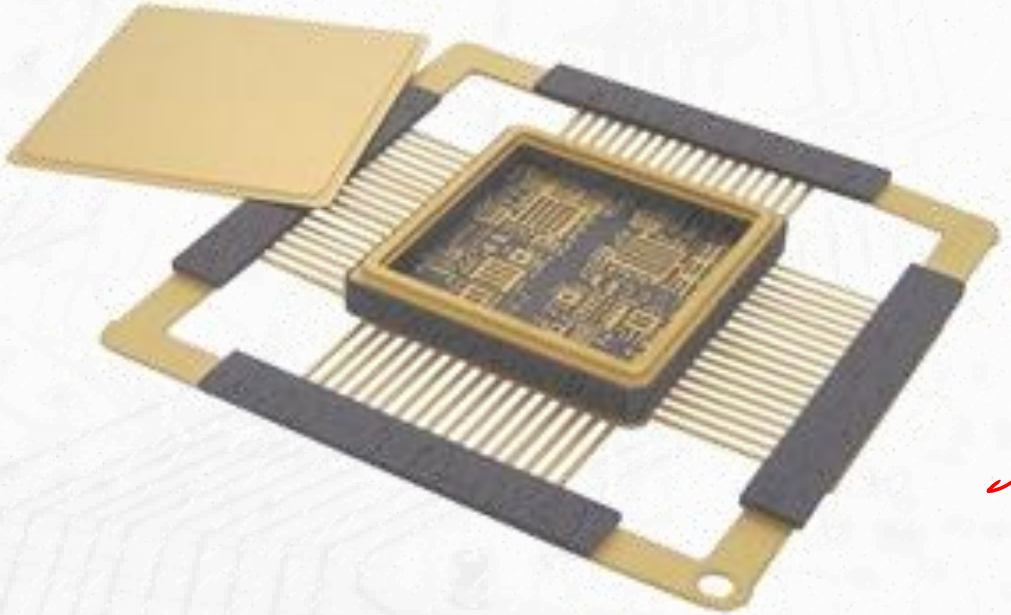
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
& ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ



**Μάθημα 10ο**

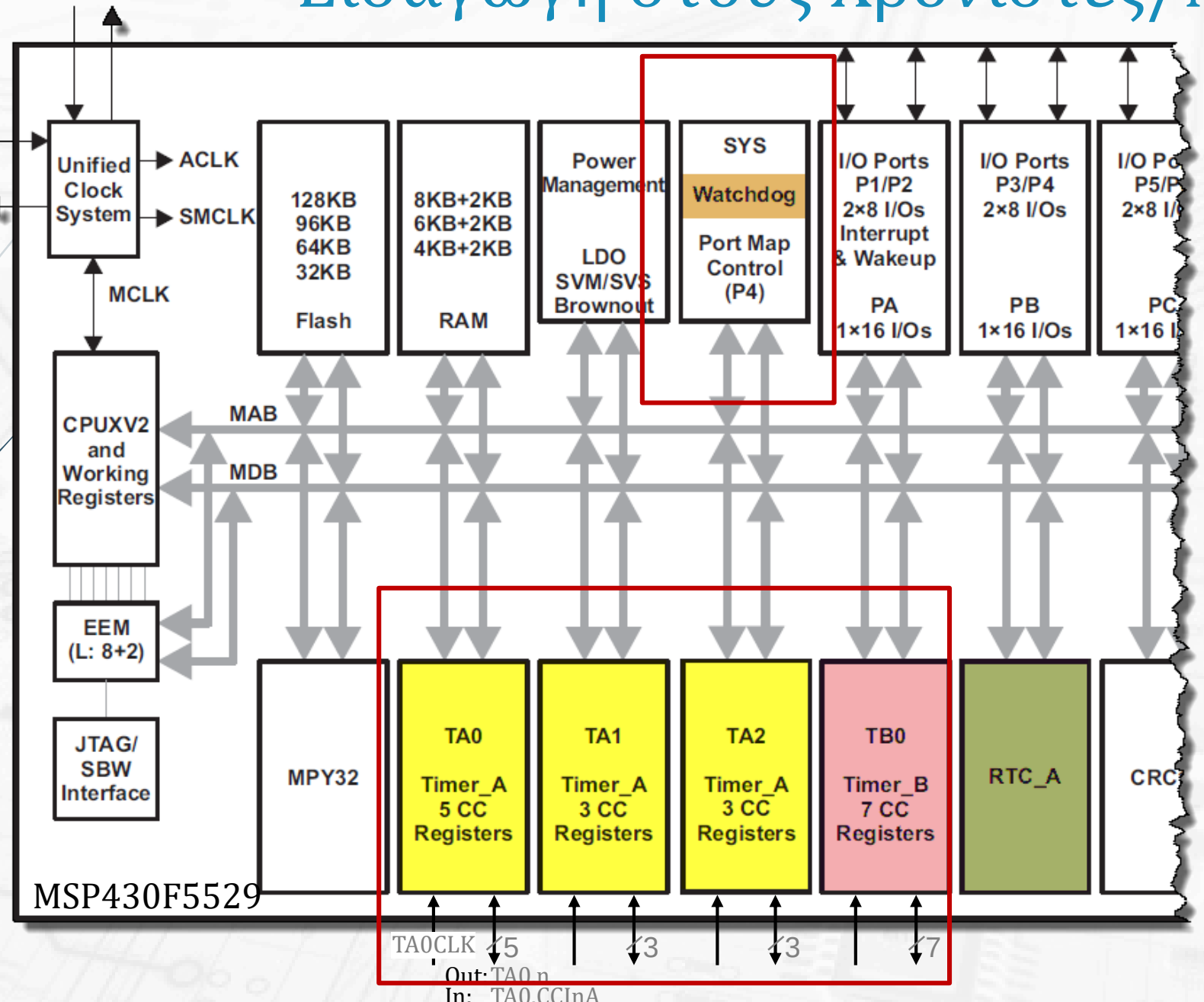
# Περιφερειακά Μικροελεγκτών
# Διαχείριση κατανάλωσης

Ενσωματωμένα Συστήματα

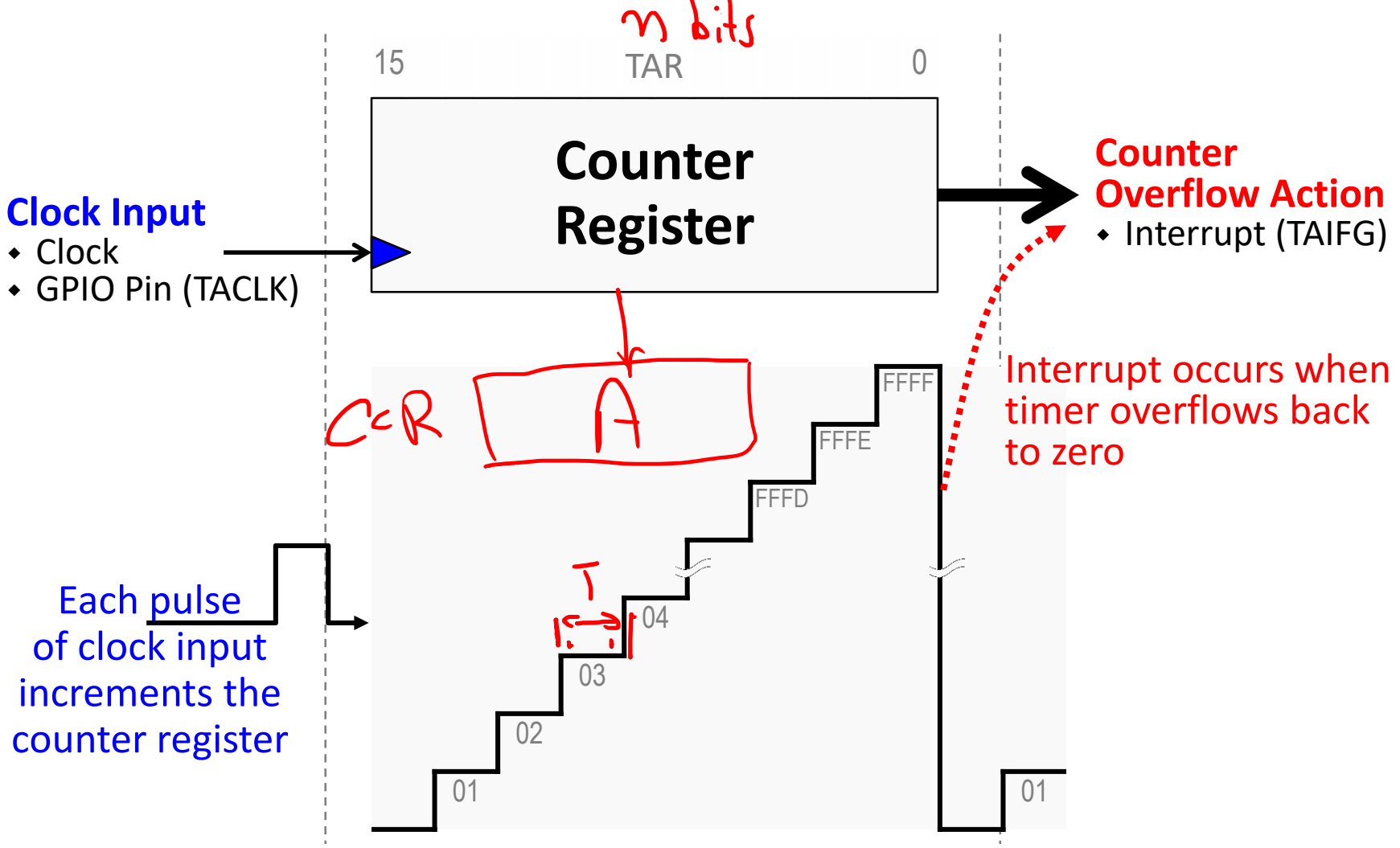
Διδάσκον: Γρηγόριος Δουμένης

Διδακτικό Υλικό: Γρηγόριος Δουμένης, Νικόλαος Γιαννακέας, Ιωάννης Μασκλαβάνος

# Εισαγωγή στους Χρονιστές/Μετρητές



# Timer/Counter Basics




**Notes**

- Timers are often called “Timer/Counters” as a counter is the essential element
- “Timing” is based on counting inputs from a known clock rate
- Actions don’t occur when writing value to counter

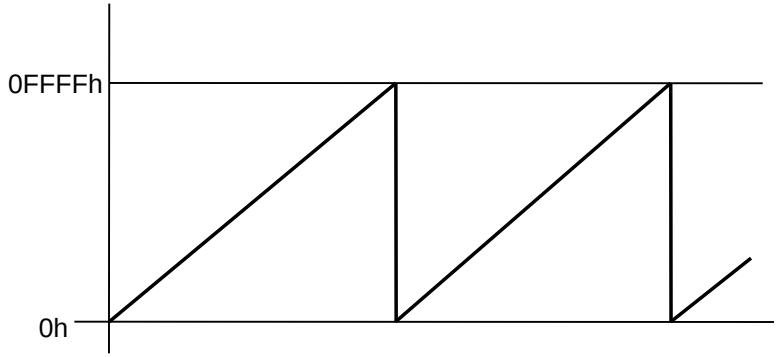
Can I 'capture' a count/time value?

# Χρονιστές/Μετρητές
## MSP430 – Timer_A

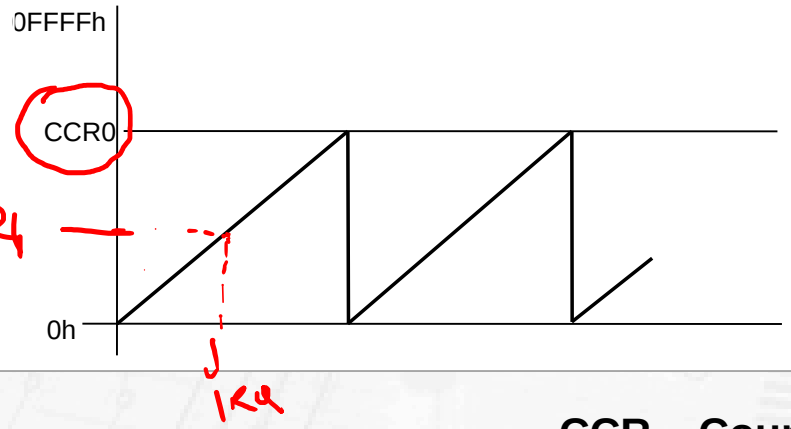
**Stop/Halt**
Timer is halted



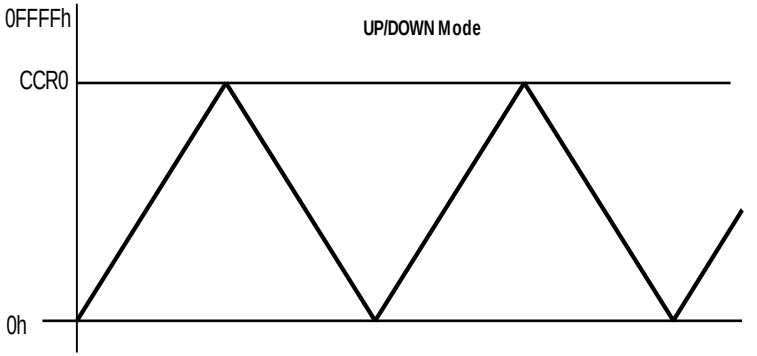
**Continuous**
Timer continuously counts up




**Up**
Timer counts between 0 and CCR0




**Up/Down**
Timer counts between 0 and CCR0 and 0

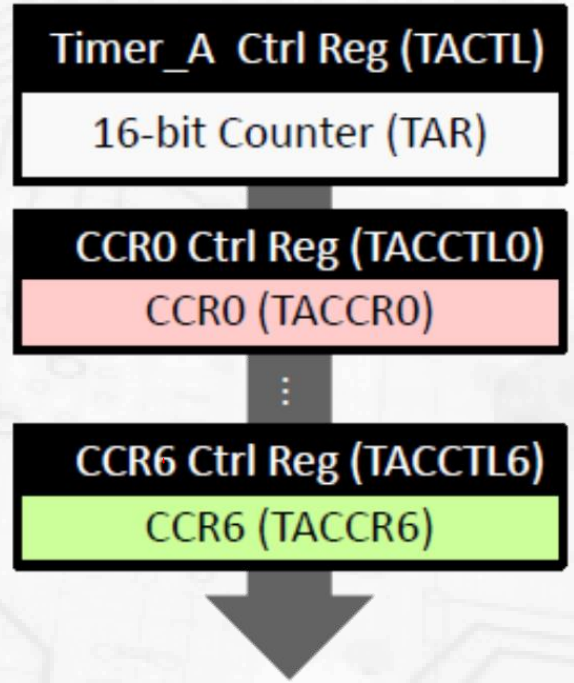



**CCR – Count Compare Register**

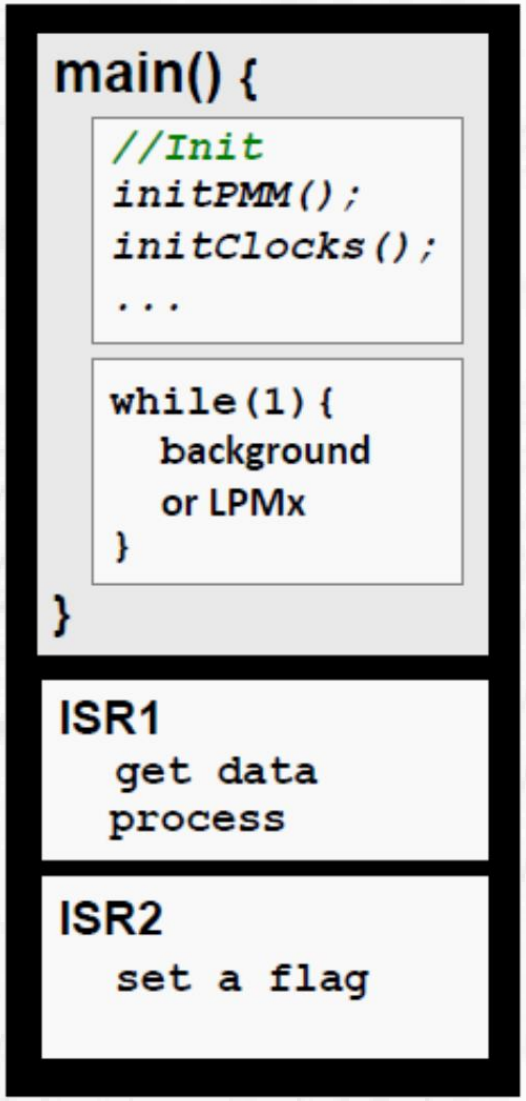
# Χρονιστές/Μετρητές
## Προγραμματισμός

### Timer Setup Code

1. Configure Timer/Counter (TACTL)
   - Clocking
   - Which Count Mode
   - Interrupt on TAR rollover?
2. Setup Capture and/or Compare Registers
   - Capture (TACCTL):
     - Input
     - Interrupt on Capture?
   - Compare (TACCTL, TACCR):
     - Compare-to Value
     - Output mode (How output signal changes at compare (EQU) events)
     - Interrupt on Compare?
3. Clear interrupt Flags & Start Timer



# Χρονιστές/Μετρητές
## Προσκήνιο – Παρασκήνιο




**System Initialization**

- The beginning part of main() is usually dedicated to setting up your system (Chapters 3 and 4)

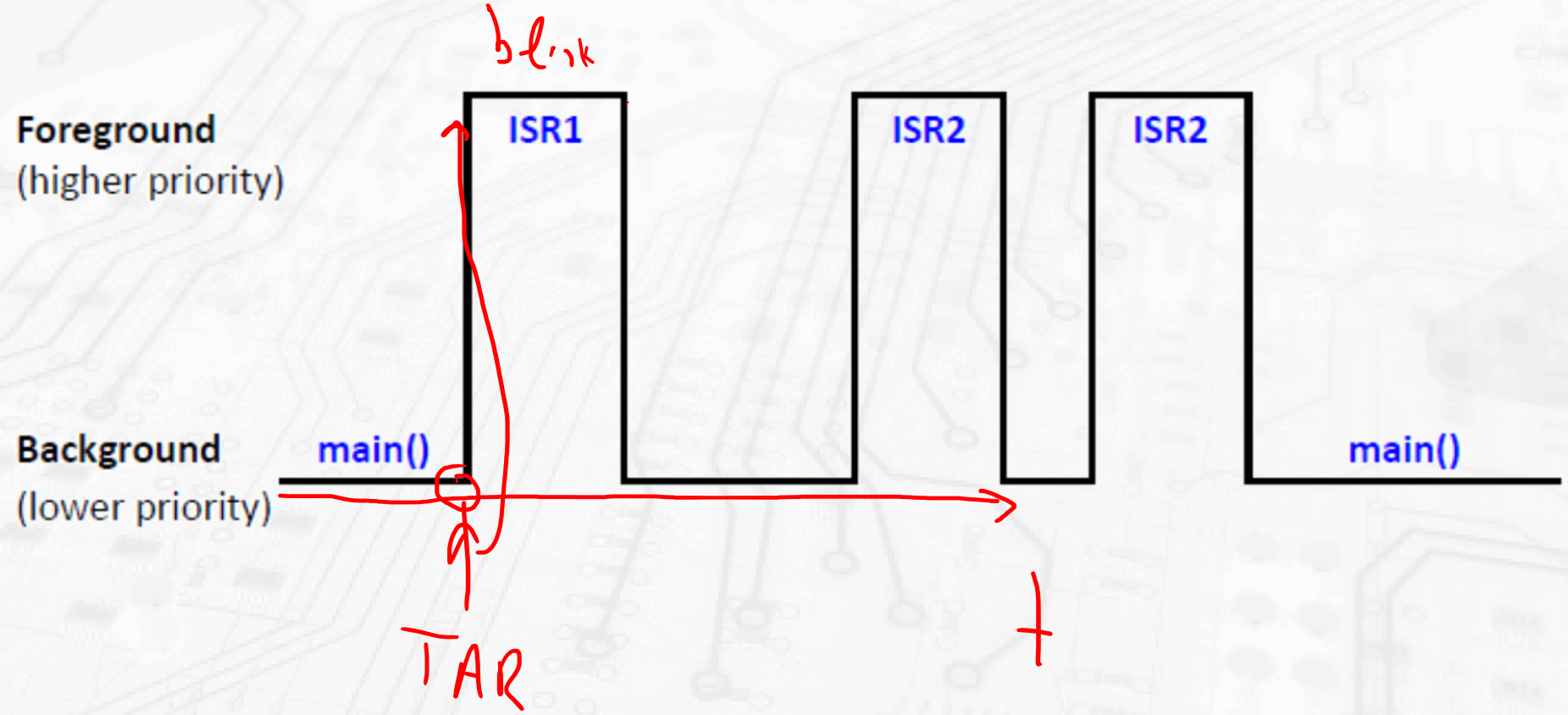
**Background**

- Most systems have an endless loop that runs 'forever' in the background
- In this case, 'Background' implies that it runs at a lower priority than 'Foreground'
- In MSP430 systems, the background loop often contains a Low Power Mode (LPMx) command – this sleeps the CPU/System until an interrupt event wakes it up

**Foreground**

- Interrupt Service Routine (ISR) runs in response to enabled hardware interrupt
- These events may change modes in Background – such as waking the CPU out of low-power mode
- ISR's, by default, are not interruptible
- Some processing may be done in ISR, but it's usually best to keep them short

# Χρονιστές/Μετρητές
## Προσκήνιο – Παρασκήνιο



# TIMER SETUP

```
#include "msp430.h"

volatile unsigned int time_stuck=0;

int main(void)
{
  WDTCTL = WDTPW + WDTHOLD;        // Stop watchdog timer

  TA0CCTL0 = CCIE;                          // CCR0 interrupt enabled
  TA0CTL = TASSEL_2 | MC_1 | ID_3;   // SMCLK/8, upmode
  TA0CCR0 = 40000;                         // Blink the LEDs every ~1sec


_BIS_SR(GIE);                 // Enter w/ interrupt
```

# TIMER IRQ

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A0 (void)
{
time_stuck=1; // Signal timer event
}
```

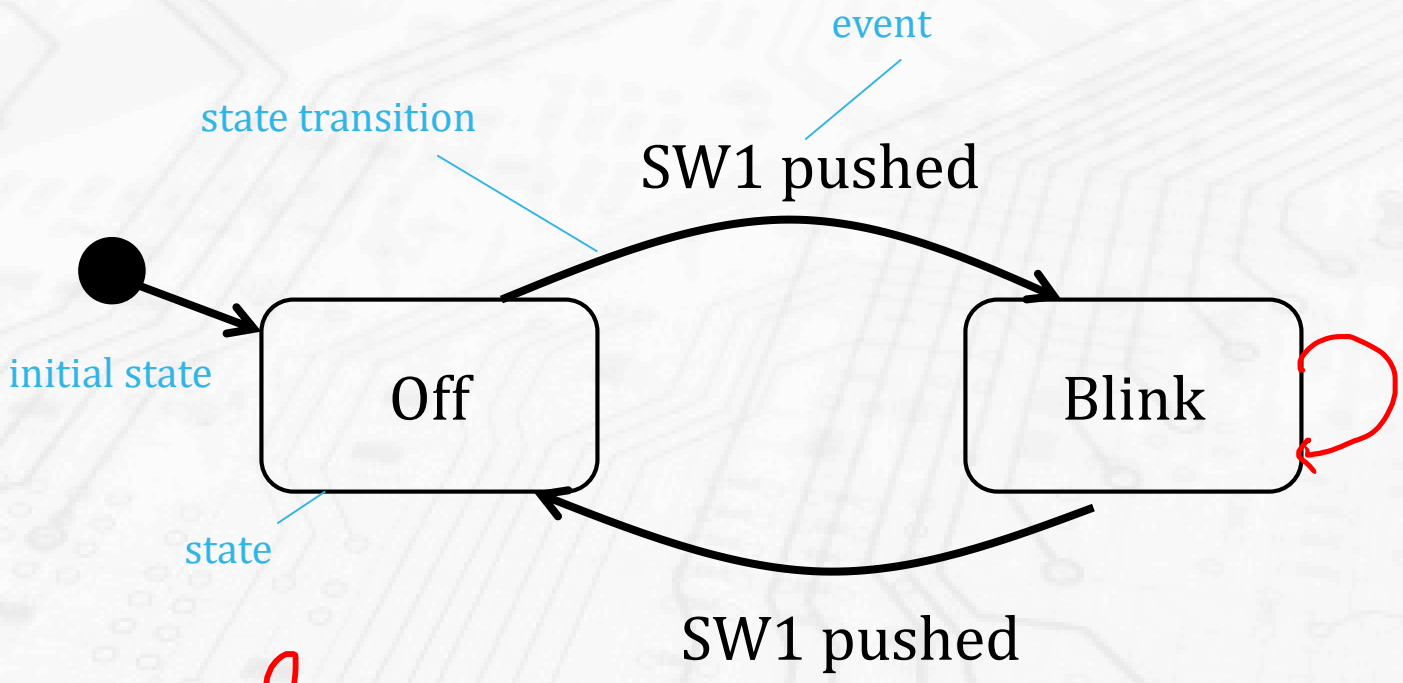
# TIMER LED BLINK

```
volatile unsigned int d_freq=0;
 volatile unsigned int i=0;
 volatile unsigned int time_count=0;
 volatile unsigned int cycle_count=0;
 for (;;)
 {
       if (time_stuck==1) {
          time_stuck=0;  //clear the flag, to permit next “interrupt”


          cycle_count++;  //this part is for the “double frequency” feature
          if (d_freq) {    //if in double frequency toggle GREEN LED
                   P4OUT ^=0x80;}
          if (cycle_count==2) {
               cycle_count=0;
               P1OUT ^=0x01;  // toggle RED LED(every ~1 sec)
              if (!d_freq) P4OUT ^=0x80;
               }
      }  //end if (time_stuck)
  } //end for
return 0;
}
```

# Μηχανές καταστάσεων

Volatile unsigned blink_state=0; // 0=off, 1 =blink

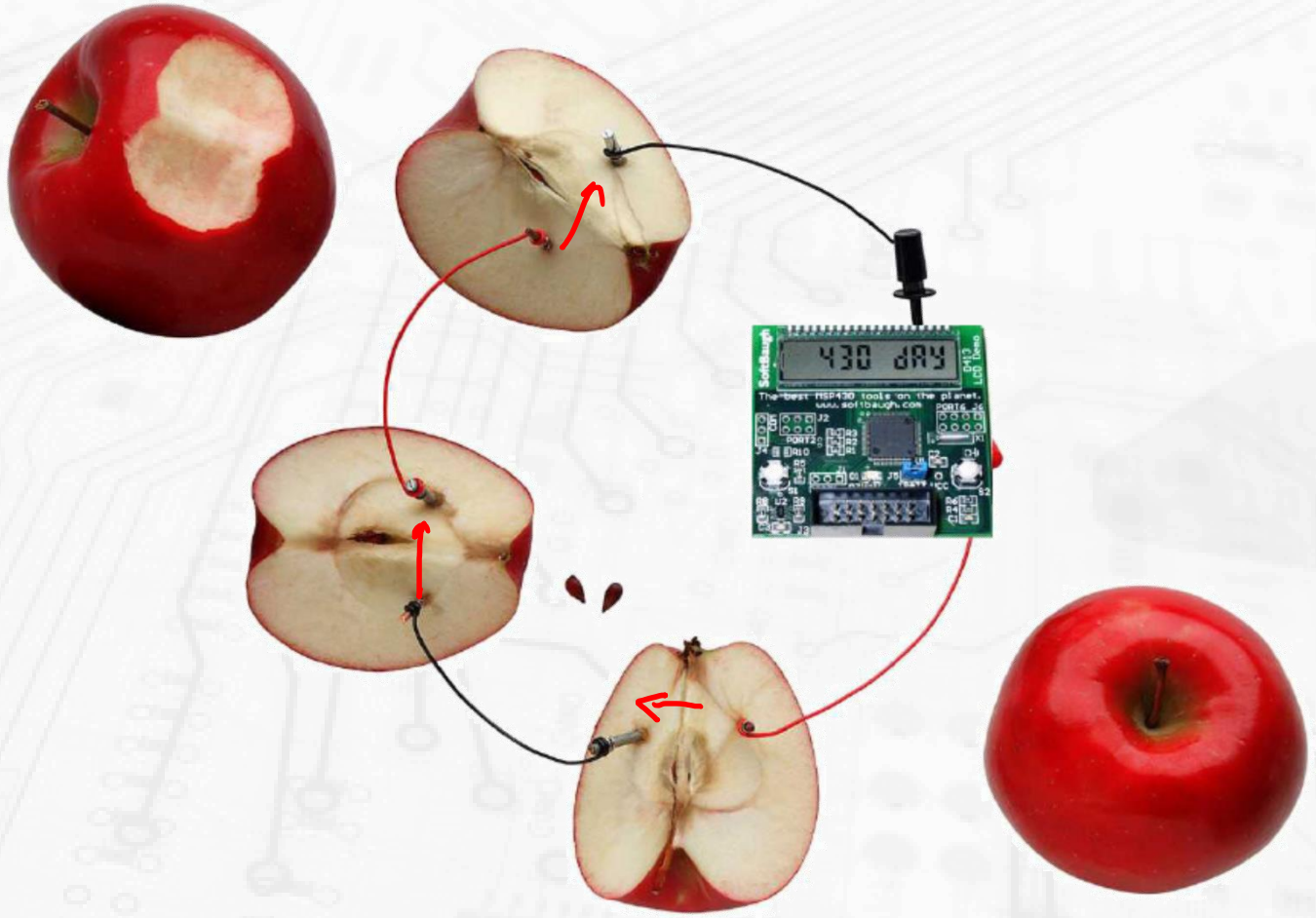



```
if (blink_state==1)
PORTA=0x01;
```

# Low Power Modes (LPM)

## Εισαγωγή

Είναι σημαντική η εξοικονόμηση ενέργειας στα **αυτόνομα ενσωματωμένα συστήματα?**



# Low Power Modes (LPM)

## Εισαγωγή

$E = P \cdot T$

$1\,J = 1\,W \cdot 1\,S$

- Οι σημερινές CPU σχεδιάζονται λαμβάνοντας υπόψιν την κατανάλωση ισχύος.
- Ενέργεια έναντι ισχύος:
  - Ισχύς είναι η κατανάλωση ενέργειας ανά μονάδα χρόνου.
  - Η παραγωγή θερμότητας εξαρτάται από την κατανάλωση ισχύος.
  - Η διάρκεια ζωής της μπαταρίας εξαρτάται από την κατανάλωση ενέργειας.
- Συνήθως χρησιμοποιείται η ισχύς (power) και για τα δυο.

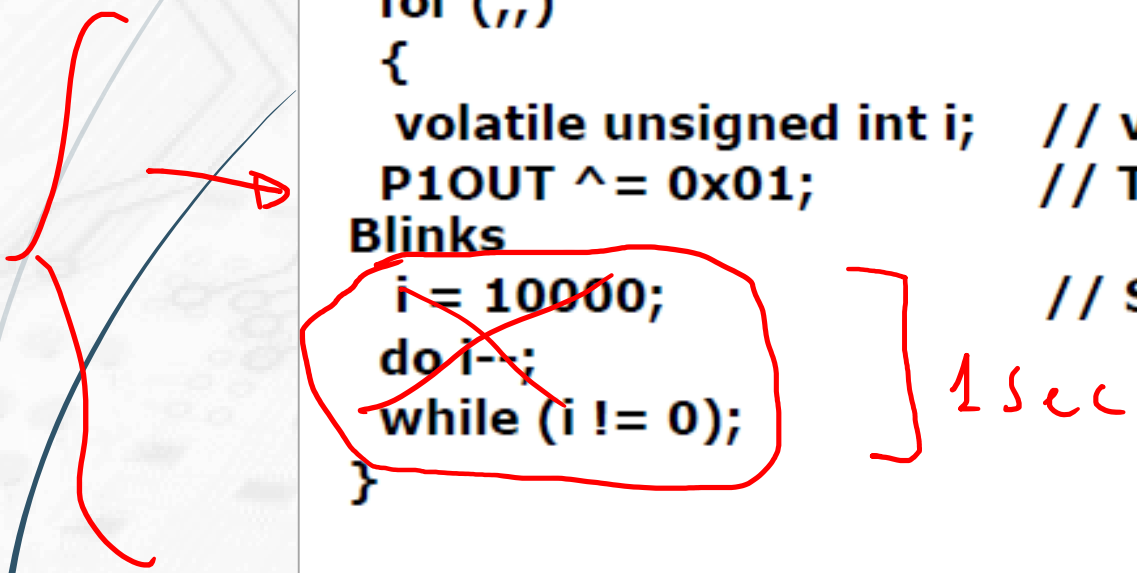
$E = 1\,W \cdot 10\,h = 10\,Wh = 10\,W \cdot 3600\,S = 36000\,\frac{J}{W \cdot S}$

# Low Power Modes (LPM)
## Εισαγωγή

- Όλα τα ψηφιακά συστήματα χτίζονται με κυκλώματα CMOS *(Complementary metal oxide semiconductor {MOS}).*
- Βασικές πηγές κατανάλωσης ισχύος:
  - **Πτώσεις τάσης** *(voltage drops)*: η κατανάλωση ισχύος είναι ανάλογη με V2 *(βελτίωση με μείωση τάσης τροφοδοσίας, ίσως με προσθήκη παράλληλου υλικού για να διατηρηθεί η απόδοση).*
  - **Αλλαγή κατάστασης εξόδων** *(toggling, dynamic):* υπάρχει κατανάλωση όταν αλλάζει η τιμή της εξόδου *(βελτίωση με εξάλειψη περιττών εναλλαγών ή glitches).*
  - **Διαρροή** *(leakage, static)*: υπάρχει διαρροή μέσω υποστρώματος ακόμη και αν δεν αλλάζει κάτι); μπορεί να βελτιωθεί με την αποσύνδεση από τη τροφοδοσία *(αλλά απαιτείται σημαντικό χρονικό διάστημα για την επανασύνδεση και εκ νέου αρχικοποίηση της εσωτερικής κατάστασης)*.

```
WDTCTL = WDTPW + WDTHOLD;          // Stop watchdog timer
  P1DIR |= 0x01;                   // Set P1.0 to output direction
  for (;;)
  {
   volatile unsigned int i;    // volatile to prevent optimization
  P1OUT ^= 0x01;               // Toggle P1.0 using exclusive-OR - Red
Blinks
   i = 10000;                  // SW Delay
  do i--;
  while (i != 0);
}

//Most of time/energy spent in the do / while loop
   (doing nothing!!)
```



# Low Power Modes (LPM)
## Εισαγωγή

**Τρόποι μείωσης Ενέργειας**

- **Στατικός** *(static):* καλείται από το χρήστη, και δεν εξαρτάται από τις δραστηριότητες CPU.
- *–Π.χ. Εντολή για shutdown/hibernate.*
- **Δυναμικός** *(dynamic):* η CPU προβαίνει σε ενέργειες για να ελέγξει την ισχύ βασιζόμενη στη δυναμική δραστηριότητα.
- *–Π.χ. Απενεργοποίηση τμημάτων.*

- Μειωμένα επίπεδα τάσης (5V -> 3.3V, 52/3.32=2.29 ).
- Χαμηλότερη συχνότητα ρολογιού *(μείωση ισχύς, όχι ενέργειας).*
- Απενεργοποίηση λειτουργικών μονάδων με σήματα ελέγχου, όταν δε χρειάζονται
- Αποσύνδεση των τμημάτων από την τροφοδοσία όταν δε χρειάζονται

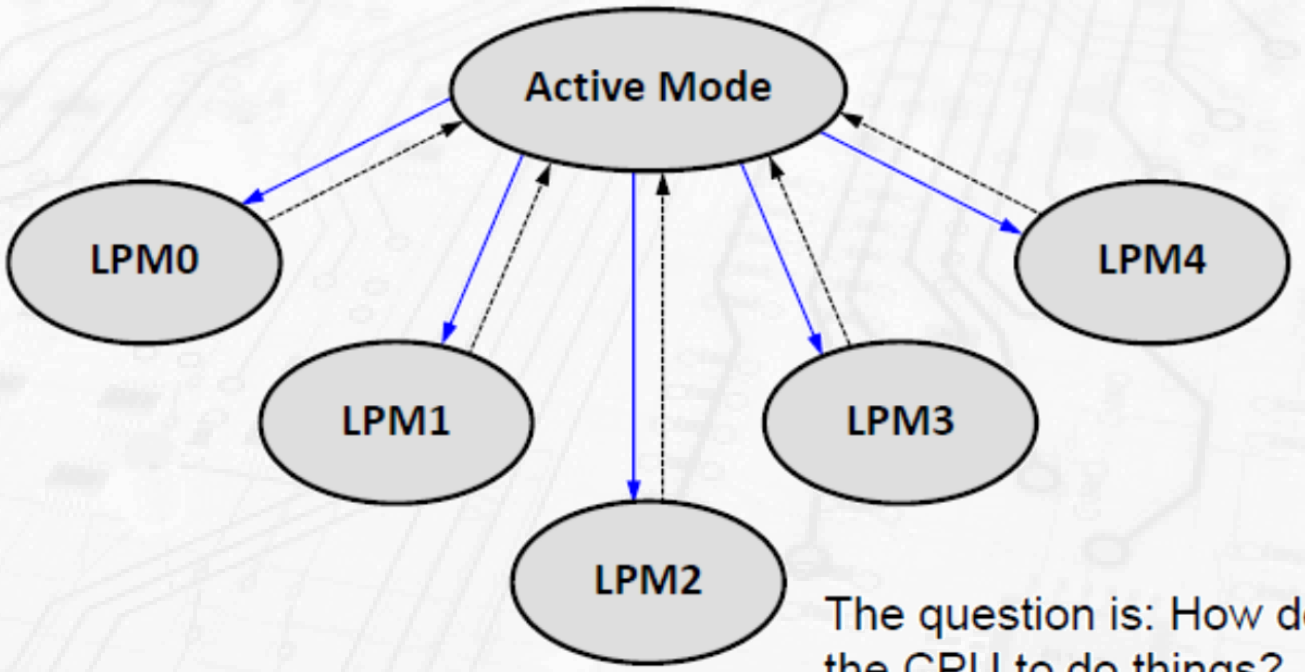
# MSP Low Power provisions

- MSP430 designed for ULP from ground up
- Peripherals optimized to reduce power and minimize CPU usage
- Intelligent, low power peripherals can operate independently of CPU and let the system stay in a lower power mode longer
  www.ti.com/ulp

- ✓ Multiple operating modes
  - 100 nA power down (RAM retained)
  - 0.3 µA standby
  - 110 µA / MIPS from RAM
  - 220 µA / MIPS from Flash
- ✓ Instant-on stable high-speed clock
- ✓ 1.8 - 3.6V single-supply operation
- ✓ Zero-power, always-on BOR
- ✓ <50nA pin leakage
- ✓ CPU that minimizes cycles per task
- ✓ Low-power intelligent peripherals
  - ADC that automatically transfers data
  - Timers that consume negligible power
  - 100 nA analog comparators
- ✓ Performance over required operating conditions

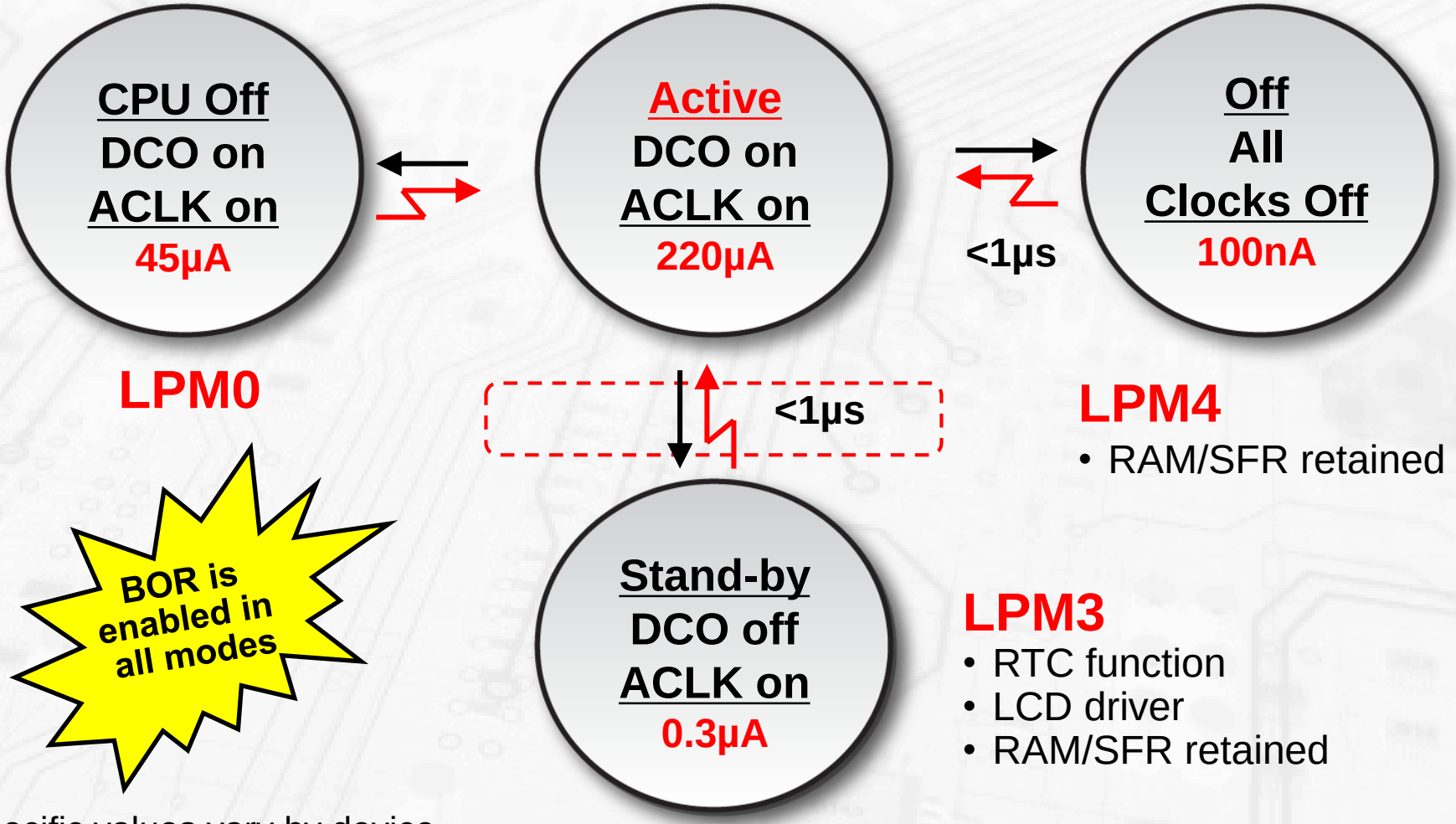
# Low Power Modes (LPM)
## Εισαγωγή

What if we could “stop” the CPU if there is nothing to do?
- i.e. when we use timers to count time



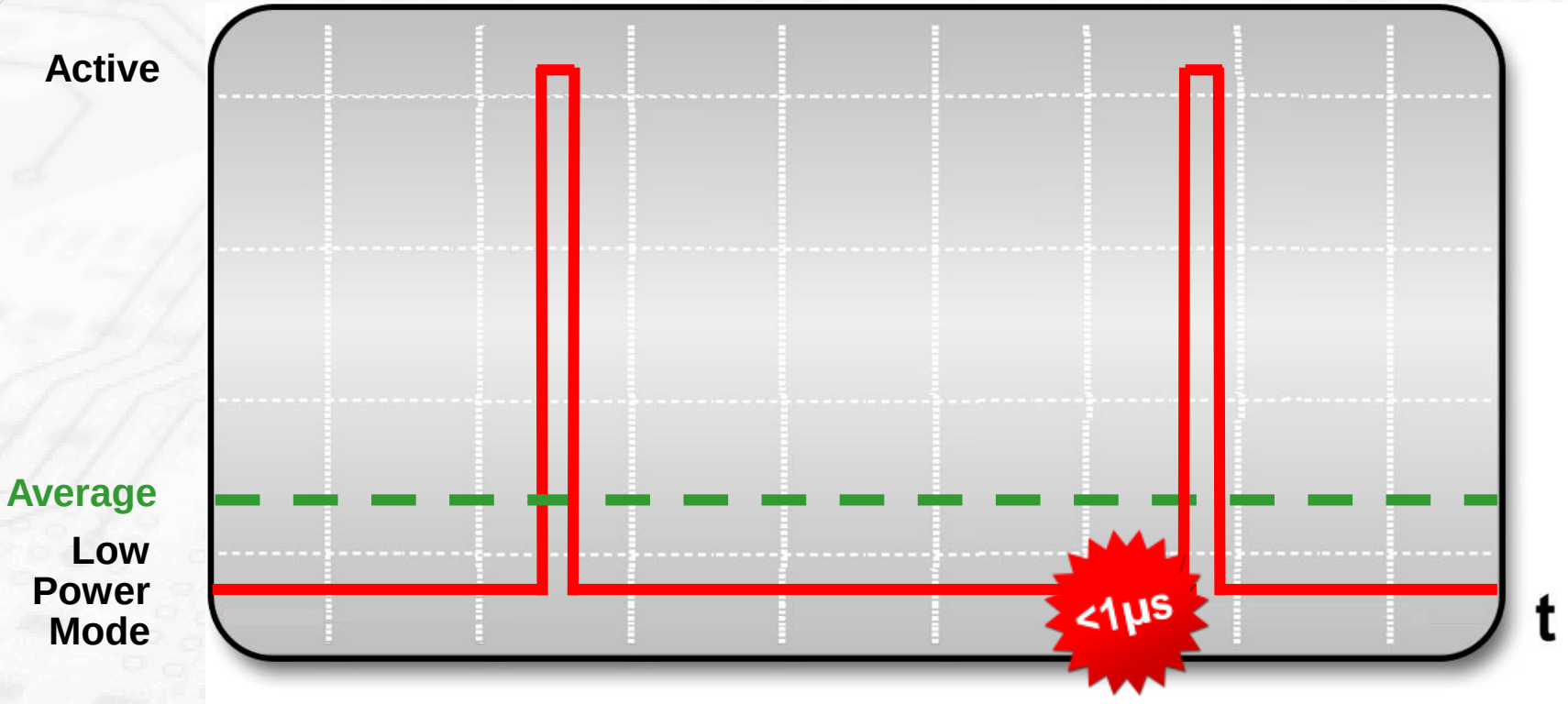

The question is: How do you “wake-up” the CPU to do things?

# MSP430 Low Power Modes




Specific values vary by device

# Ultra-Low Power Activity Profile

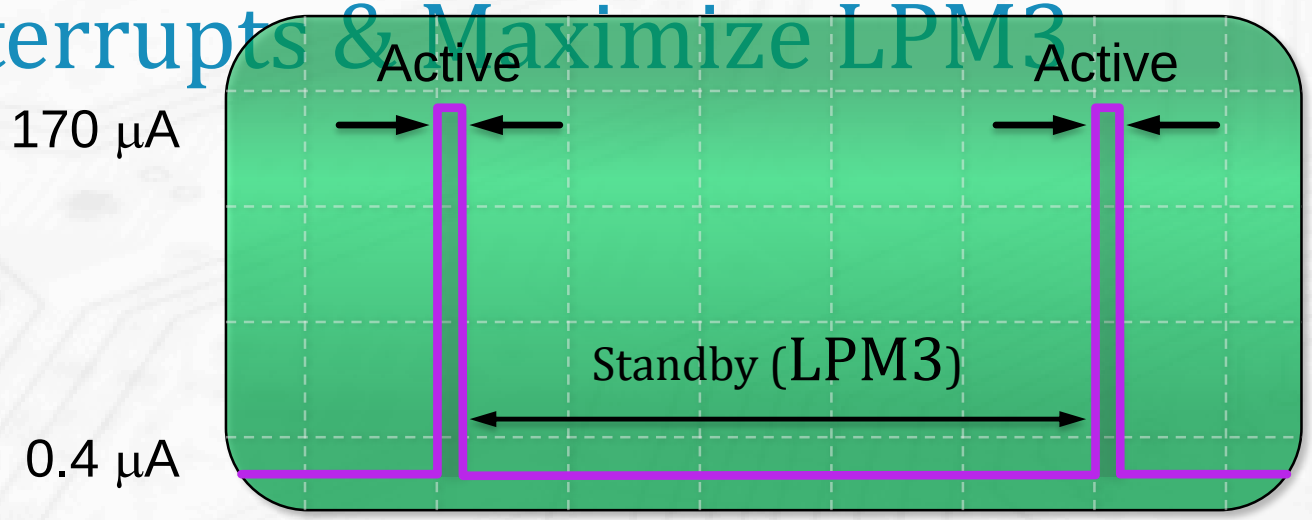


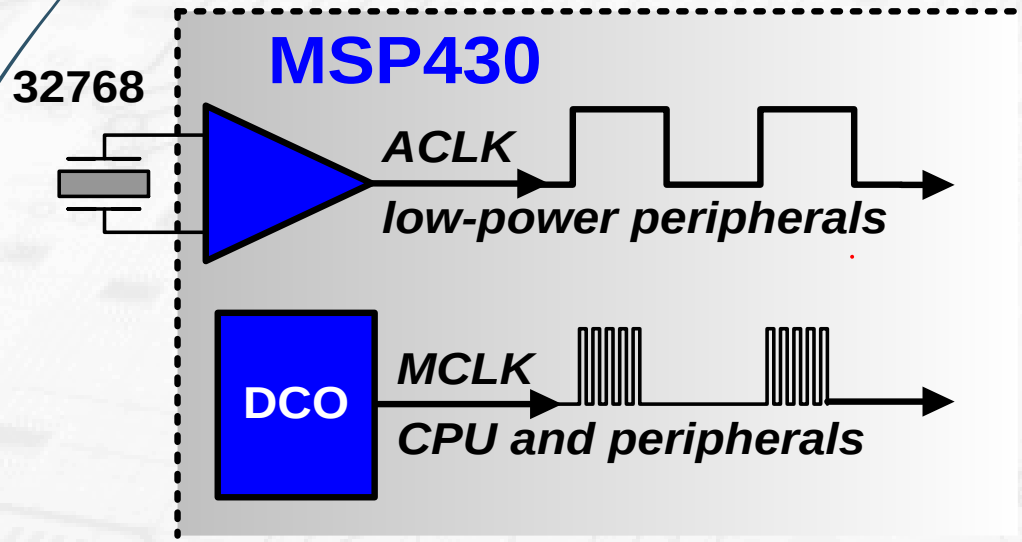

- Minimize active time
- Maximize time in Low Power Modes
- Interrupt driven performance on-demand with <1µs wakeup time
- Always-On, Zero-Power Brownout Reset (BOR)

# Low Power Modes (LPM)
## Εισαγωγή

- Η μεταγωγή σε κατάσταση χαμηλής κατανάλωσης κοστίζει *(για να ελεγχθεί η εσωτερική κατάσταση, ειδικά σε διοχέτευση):*
  - Χρόνο.
  - Ενέργεια.
- Πρέπει να αποφασιστεί αν αξίζει η αλλαγή κατάστασης.
- Μπορεί να χρησιμοποιηθεί διάγραμμα καταστάσεων ενέργειας *(power state machine)*. Κάθε κατάσταση συνδέεται στο διάγραμμα, συνδέεται με μια κατάσταση λειτουργίας του CPU.
- Η αλλαγή κατάστασης κοστίζει, γιατί απαιτείται να ελεγχθεί κατάλληλα η εσωτερική λογική της CPU *(για να μην αλλοιωθούν τα δεδομένα).*
- Η εκκίνηση πρέπει να γίνει προσεκτικά για να αποφευχθούν υπερτάσεις που θα προκαλέσουν δυσλειτουργία.

# Use Interrupts & Maximize LPM3




*Leave On the Slow Clock*

- Low power clock and peripherals interrupt CPU only for processing

*On-Demand CPU Clock*

- DCO starts immediately
- CPU processes data and quickly returns to Low Power Mode

# Low Power Mode Configuration

| Reserved | V | SCG1 | SCG0 | OSC OFF | CPU OFF | GIE | N | Z | C |
|---|---|---|---|---|---|---|---|---|---|

R2/SR

| | | |
|---|---|---|
| Active Mode | 0 0 0 0 | ~ 250uA |
| LPM0 | 0 0 0 1 | ~ 35uA |
| LPM3 | 1 1 0 1 | ~ 0.8uA |
| LPM4 | 1 1 1 1 | ~ 0.1uA |

```
bis.w   #CPUOFF,SR          ; LPM0
```

LPM in Assembly

# ULP is Easy!

Using Low Power Modes are easy

```
void main(void)
{
    WDT_init();  // initialize Watchdog Timer
    while(1)
    {
        __bis_SR_register(LPM3_bits + GIE);     // Enter LPM3, enable interrupts
        activeMode();                           // in active mode. Do stuff!
    }
}

#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer (void)
{
     __bic_SR_register_on_exit(LPM3_bits);      // Clear LPM3 bits from 0(SR), Leave LPM3, enter active mode
}
```

# Low Power Modes (LPM)
## MSP430

| Operating Mode | CPU (MCLK) | SMCLK | ACLK | RAM Retention | BOR | Self Wakeup | Interrupt Sources |
|---|---|---|---|---|---|---|---|
| Active | ☒ | ☒ | ☒ | ☒ | ☒ | ☒ | Timers, ADC, DMA, WDT, I/0, External Interrupt, COMP, Serial, RTC, other… |
| LPM0 | | ☒ | ☒ | ☒ | ☒ | ☒ | |
| LPM1 | | ☒ | ☒ | ☒ | ☒ | ☒ | |
| LPM2 | | | ☒ | ☒ | ☒ | ☒ | |
| LPM3 | | | ☒ | ☒ | ☒ | ☒ | |
| LPM3.5 | | | | | ☒ | ☒ | External Interrupt, RTC |
| LPM4 | | | | ☒ | ☒ | | External Interrupt |
| LPM4.5 | | | | | ☒ | | External Interrupt |

# Low Power Modes (LPM)
## MSP430

```
main()
{
   initGpio();
   initClocks();
   initTimers();

  while(1){
     _low_power_mode_3();
     filter();
     }
}


#pragma vector = TIMER1_A0
__interrupt ISR()
{
  getSample();
   _low_power_mode_off_on_exit();
} // Return from interrupt (RETI)
```


- Executing LPM3 function puts the processor standby
- Unless an interrupt occurs, CPU will stay asleep
- Since ISR exits from LPM, we need additional code (such as a while{} loop)

- An interrupt wakes the CPU
- Status Register (SR) is saved to stack (including LPM bits)
- Exiting ISR routine:
  - 'exit' fcn modifies saved SR (clearing LPM) before restore
  - RETI instruction restores SR from stack
  - With LPM "off", CPU returns to instruction after LPM intrinsic; e.g. filter()

# Low Power Modes (LPM)
## MSP430

**NO LPM:**
**If MCLK = 25MHz -> 25 MPIS ? And**
**0,18mA*25= 4,5mA perpetually -> 4,5mAh/hour**
**With 1000mAh battery -> ~10 days operation**

**With LPM:**
**Assume ~1000 instructions per repeat (if time stuck ==1)**
**Then 1000 instructions/25MHz -> 40 μSec!!**
**Αρα 4,5mA για 40 μsec και μετά 1,9μA για 999960μsec!!**

**With 1000mAh battery -> ~?? Days**
**$(4,5mA*40*10^{-6}sec+1,9*10^{-3}mA*999960*10^{-6}sec)/1sec$**
**$=\sim 2080*10^{-6}mA=\sim 2*10^{-3}mA$ average -> $2*10^{-3}mAh$/hour**
**Δηλ. 1000mAh/ $2*10^{-3}mA$ = 500.000hours= 57years!!!**

# Low Power Modes (LPM)
## MSP430

| Mode | | G2xx | F5xx | FR57xx | FR58xx FR59xx |
|---|---|---|---|---|---|
| Performance (max) | | 16 MHz | 25 MHz | 24 MHz (FRAM at 8MHz) | 16 MHz (FRAM at 8MHz) |
| Flex Unified Memory | | No | No | FRAM (16K) | FRAM (64K) |
| Active | AM | 230 µA (1MHz | 180 µA/MHz | 100 µA/MHz | <100 µA/MHz |
| Standby | LPM3 | 0.7 µA | 1.9 µA | 6.3 µA | 0.7 µA |
| RTC | LPM3.5 | | 2.1 µA | 1.5 µA | 0.4 µA |
| Off | LPM4 | 0.1 µA | 1.1 µA | 5.9 µA | 0.6 µA |
| | LPM4.5 | | 0.2 µA | 0.3 µA | 0.1 µA |
| Wake-up from | Standby | 1.5 µs | 3.5 µs or 150 µs | 78 µs | <10 µs |
| | Off | - | 2000 µs | 310 µs | 150 µs |

The LPM's are great, but how do I get there?

# Low Power Modes (LPM)
## MSP430

| Enter LPMx | C Compiler Intrinsic | Writing to SR with Intrinsic |
|---|---|---|
| LPM0 | _low_power_mode_0(); | _bis_SR_register( GIE + LPM0_bits ); |
| LPM1 | _low_power_mode_1(); | _bis_SR_register( GIE + LPM1_bits ); |
| LPM2 | _low_power_mode_2(); | _bis_SR_register( GIE + LPM2_bits ); |
| LPM3 | _low_power_mode_3(); | _bis_SR_register( GIE + LPM3_bits ); |
| LPM4 | _low_power_mode_4(); | _bis_SR_register( GIE + LPM4_bits ); |

- As written, both intrinsic functions *enable interrupts* and the associated *low-power mode*
- bis (and bic) instructions mimic assembly language:
  - bis = bit set
  - bic = bit clear
- bis/bic intrisics allows greater flexibility in selecting bits to set/clear

How do I exit LPMx?
Can I automatically reenter LPM?