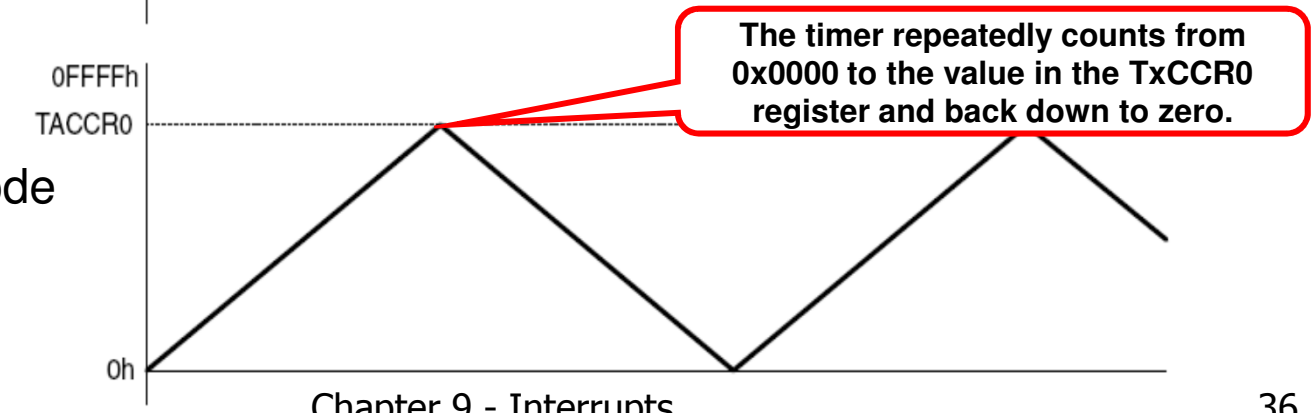
- 01 = Up Mode



- 10 = Continuous Mode



- 11 = Up/Down Mode



Example 9.4 – Timer_A

```
                .cdecls C,"msp430.h"            ; MSP430
TA_CTL          .equ    TASSEL_2|ID_3|MC_1|TAIE ; 000000 10 11 01 000 1 = SMCLK,/8,UP,IE
TA_FREQ         .equ    0xffff                  ; clocks
STACK           .equ    0x0400

; Code Section -------------------------------------------------------------
                .text                           ; beginning of executable code
start:          mov.w   #STACK,SP               ; init stack pointer
                mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; stop WDT
                bis.b   #0x01,&P1DIR            ; set P1.0 as output

                clr.w   &TAR                    ; reset timerA
                mov.w   #TA_CTL,&TACTL          ; set timerA control reg
                mov.w   #TA_FREQ,&TACCR0        ; set interval (frequency)
                bis.w   #LPM0|GIE,SR            ; enter LPM0 w/interrupts

error:          jmp     $                       ; SHOULD NEVER GET HERE!!!!!!!! !!!

; Timer A ISR --------------------------------------------------
TA_isr:         bic.w   #TAIFG,&TACTL           ; acknowledge inte
                xor.b   #0x01,&P1OUT            ; toggle P1.0
                reti

; Interrupt Vectors --------------------------------------------
                .sect   ".int08"                ; timer A section
                .word   TA_isr                  ; timer A isr
                .sect   ".reset"                ; MSP430 RESET Vect
                .word   start                   ; start address
                .end
```

TASSEL_2 = SMCLK

MC_1 = UP Mode

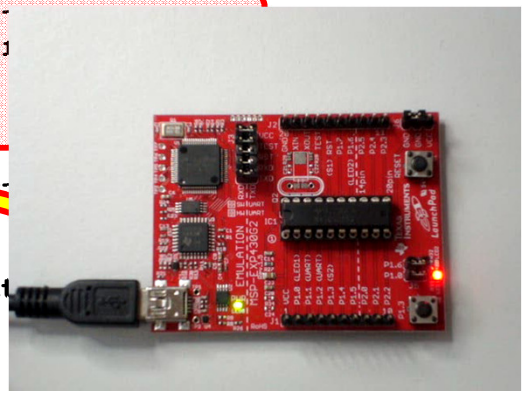
ID_3 = /8

Enable Interrupt

Put the processor to sleep!

NEVER EXECUTE!!!

Timer A ISR



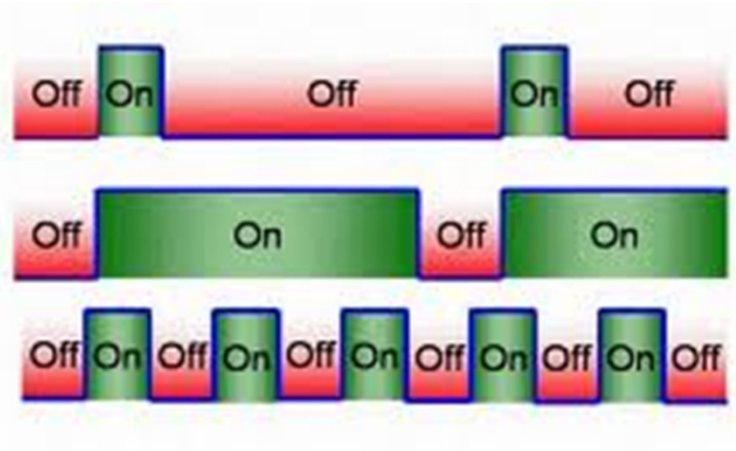
Quiz 9.3

1. How could I speed up the blink?

```
        .cdecls C,"msp430.h"       ; MSP430
TA_CTL  .equ    TASSEL_2|ID_3|MC_1|TAIE
TA_FREQ .equ    0xffff             ; clocks
STACK   .equ    0x0400             ; top of stack

; Code Section -------------------------------------
        .text
start:  mov.w   #STACK,SP          ; init stack pointer
        mov.w   #WDTPW|WDTHOLD,&WDTCTL
        bis.b   #0x01,&P1DIR       ; set P1.0 as output

        clr.w   &TAR               ; reset timerA
        mov.w   #TA_CTL,&TACTL     ; configure timerA
        mov.w   #TA_FREQ,&TACCR0   ; set interval
        bis.w   #LPM0|GIE,SR       ; LPM0 w/interrupts

error:  jmp     $                  ; NEVER GET HERE

; Timer A ISR --------------------------------------
TA_isr: bic.w   #TAIFG,&TACTL      ; acknowledge
        xor.b   #0x01,&P1OUT       ; toggle P1.0
        reti

; Interrupt Vectors --------------------------------
        .sect   ".int08"           ; timer A section
        .word   TA_isr             ; timer A isr
        .sect   ".reset"           ; MSP430 RESET Vector
        .word   start              ; start address
        .end
```

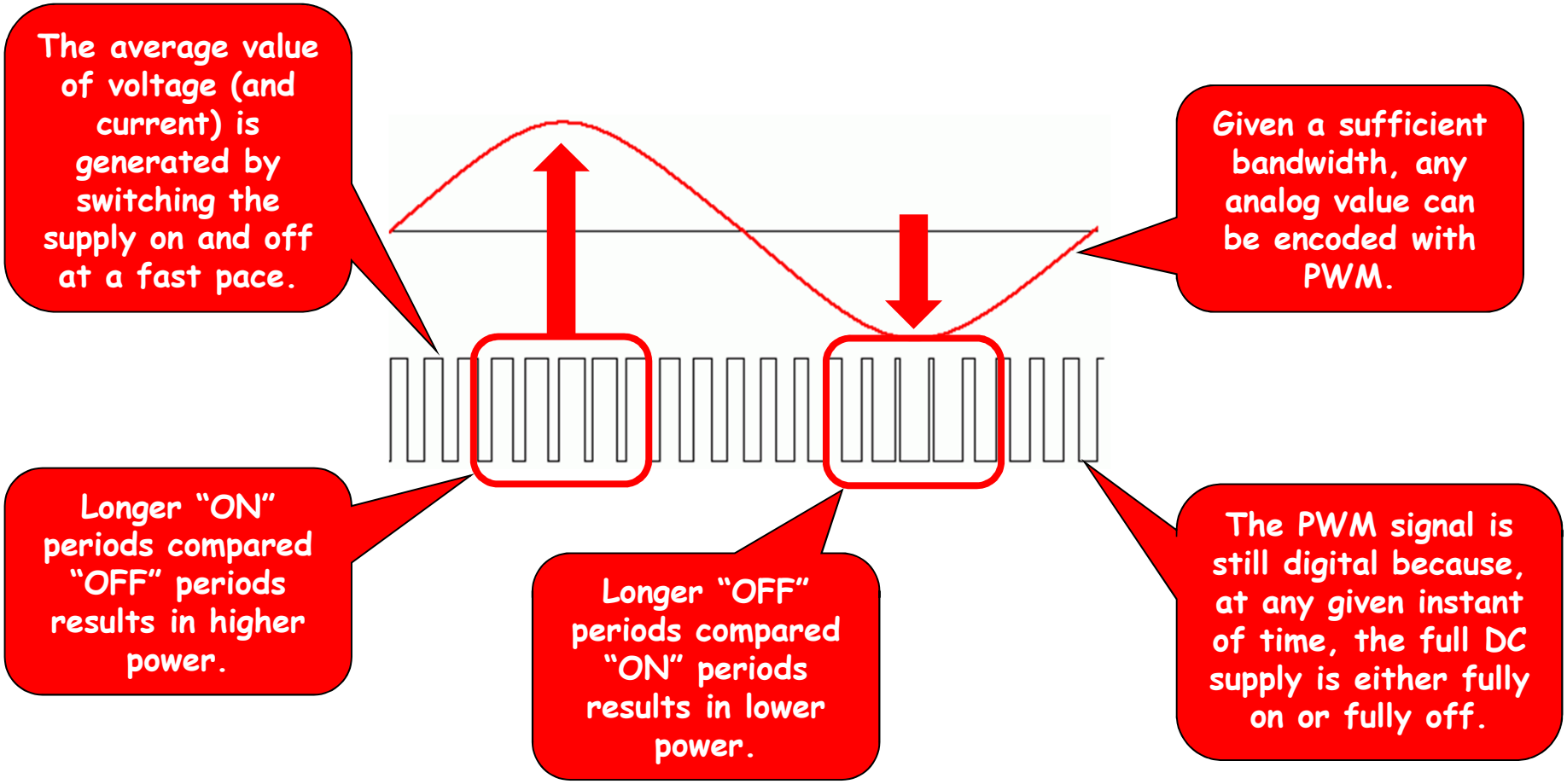
2. What happens if the Timer_A ISR doesn't acknowledge the interrupt?

Pulse Width Modulation



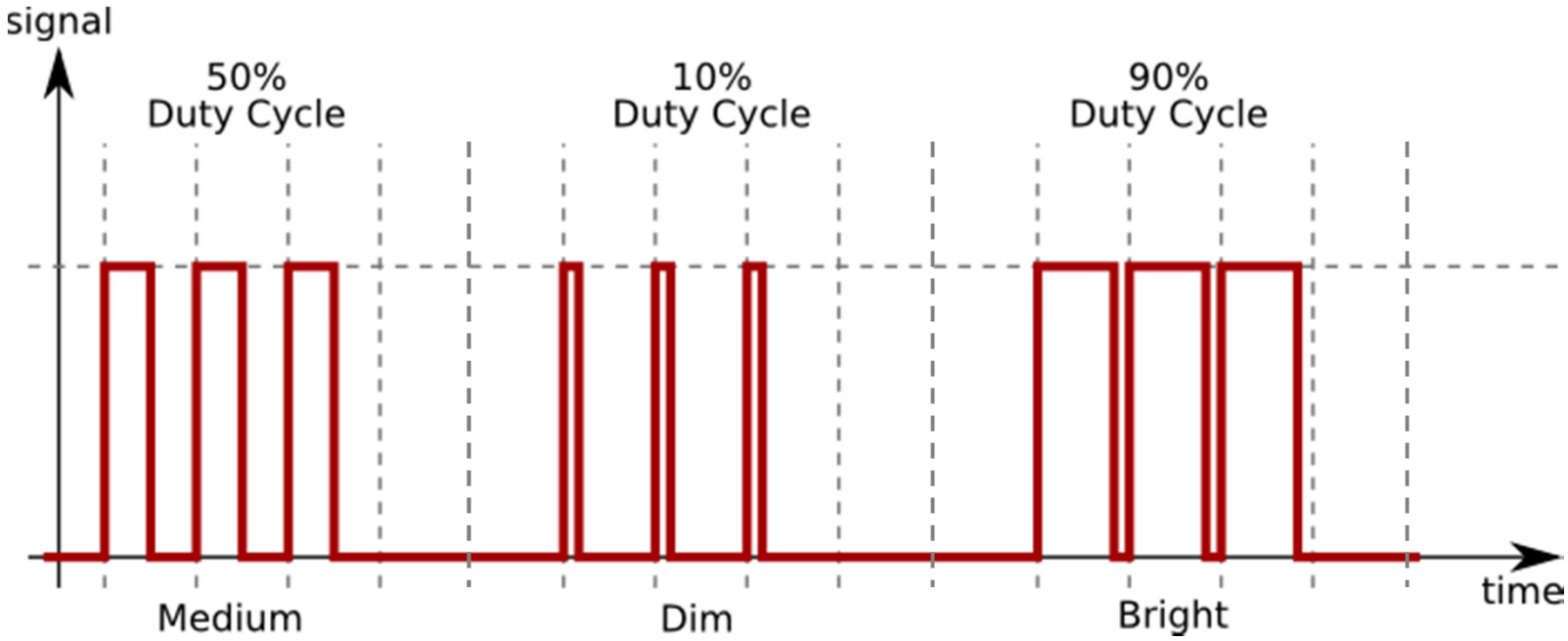
Pulse Width Modulation (PWM)

- PWM is a technique of digitally generating analog signals.



Examples of PWM Machines

PWM – Frequency/Duty Cycle

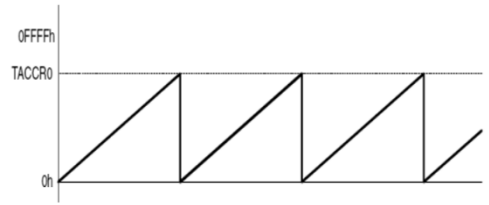


Device	Frequency	Duty Cycle
Speaker	Pitch	Volume
LED	Flicker	Brightness
Motor	Speed	Speed
Heater	Steadiness	Heat

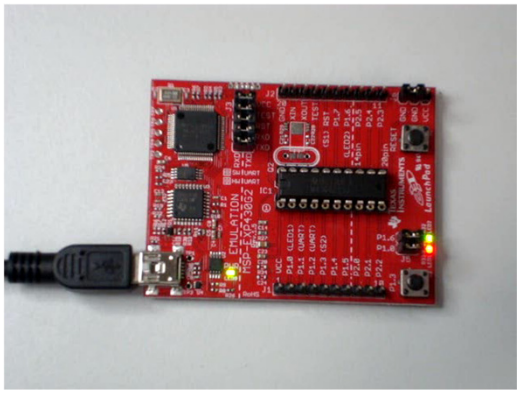
Example 9.5 – PWM w/Timer_A

```
            .cdecls C,"msp430.h"
TA_CTL      .equ    TASSEL_2|ID_0|MC_1|TAIE ; SMCLK, /1, UP, IE
TA_FREQ     .equ    120                     ; FREQ / SMCLK = 0.0001 = 100 us
STACK       .equ    0x0400                  ; top of stack

            .bss    pwm_duty,2              ; PWM duty cycle counter
            .bss    pwm_cnt,2               ; PWM frequency counter
; Code Section -------------------------------------------------------------
            .text
start:      mov.w   #STACK,SP               ; init stack pointer
            mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; stop watchdog
            bis.b   #0x41,&P1DIR            ; set P1.0,6 as output
            mov.b   #0x40,&P1OUT
            clr.w   &TAR                    ; reset timerA
            mov.w   #TA_CTL,&TACTL          ; set timerA control reg
            mov.w   #TA_FREQ,&TACCR0        ; set interval (frequency)
            clr.w   &pwm_duty               ; init PWM counters
            clr.w   &pwm_cnt

loop:       bis.w   #LPM0|GIE,SR            ; red -> green, enter LPM0 w/interrupts
            inc.w   &pwm_duty               ; increment &
            cmp.w   #100,&pwm_duty          ; 100% (full on)?
              jlo   loop                    ; n

loop02:     bis.w   #LPM0|GIE,SR            ; green -> red, enter LPM0 w/interrupts
            dec.w   &pwm_duty               ; decrement % (full off)?
              jne   loop02                  ; n
            jmp     loop                    ; y, repeat

; Timer A ISR --------------------------------------------------------------
TA_isr:     bic.w   #TAIFG,&TACTL           ; acknowledge interrupt
            cmp.w   &pwm_duty,&pwm_cnt      ; in duty cycle?
              jne   TA_isr2                 ; n
            xor.b   #0x41,&P1OUT            ; y, turn on LEDs

TA_isr2:    inc.w   &pwm_cnt                ; increment %
            cmp.w   #100,&pwm_cnt           ; 100%?
              jlo   TA_isr4                 ; n
            mov.b   #0x40,&P1OUT            ; y, reset LEDs state
            clr.w   &pwm_cnt                ; clear counter
            bic.w   #LPM0,0(SP)             ; wakeup processor

TA_isr4:    reti                            ; return from interrupt

; Interrupt Vectors --------------------------------------------------------
            "                               ; timerA isr
            ".reset"
            start                           ; PUC reset
```

Timer A Setup



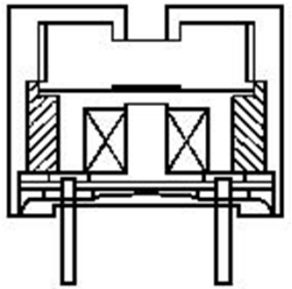
Adjust duty cycle in main program. (Sleep during cycle)

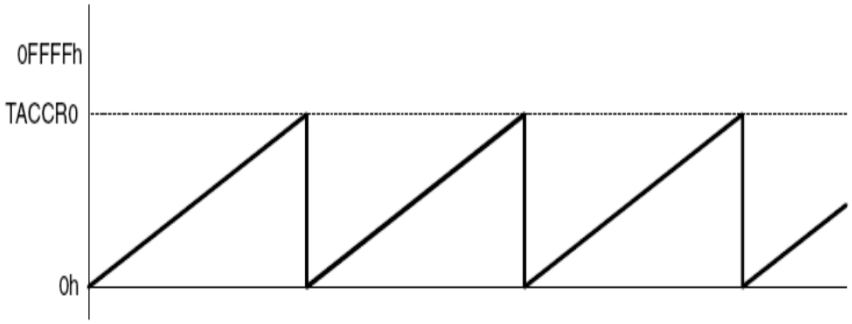


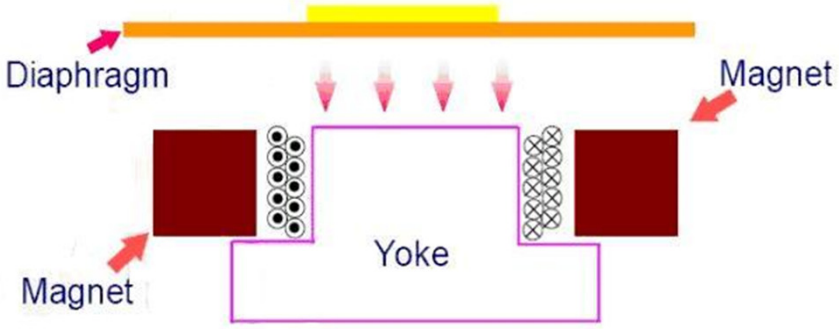
Use Timer A ISR to PWM LEDs

Speaker (Transducer)

- A speaker or magnetic transducer consists of a iron core, wound coil, yoke plate, permanent magnet, and vibrating diaphragm with a movable iron piece.
- A positive AC signal produces a fluctuating magnetic field, which causes the diaphragm to vibrate up and down, thus vibrating air.
- Use PWM on P1.1/P1.2 to produce tones.





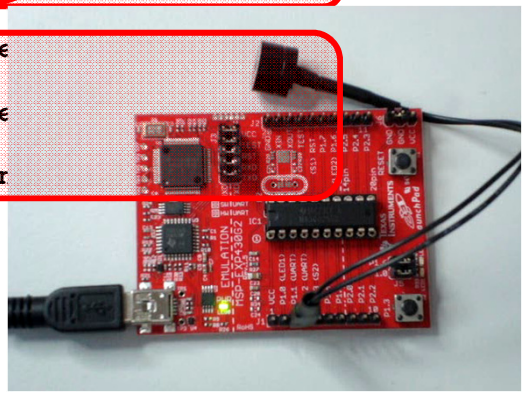


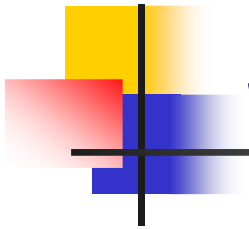
Example 9.6 – Watchdog PWM

```
                .cdecls C,"msp430.h"            ; include c header
WDT_CPS         .equ    1200000/500             ; WD clocks / second
STACK           .equ    0x0400                  ; stack

                .bss    WDTSecCnt,2             ; WDT second counter
                .bss    buzzON,1                ; buzzer on flag
; Code Section -------------------------------------------------------
                .text                           ; program section
start:          mov.w   #STACK,SP               ; initialize stack point
                mov.w   #WDT_MDLY_0_5,&WDTCTL   ; set WD timer interv      0.5 ms
                mov.w   #WDT_CPS,&WDTSecCnt     ; initialize 1 sec      ounter
                mov.b   #WDTIE,&IE1             ; enable WDT int        pt
                bis.b   #0x07,&P1DIR            ; P1.0 (LED)      2 (speaker)
                mov.b   #0x04,&P1OUT            ; set P1.1      1.2 to toggle
                clr.b   buzzON                  ; turn      buzzer
                bis.w   #LPM0|GIE,SR            ; ena      interrupts / sleep
                jmp     $                       ;      hould never get h

; Watchdog ISR -------------------------------------------------------
WDT_ISR:        tst.b   buzzON                  ; buzzer on?
                  jeq   WDT_02                  ; n
                xor.b   #0x06,&P1OUT            ; y, use 50% PWM

WDT_02:         dec.w   &WDTSecCnt              ; decr      nt counte
                  jne   WDT_04                  ;
                mov.w   #WDT_CPS,&WDTSecCnt     ; y, re-initialize
                xor.b   #0x01,&P1OUT            ; toggle led
                xor.b   #0xff,buzzON            ; toggle buzzer or

WDT_04:         reti                            ; return from WDT

                .sect   ".int10"
                .word   WDT_ISR                 ; Watchdog ISR
                .sect   ".reset"
                .word   start                   ; RESET ISR
                .end
```

PWM speaker (toggle P1.1/P1.2) when buzzON is non-zero (50% duty cycle)

Toggle buzzON every second





Summary

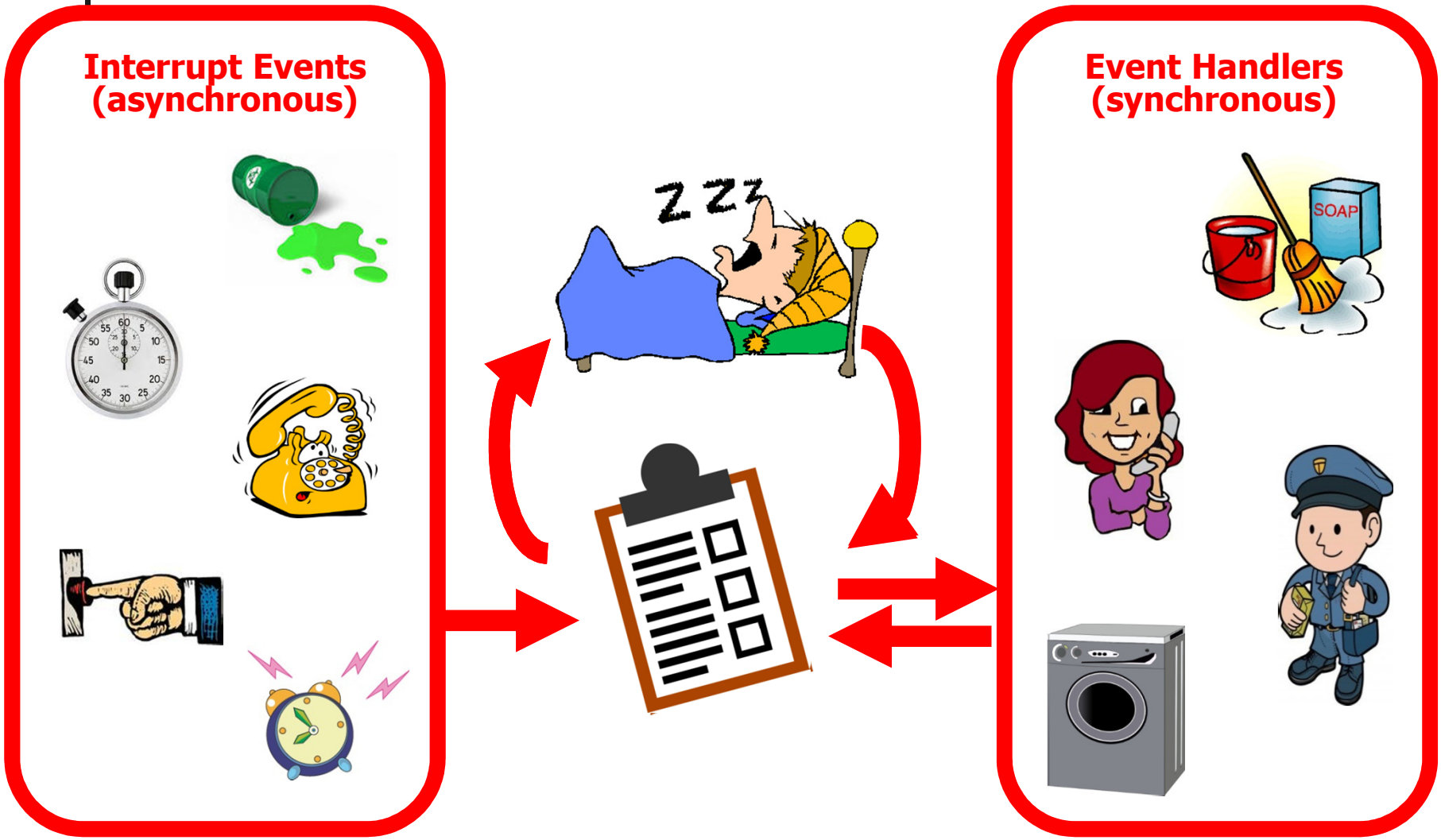
- By coding efficiently you can run multiple peripherals at high speeds on the MSP430
- Polling is to be avoided – use interrupts to deal with each peripheral only when attention is required
- Allocate processes to peripherals based on existing (fixed) interrupt priorities - certain peripherals can tolerate substantial latency
- Use GIE when it's shown to be most efficient and the application can tolerate it – otherwise, control individual IE bits to minimize system interrupt latency.
- An interrupt-based approach eases the handling of *asynchronous* events

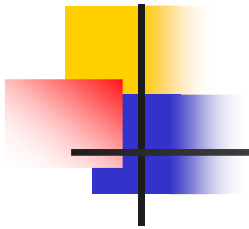
Event Driven Programming Model

Programming Paradigms

- **Imperative Programming**
  - computation in terms of statements that change a program state
- **Functional Programming**
  - computation as the evaluation of mathematical functions and avoids state and mutable data.
- **Procedural / Structured Programming**
  - specifying the steps the program must take to reach the desired state
- **Object Oriented Programming (OOP)**
  - uses "objects" – data structures consisting of datafields and methods together with their interactions – to design applications and computer programs.
- **Declarative Programming**
  - expresses the logic of a computation without describing its control flow
- **Automata-based Programming**
  - the program models a finite state machine or any other formal automata.
- **Event Driven Programming**
  - the flow of the program is determined by events, i.e., sensor outputs, user actions (mouse clicks, key presses), messages from other programs or threads.

Events / Event Handlers





Event Driven Programming

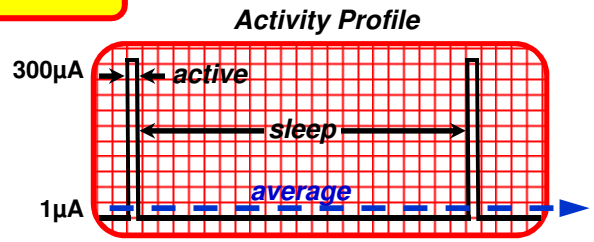
- System events
  - sensor outputs (completion interrupts)
  - internal generated events (timers)
  - user actions (mouse clicks, key presses)
  - messages from other programs or threads.
- Program has two sections:
  - event selection.
  - event handling.
- Main loop
  - constantly running main loop, or
  - sleep w/interrupts (preferred)

Example 9.7 – EDP Model

```
 .cdecls C,"msp430.h"
TA_CTL      .equ    TASSEL_2+ID_3+MC_1+TAIE ; SMCLK,/8,UP,IE
TA_FREQ     .equ    0xffff                  ; timerA frequency
WDT_CPS     .equ    1200000/32000           ; WDT clocks/second
STACK       .equ    0x0400                  ; tos

WDT_EVENT   .equ    0x0001                  ; WDT event
TA_EVENT    .equ    0x0002                  ; timerA event

; Data Section ------------------------------------------------
            .bss    WDTSecCnt,2             ; WDT second count
            .bss    sys_event,2             ; system events
; Code Section ------------------------------------------------
            .text
start:      mov.w   #STACK,SP               ; init stack pointer
            mov.w   #WDT_MDLY_32,&WDTCTL    ; WDT interval
            mov.w   #WDT_CPS,&WDTSecCnt     ; WDT 1s counter
            bis.b   #WDTIE,&IE1             ; enable WDT interrupt
            clr.w   &TAR                    ; reset timerA
            mov.w   #TA_CTL,&TACTL          ; timerA control register
            mov.w   #TA_FREQ,&TACCR0        ; interval (frequency)
            bis.b   #0x41,&P1DIR            ; set P1.0,6 as output
            clr.w   &sys_event              ; clear pending events
```

Event Loop

```
; Event Loop --------------------------------------------------
loop:       bic.w   #GIE,SR                 ; disable interrupts
            cmp.w   #0,&sys_event           ; interrupt pending?
              jne   loop02                  ; y
            bis.w   #GIE|LPM0,SR            ; n, enable/sleep

loop02:     cmp.w   #WDT_EVENT,&sys_event   ; WDT event?
              jne   loop04                  ; n
            bic.w   #WDT_EVENT,&sys_event   ; y, clear event
            xor.b   #0x01,&P1OUT            ; toggle red LED

loop04:     cmp.w   #TA_EVENT,&sys_event    ; timerA event?
              jne   loop06                  ; n
            bic.w   #TA_EVENT,&sys_event    ; y, clear event
            xor.b   #0x40,&P1OUT            ; toggle green LED

loop06:
;           << process other events here >>
            jmp     loop                    ; loop indefinitely
```



Watchdog Event

```
; Watchdog ISR ------------------------------------------------
WDT_ISR:    dec.w   &WDTSecCnt              ; 1 second?
              jne   WDT_02                  ; n
            mov.w   #WDT_CPS,&WDTSecCnt     ; y, reset counter
            bis.w   #WDT_EVENT,&sys_event   ; schedule WDT event
            bic.w   #GIE|LPM0,0(SP)         ; wakeup processor

WDT_02:     reti                            ; exit ISR
```

```
; Timer A ISR -------------------------------------------------
TA_isr:     bic.w   #TAIFG,&TACTL           ; ack interrupt
            bis.w   #TA_EVENT,&sys_event    ; schedule timerA event
            bic.w   #GIE|LPM0,0(SP)         ; wakeup processor
            reti
```

```
; Interrupt Vectors -------------------------------------------
            .sect   ".int__"                ; timerA section
            .word   TA_                     ; timerA isr
            .sect   "__10"                  ; Watchdog Vector
            .word   ISR                     ; Watchdog ISR
            .word   0                       ; PUC Vector
            .word   0                       ; RESET ISR
```

TimerA Event