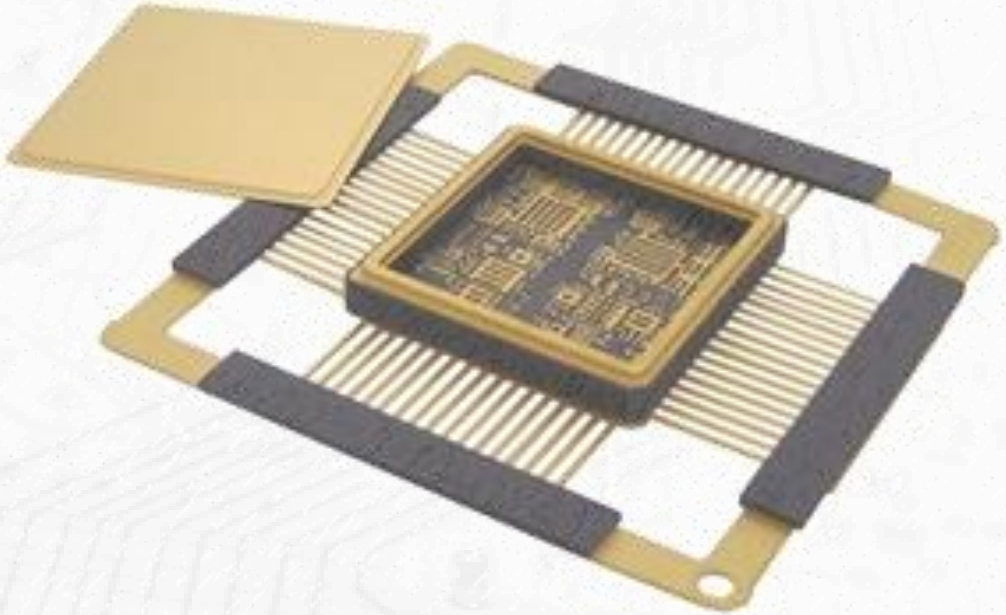
ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
& ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ



Μάθημα 6ο

Περιφερειακά Μικροελεγκτών
Bitwise operations

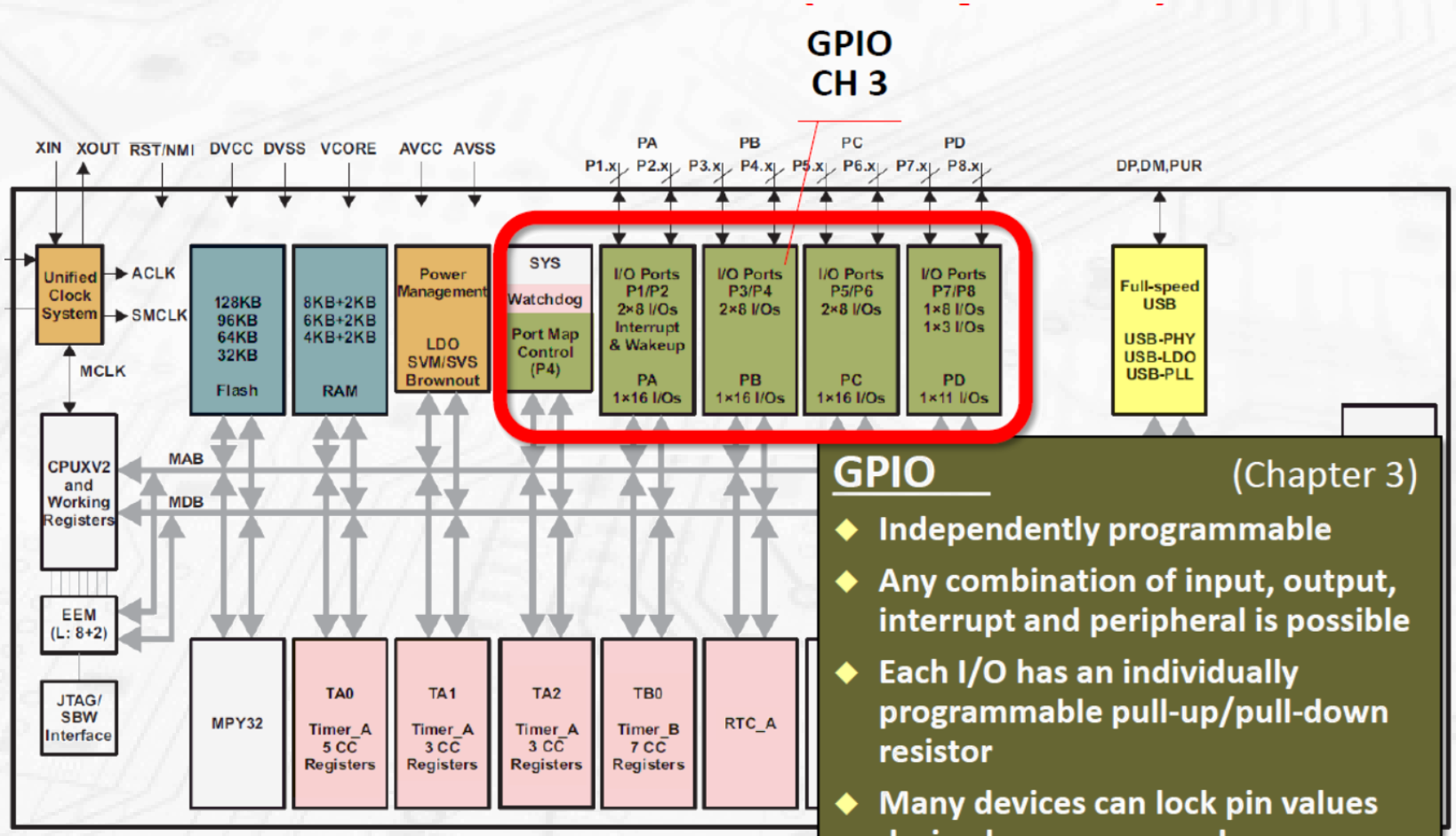
Ενσωματωμένα Συστήματα

Διδάσκον: Γρηγόριος Δουμένης

Διδακτικό Υλικό: Γρηγόριος Δουμένης, Νικόλαος Γιαννακέας, Ιωάννης Μασκλαβάνος

Αναπτυξιακό Σύστημα MSP430

Αρχιτεκτονική-GPIOs



'F5529 block diagram

GPIO (Chapter 3)

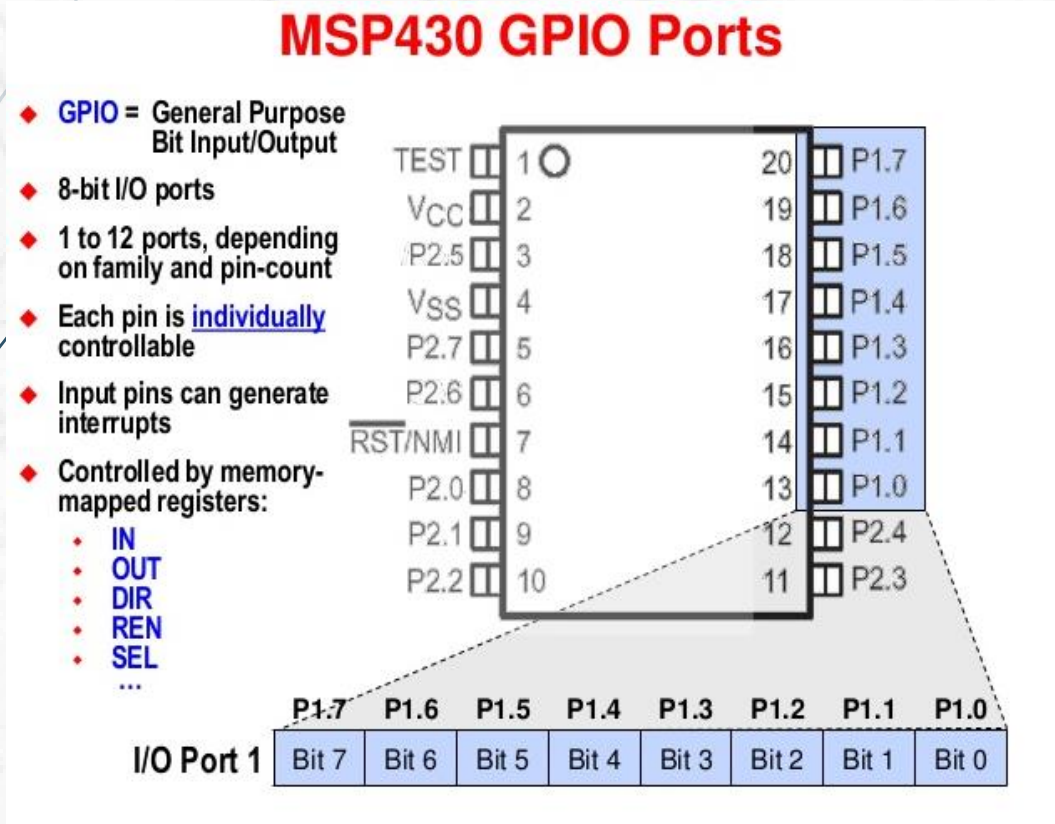
- Independently programmable
- Any combination of input, output, interrupt and peripheral is possible
- Each I/O has an individually programmable pull-up/pull-down resistor
- Many devices can lock pin values during low-power modes
- Some devices support touch-sense capability built into the pins

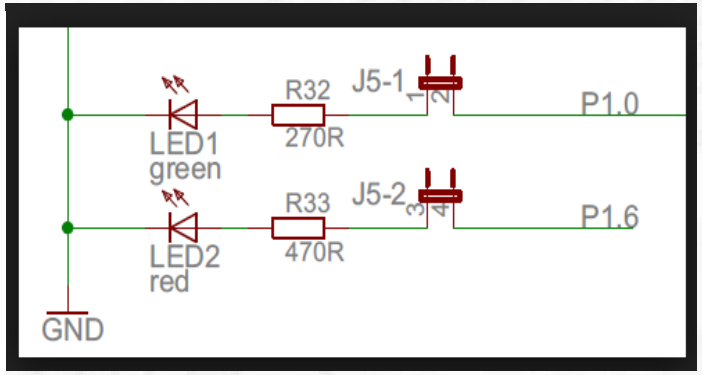
Digital i/o

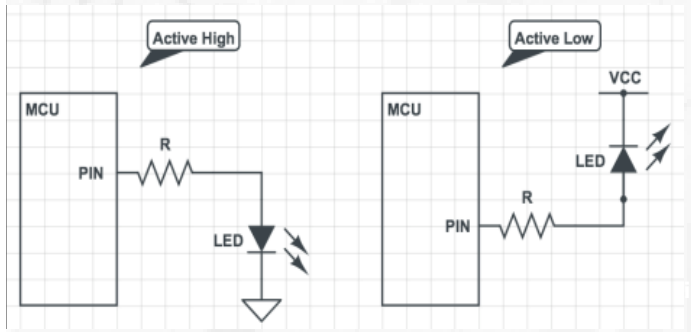
- DSP MSP430 devices have up to eight digital I/O ports implemented, P1 to P8. Each port has eight I/O pins.
- Every I/O pin is individually configurable for input or output direction, and each I/O line can be individually read or written to.
- Ports P1 and P2 have interrupt capability.
- The digital I/O features include:
  - Independently programmable individual I/Os
  - Any combination of input or output
  - Independent input and output data registers
  - Individually configurable pullup or pulldown resistors
- On eZ430 board, RED LED is connected to P1.0

MSP430 GPIO (1/3)

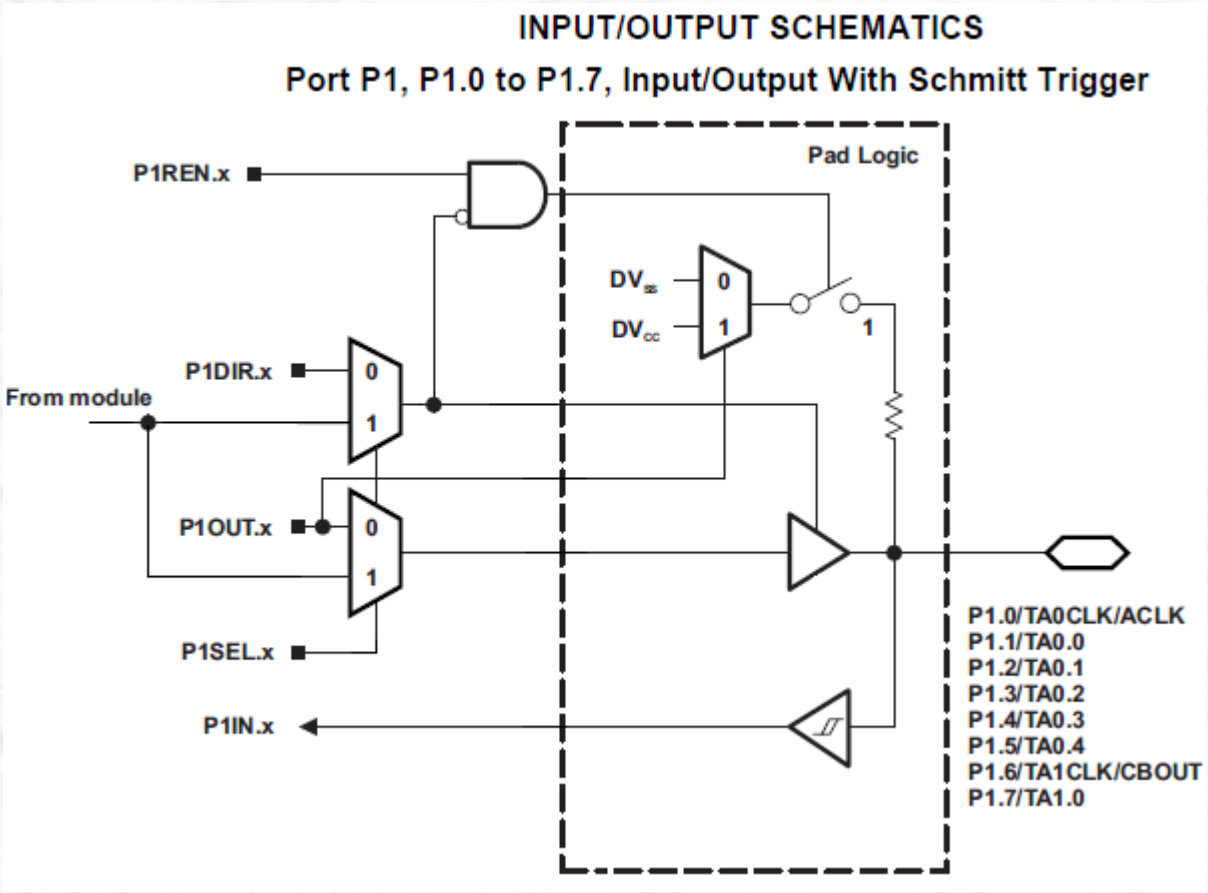
- Board configuration







GPIO Block



MSP430 F5xx GPIO registers

- The MSP430 family defines 11 I/O ports, P0 through P10, although no chip implements more than 10 of them. P0 is only implemented on the '3xx family. P7 through P10 are only implemented on the largest members (and highest pin count versions) of the '4xx and '2xx families.
- The pins are divided into 8-bit groups called "ports", each of which is controlled by a number of 8-bit registers. In some cases, the ports are arranged in pairs which can be accessed as 16-bit registers.
- The newest '5xx and '6xx families has P1 through P11, and the control registers are reassigned to provide more port pairs. Each port is controlled by the following registers.
- PxIN Port x input. This is a read-only register, and reflects the current state of the port's pins.
- PxOUT Port x output. The values written to this read/write register are driven out the corresponding pins when they are configured to output.
- PxDIR Port x data direction. Bits written as 1 configure the corresponding pin for output. Bits written as 0 configure the pin for input.
- PxSEL Port x function select. Bits written as 1 configure the corresponding pin for use by the specialized peripheral. Bits written as 0 configure the pin for general purpose I/O. Port 0 ('3xx parts only) is not multiplexed with other peripherals and does not have a P0SEL register.
- PxREN Port x resistor enable ('2xx & '5xx only). Bits set in this register enable weak pull-up or pull-down resistors on the corresponding I/O pins even when they are configured as inputs. The direction of the pull is set by the bit written to the PxOUT register.

MSP430 F5xx GPIO registers (II)

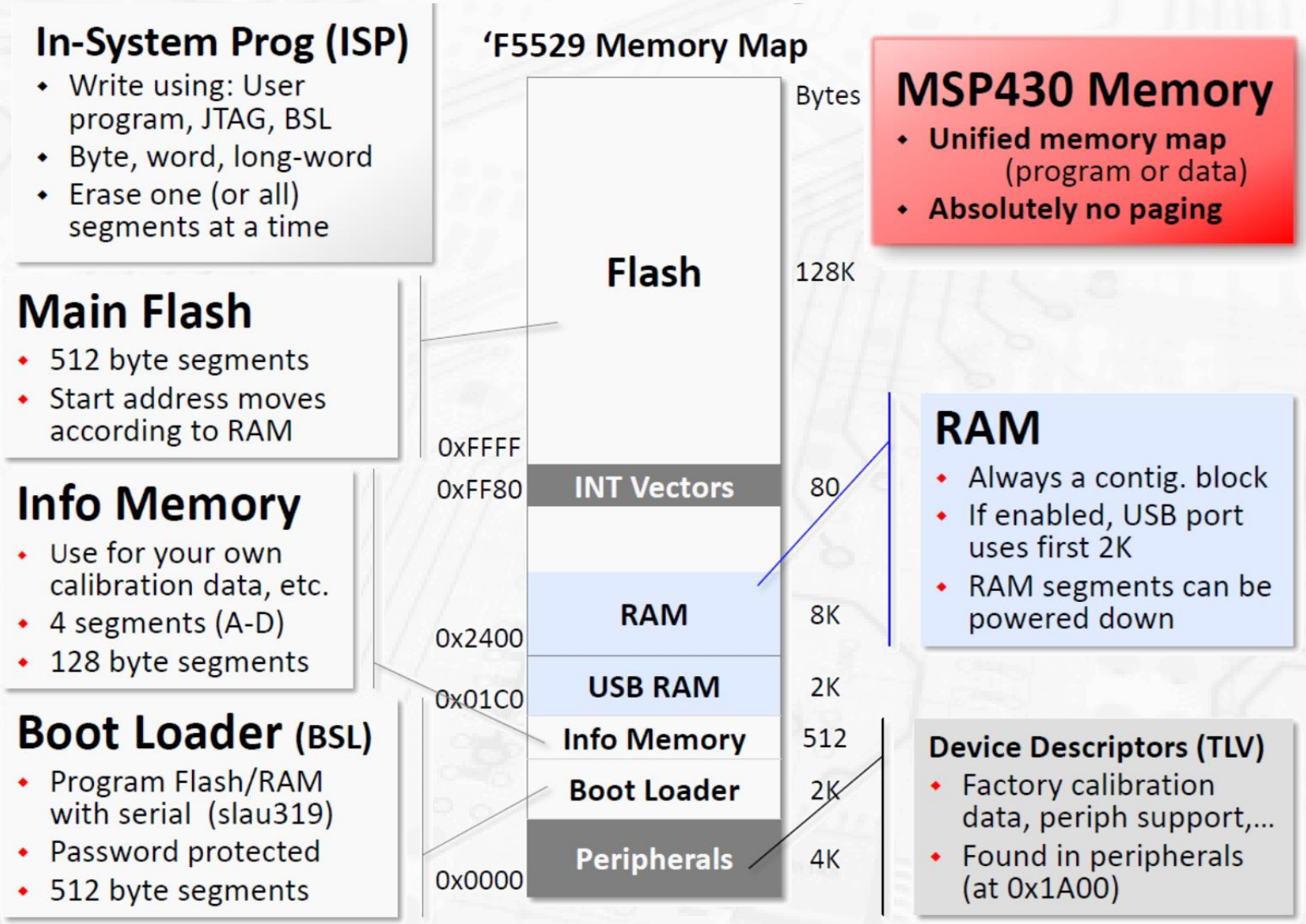
- PxDS Port x drive strength ('5xx only). Bits set in this register enable high current outputs. This increases output power, but may cause EMI.

- Ports 0–2 can produce interrupts when inputs change. Additional registers configure this ability:
- PxIES Port x interrupt edge select. Selects the edge which will cause the PxIFG bit to be set. When the input bit changes from matching the PxIES state to not matching it (i.e. whenever a bit in PxIES XOR PxIN changes from clear to set), the corresponding PxIFG bit is set.
- PxIE Port x interrupt enable. When this bit and the corresponding PxIFG bit are both set, an interrupt is generated.
- PxIFG Port x interrupt flag. Set whenever the corresponding pin makes the state change requested by PxIES. Can be cleared only by software. (Can also be set by software.)
- PxIV Port x interrupt vector ('5xx only). This 16-bit register is a priority encoder which can be used to handle pin-change interrupts. If n is the lowest-numbered interrupt bit which is pending in PxIFG and enabled in PxIE, this register reads as 2n+2. If there is no such bit, it reads as 0. The scale factor of 2 allows direct use as an offset into a branch table. Reading this register also clears the reported PxIFG flag.

- Some pins have special purposes either as inputs or outputs. (For example, timer pins can be configured as capture inputs or PWM outputs.) In this case, the PxDIR bit controls which of the two functions the pin performs when the PxSEL bit is set. If there is only one special function, then PxDIR is generally ignored. The PxIN register is still readable if the PxSEL bit is set, but interrupt generation is disabled. If PxSEL is clear, the special function's input is frozen and disconnected from the external pin. Also, configuring a pin for general purpose output does not disable interrupt generation.

MSP430
Αρχιτεκτονική-Μνήμη

MEMORY MAP



MSP430 GPIO memory map

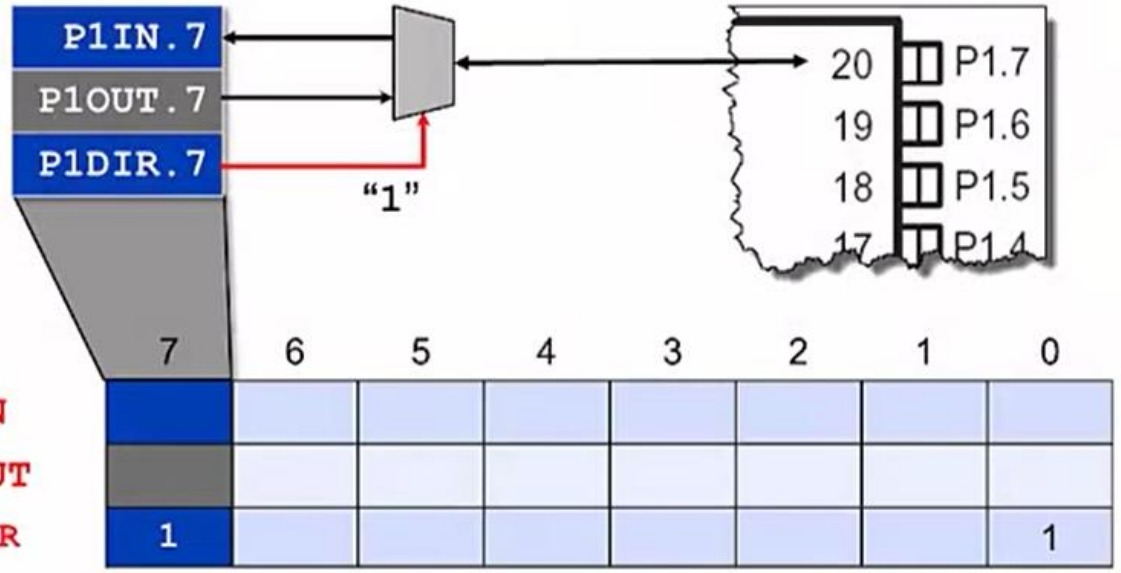
'5xx–'6xx & '0xx families

		PxIN	PxOUT	PxDIR	PxREN	PxDS	PxSEL	PxIV	PxIES	PxIE	PxIFG
PA	P1	0x200	0x202	0x204	0x206	0x208	0x20A	0x20E	0x218	0x21A	0x21C
	P2	0x201	0x203	0x205	0x207	0x209	0x20B	0x21E	0x219	0x21B	0x21D
PB	P3	0x220	0x222	0x224	0x226	0x228	0x22A				
	P4	0x221	0x223	0x225	0x227	0x229	0x22B				
PC	P5	0x240	0x242	0x244	0x246	0x248	0x24A				
	P6	0x241	0x243	0x245	0x247	0x249	0x24B				
PD	P7	0x260	0x262	0x264	0x266	0x268	0x26A				
	P8	0x261	0x263	0x265	0x267	0x269	0x26B				
PE	P9	0x280	0x282	0x284	0x286	0x288	0x28A				
	P10	0x281	0x283	0x285	0x287	0x289	0x28B				
P11		0x2A0	0x2A2	0x2A4	0x2A6	0x2A8	0x2AA				
PJ		0x320	0x322	0x324	0x326	0x328	*4 bits only; shared with JTAG pins.*				

- P1OUT = 0x01; -> Goto to address 0x202 and write bits 00000001
- Symbolic mapping: https://e2e.ti.com/cfs-file/__key/communityserver-discussions-components-files/166/msp430f5529.h

MSP430 GPIO S/W Configuration

PxDIR (Pin Direction): Input or Output

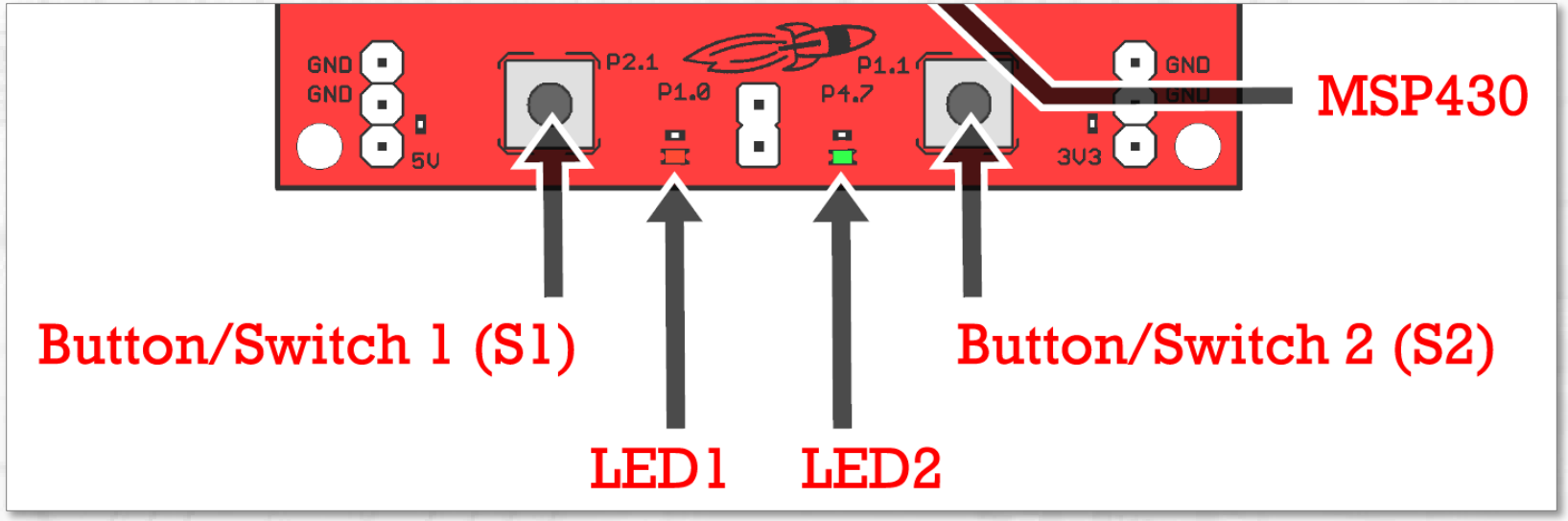


- PxDIR.y: 0 = input
  1 = output
- Register example:
  `P1DIR &= 0x81;`
- MSP430ware example:

```
#include <driverlib.h>

GPIO_setAsOutputPin( GPIO_PORT_P1, GPIO_PIN0 + GPIO_PIN7 );
```

MSP430 GPIO programming

Launchpad →	F5529	FR4133	FR5969	LED Color
LED1	P1.0	P1.0	P4.6	Red LED (with Jumper)
LED2	P4.7	P4.0	P1.0	Green LED
Button 1	P2.1	P1.2	P4.5	
Button 2	P1.1	P2.6	P1.1	



HELLO WORLD

```
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);
}
```

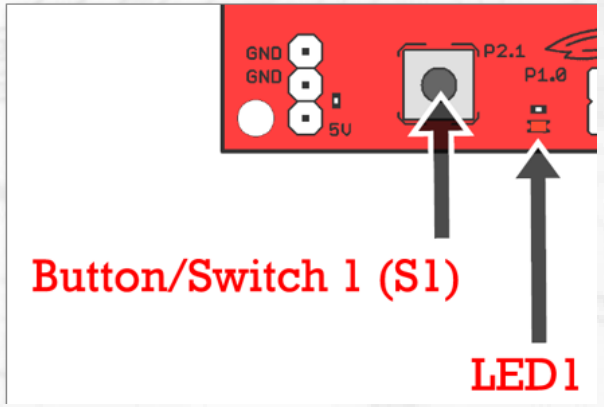
Bit manipulations

- Port Initialization (8-bit access):
- Set P1.0 as output P1DIR=0x01; (0b00000001) then
- …… (other initializations) then
- Set P1.1 as input P1DIR=0x00; (0b00000000)

- Problem P1.0=0 !

- We need to change bit 1 of P1DIR WITHOUT affecting bit 0

Launchpad →	F5529	F
LED1	P1.0	
LED2	P4.7	
Button 1	P2.1	
Button 2	P1.1	



Bit (Boolean) operations

Άλγεβρα Boole (1ο Εξαμ.!!)

- AND: C=A&B;
  A&1=A,
  A&0=0

A	B	A & B
0	0	0
0	1	0
1	0	0
1	1	1

(b) AND

- OR: C=A|B;
  A|1=1,
  A|0=A

A	B	A | B
0	0	0
0	1	1
1	0	1
1	1	1

(a) OR

- XOR: C=A^B;
  A^1=!A,
  A^0=A

A	B	A ∧ B
0	0	0
0	1	1
1	0	1
1	1	0

Table 2.7: XOR Truth Table

- Source http://www.glitovsky.com/Tutorialv0_4.pdf

Byte Boolean operations

Operations with masks

- AND: C=A&B;

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A	0	0	0	0	1	0	1	0
B	0	1	0	0	0	1	1	0
A&B	0	0	0	0	0	0	1	0

- Change bit 3 of A to 0

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A	0	0	0	0	1	0	1	0
B	X	X	X	X	0	X	1	X
Result	0	0	0	0	0	0	1	0

Byte Boolean operations

Operations with masks

- AND: C=A&B;

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A	0	0	0	0	1	0	1	0
B	0	1	0	0	0	1	1	0
A&B	0	0	0	0	0	0	1	0

- Change bit 3 of A to 0

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
A	0	0	0	0	1	0	1	0
B	X	X	X	X	0	X	1	X
Result	0	0	0	0	0	0	1	0

Set P1DIR bit to zero

Operations with masks

- Set P1.0 as output P1DIR=0x01; (0b00000001) then
- …… Set P1.1 as input P1DIR=0x00; (0b00000000)

- Problem P1.0=0 !

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1DIR	0(X)	0(X)	0(X)	0(X)	0(X)	0(X)	0(X)	1 (X)
MASK (OP)								
Result	X	X	X	X	X	X	0	1 (X)

- P1DIR= P1DIR & 0xFC; (0x11111101)

Set P1DIR bit to one

Operations with masks

- P1DIR is 0 upon reset (thus P1.1 is input)
  However, P1.x might be set as input somewhere in the code (previously)
- Set P1.0 as output P1DIR=0x01; (0bXXXXXXX1) WITHOUT affecting the other bits

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1DIR	0(X)	0(X)	0(X)	0(X)	0(X)	0(X)	0(X)	0 (X)
MASK (OP)								
Result	X	X	X	X	X	X	X	1 (X)

- P1DIR= P1DIR | 0x01; -> P1DIR |= 0x01;

Blink the LED

- Each time the loop is executed
- If (P1.0 = 0) then P1.0 =1; (?!?)

	Bit 7	Bit 6	Bit 7	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
P1OUT	X	X	X	X	X	X	X	X
MASK (OP)								
Result	X	X	X	X	X	X	X	!X

- P1OUT=

EMBEDDED SYSTEMS LAB projects

WORKFLOW

- Learn the CCS IDE / Use the MSP board
- *MSP board on loan* (home project)
- Execute simple programs (Hello world)
- Assign mid-term project (timed execution)
- Decompose into exercises:
  - Use GPIO (blink led / read button)
  - Use timers (timed events)
  - Use interrupts (timers / read button)
  - Achieve low power execution (use LP modes)

- Mid-term project+presentation -> Grade

Sources and Material

- https://training.ti.com/msp430-workshop-series
- MSP_Design_workshop.pdf
- http://www.glitovsky.com/Tutorialv0_4.pdf
- Download CCS
  http://software-dl.ti.com/ccs/esd/documents/ccs_downloads.html
  - CCS 10.x.x (win 64)
  - CCS 8.3.1 (win 32)
- Various options:
  - Download support for MSP430 only
  - No need for driverlib, Energia,…
- OR
  https://dev.ti.com/ (CCScloud, requires TI account)

Design algorithm (using GPIO and Timers):

1. Blink the RED LED every 2 secs and the Green LED every sec.
2. Start/Stop the blink process via the button
3. Double-click switches GREEN LED between hi/lo frequency
   (i.e. high freq=1Hz, low freq=2Hz)
4. Read the on-board temperature and/or voltage via the on-chip ADC

- Each step is a separate project (with potentially common code)

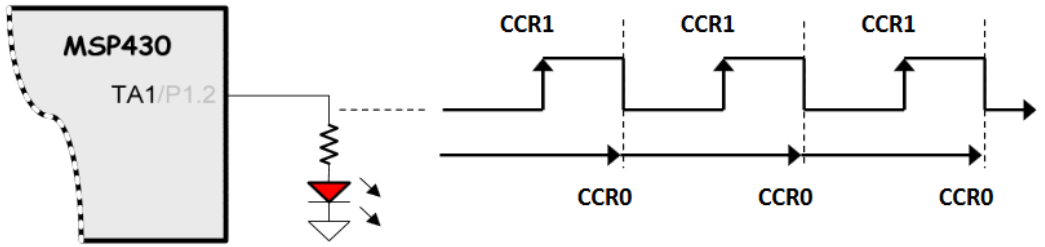
HELLO WORLD

```
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);
}
```

ΑΣΚΗΣΗ (Project – I)

- Hello world (LED blink)
- Exercises:
  - Reduce blink Frequency
  - Blink RED/Green (alternating)
  - Blink Green twice as fast

Replace Software With Hardware



Zero CPU overhead

```
CCR0 = Period;                  // Setup timer
CCTL1 = OUTMOD0_3;
CCR1 = Duty;
TACTL = TASSEL1 + MC0;          // Start timer
_BIS_SR(CPUOFF);                // No CPU
```

100% CPU overhead

```
for (;;) {
  P1OUT |= 0x04;                // Set
  delay1();
  P1OUT &= ~0x40;               // Reset
  delay2();
 }
```

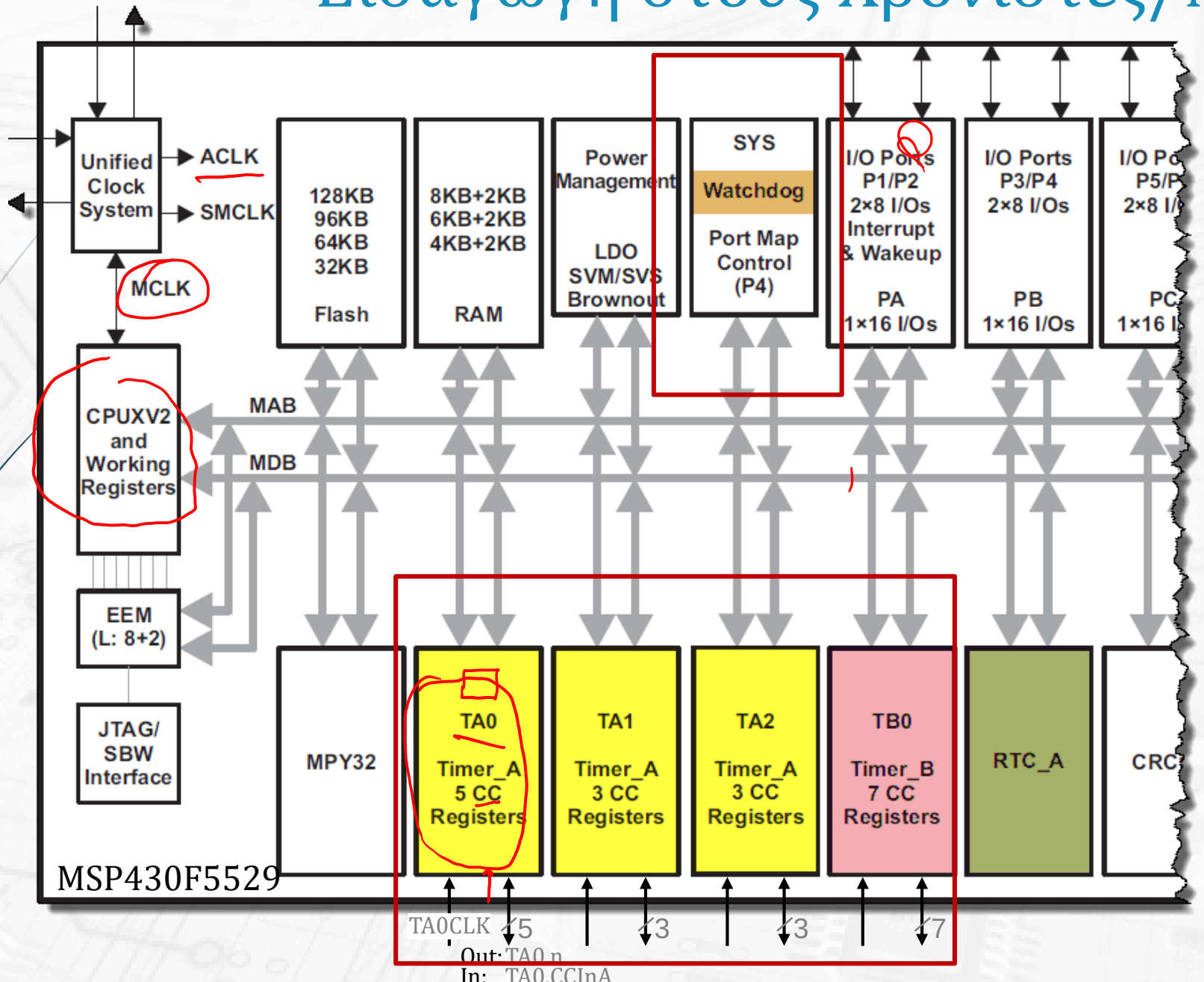
Εισαγωγή στους Χρονιστές/Μετρητές

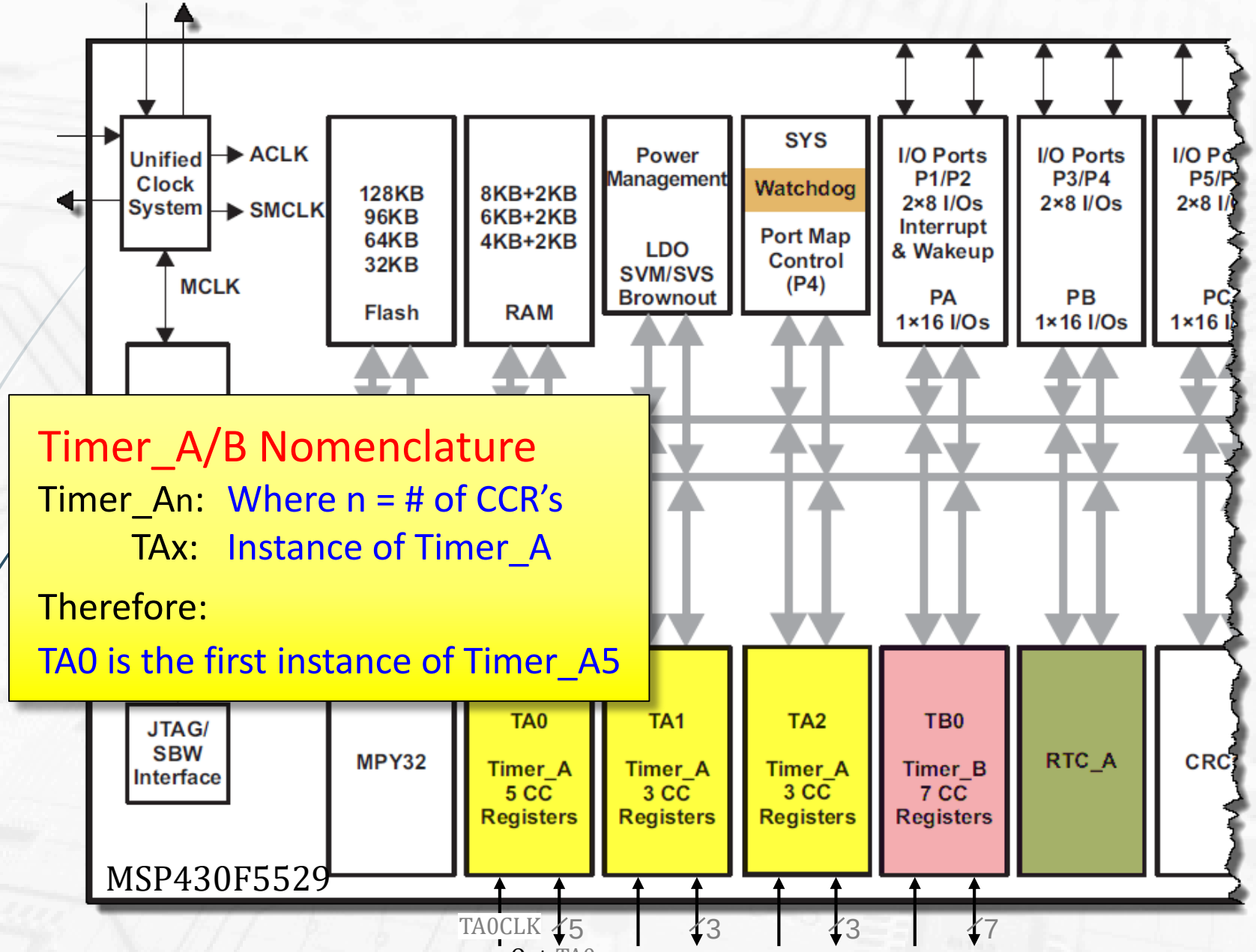
- Οι χρονιστές (Timers) είναι κυκλώματα των μικροελεγκτών τα οποία «δίνουν» ρυθμό στα υπόλοιπα κυκλώματα
- Δεν υπάρχει μόνο ένας χρονιστής αλλά περισσότεροι ώστε να υπάρχουν διαφορετικές επιλογές «ρυθμού» ανάλογα με την εφαρμογή
- Οι χρήση των χρονιστών μπορεί να βελτιστοποιήσει την κατανάλωση της ενέργειας ενός ενσωματωμένους συστήματος. Να μην σπαταλώνται πόροι σε περίπτωση που δεν χρειάζονται
- Συνδυάζονται με λειτουργίες χαμηλής ισχύος (Low Power Modes – LPM) για περεταίρω μείωση της κατανάλωσης

Εισαγωγή στους Χρονιστές/Μετρητές

```
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);
}
```

Εισαγωγή στους Χρονιστές/Μετρητές





Timer_A/B Nomenclature

Timer_An: Where n = # of CCR's

TAx: Instance of Timer_A

Therefore:

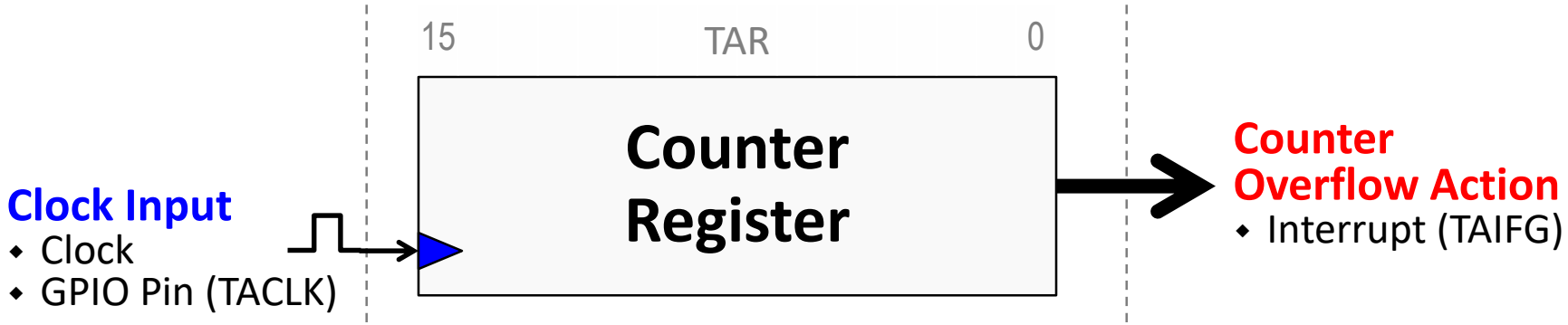
TA0 is the first instance of Timer_A5

Χρονιστές/Μετρητές
Λειτουργία

- Flexible clock sources & distribution:
  - 5 clocks from 7 sources (2 external, 5 internal)
  - Selections suitable for high-speed & low-power operations
- Wide range of operating frequency
  - 10kHz to 48 MHz
  - Fine intermediate steps with dividers & tuning
- Configurable & robust system:
  - Run-time lockable configuration
  - Failsafe mechanism with interrupts for external sources

- Overview of MSP430 Timers
- Timer Basics: How timers work
  - Counter
  - Capture
  - Compare
- Timer Details: Configuring TIMER_A
  - 1. Counter: TIMER_A_configure…()
  - 2. Capture: TIMER_A_initCapture()
    Compare: TIMER_A_initCompare()
  - 3. Clear Interrupt Flags and TIMER_A_startTimer()
  - 4. Interrupt Code (Vector & ISR)
- Differences between Timer's A and B
- Lab Exercise

Timer/Counter Basics



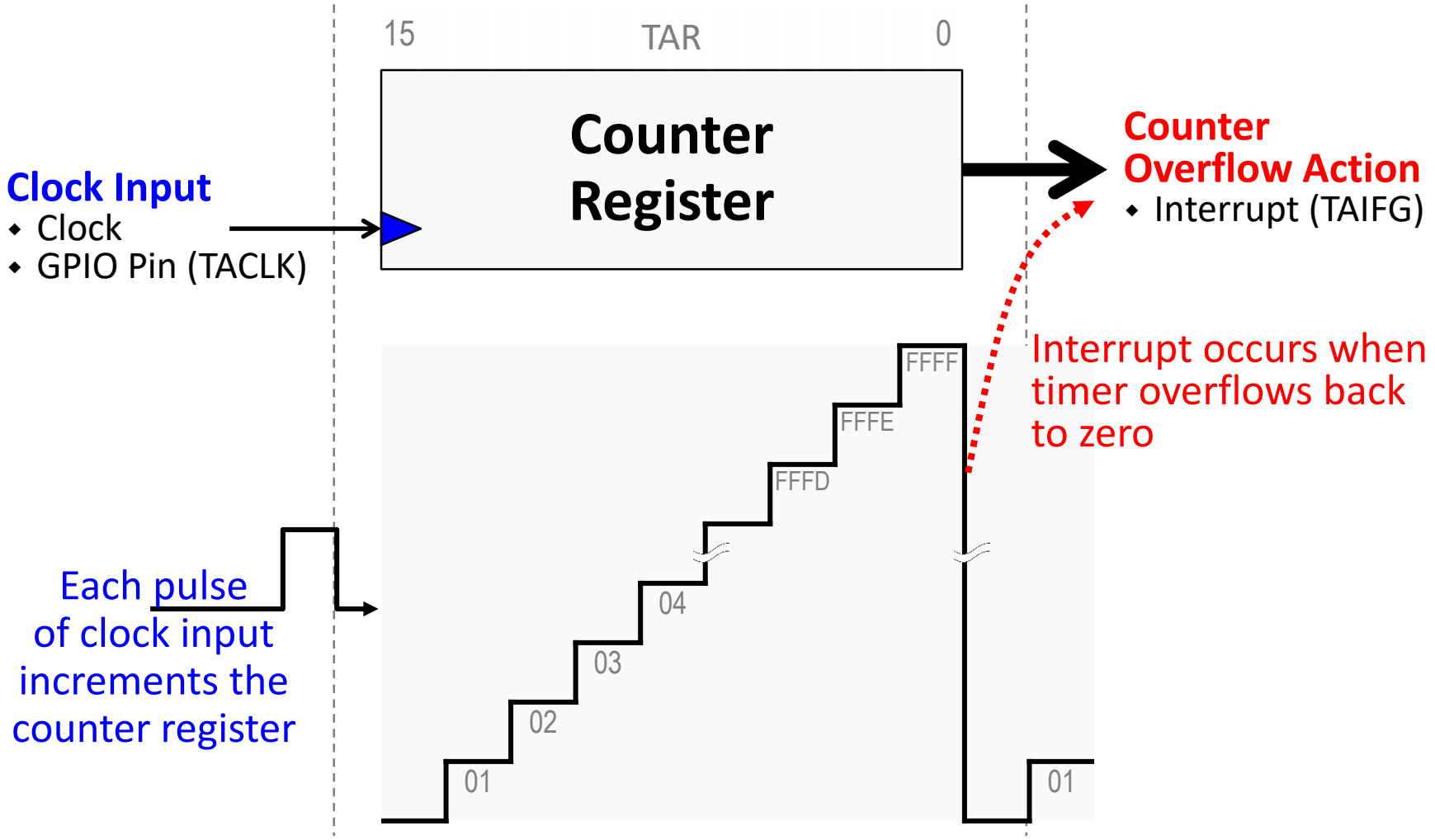
Input | **Counter** | **Action**

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

What happens on each clock input?

Timer/Counter Basics

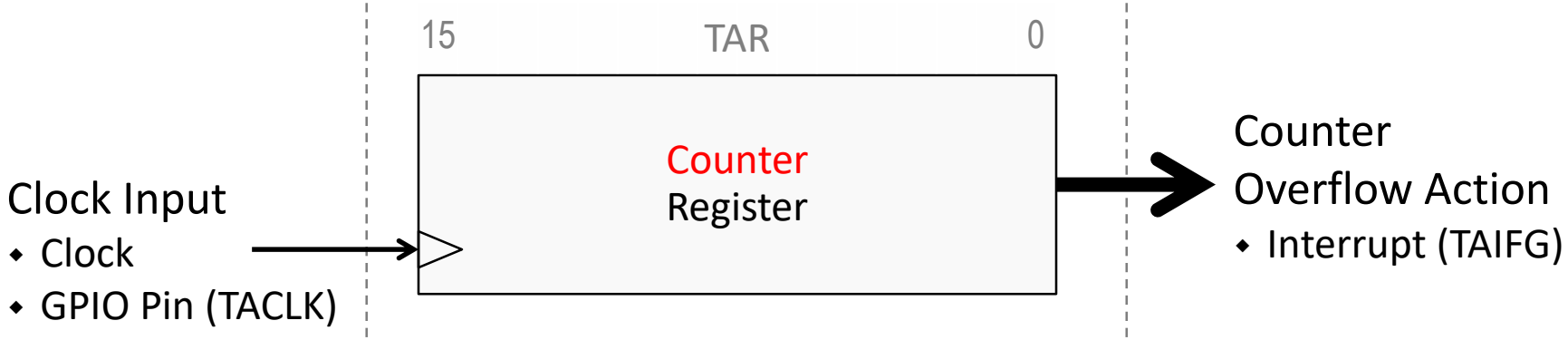


Notes

- 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?

Capture Basics

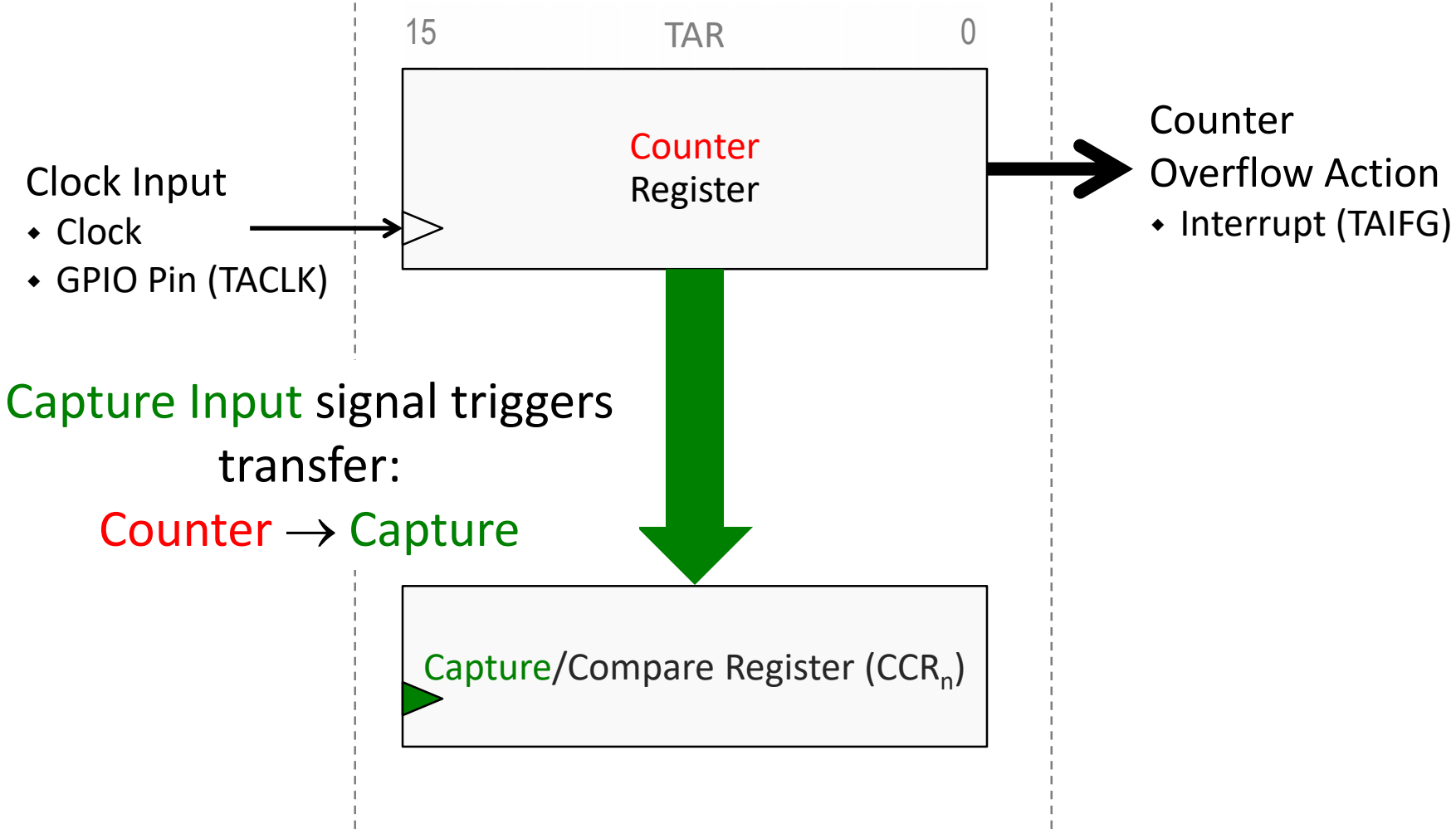


Notes

- Capture time (i.e. count value) when Capture Input signal occurs

Alternatively, use CCR for compare...

Capture Basics

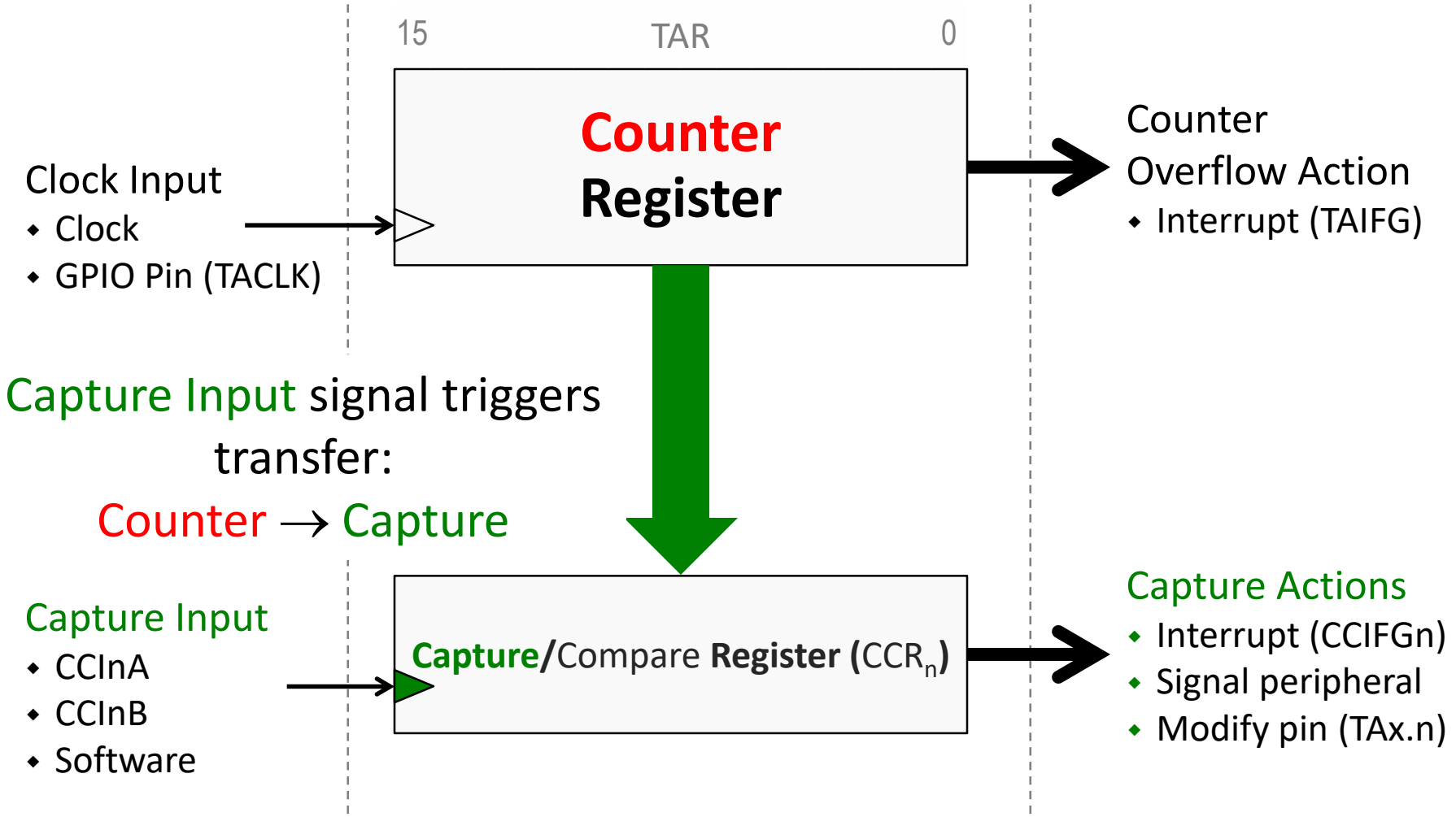


Notes

- Capture time (i.e. count value) when Capture Input signal occurs
- When capture is triggered, count value is placed in CCR and an interrupt is generated

Alternatively, use CCR for compare...

Capture Basics



Notes

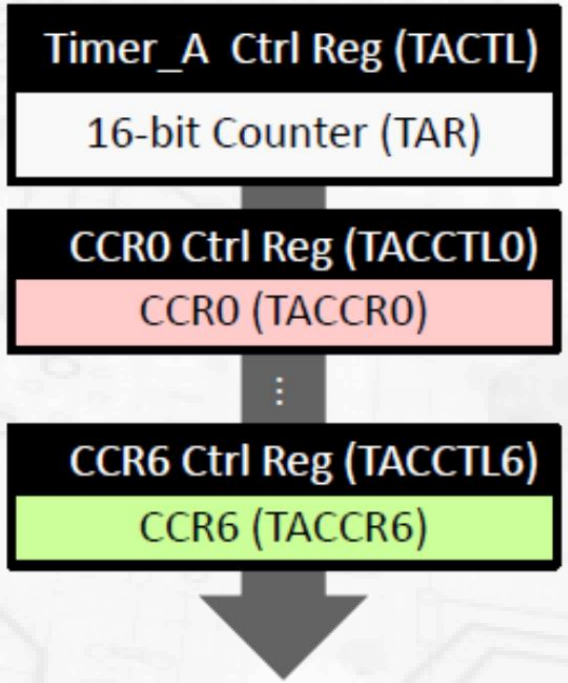
- Capture time (i.e. count value) when Capture Input signal occurs
- When capture is triggered, count value is placed in CCR and an interrupt is generated
- Capture Overflow (COV): indicates 2nd capture to CC

Alternatively, use CCR for compare...

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

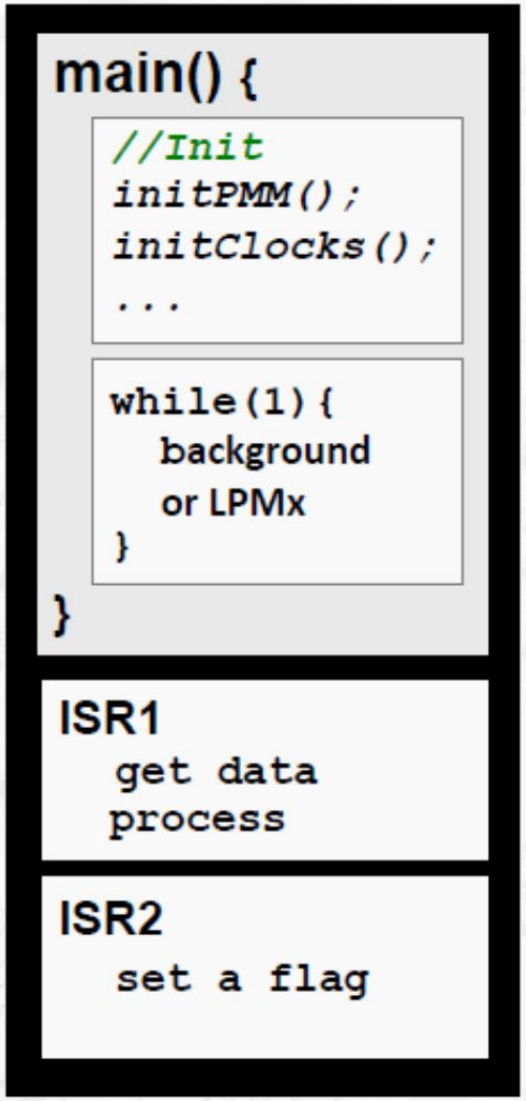


Χρονιστές/Μετρητές
Προγραμματισμός

```
#include "msp430.h"

volatile unsigned int time_tick=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
```

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

Χρονιστές/Μετρητές
Προσκήνιο – Παρασκήνιο



System Initialization

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

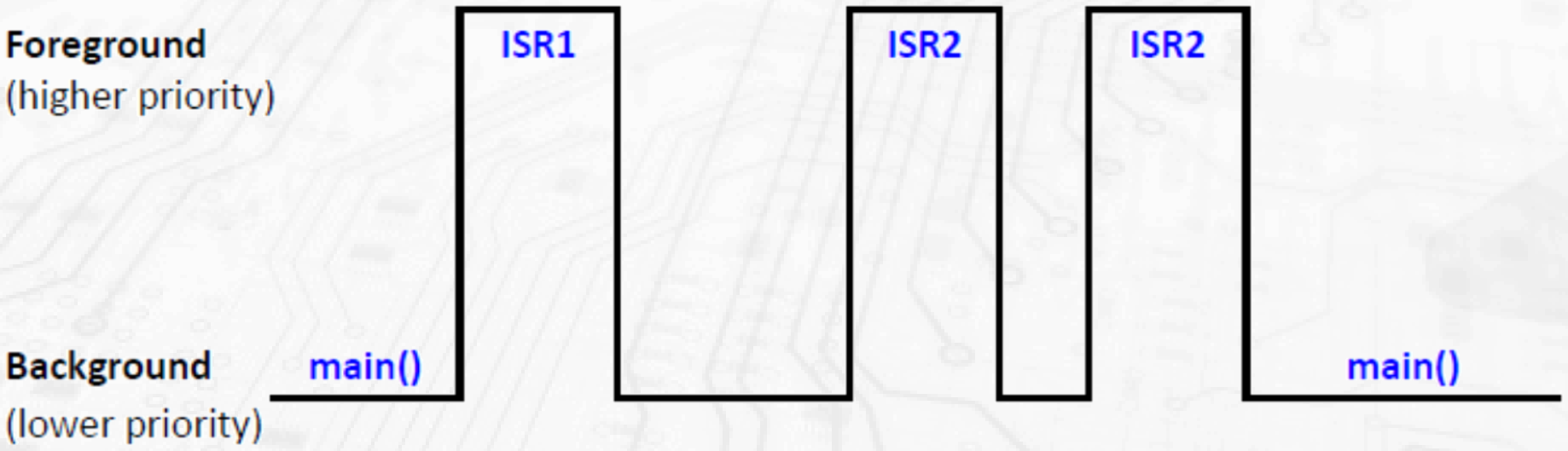
Background

- 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

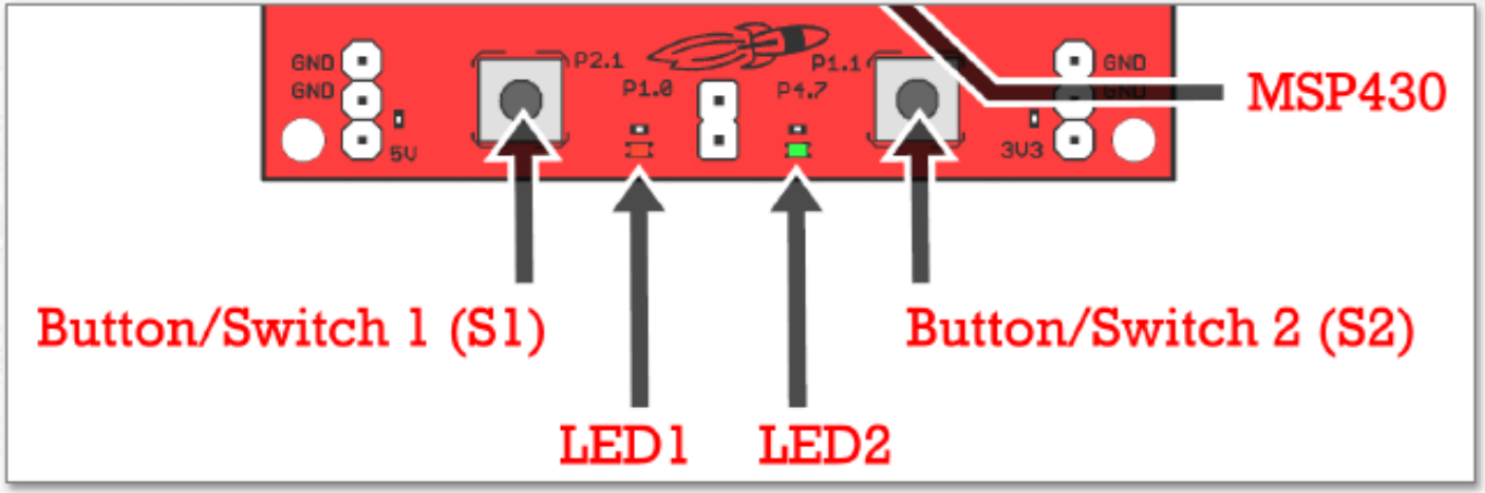


Χρονιστές/Μετρητές
Προγραμματισμός

```
  volatile unsigned time_tick =0;  // Flag (semaphore) between main and ISR

for (;;)
  {

        if (time_tick==1) {
            time_tick=0;  //clear the flag, to permit next "interrupt"
            P1OUT ^=0x01;  // toggle RED LED(every ~1 sec)
         }  //end if (time_tick)

  } //end for
return 0;
}  //end main

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

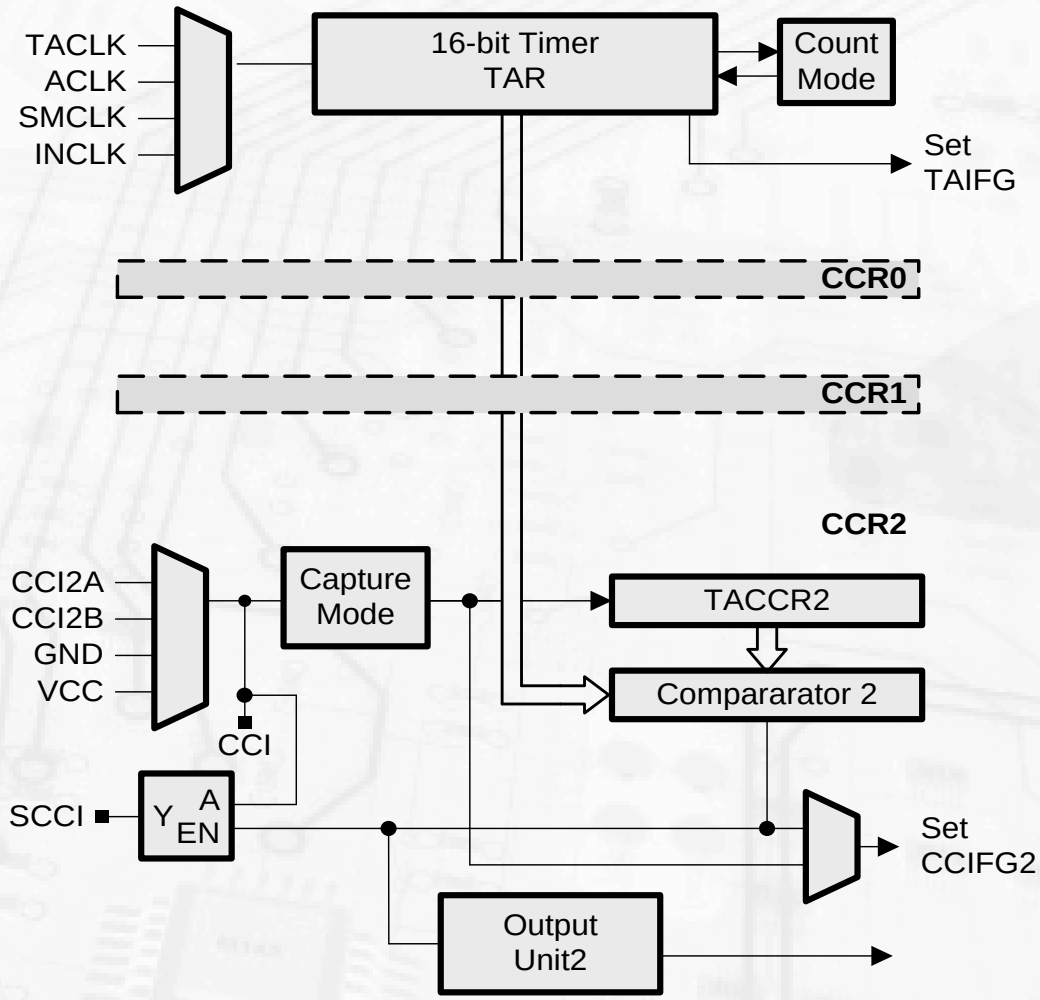
Χρονιστές/Μετρητές
Προγραμματισμός

Launchpad →	F5529	FR4133	FR5969	LED Color
LED1	P1.0	P1.0	P4.6	Red LED (with Jumper)
LED2	P4.7	P4.0	P1.0	Green LED
Button 1	P2.1	P1.2	P4.5	
Button 2	P1.1	P2.6	P1.1	



Χρονιστές/Μετρητές
MSP430 – Timer_A

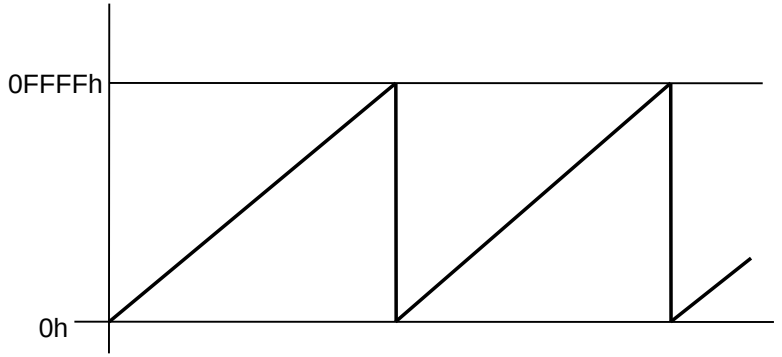
- Asynchronous 16-Bit timer/counter
- Continuous,
  up-down,
  up count modes
- Multiple capture/compare registers
- PWM outputs
- Interrupt vector register for fast decoding
- Can trigger DMA transfer
- On all MSP430s



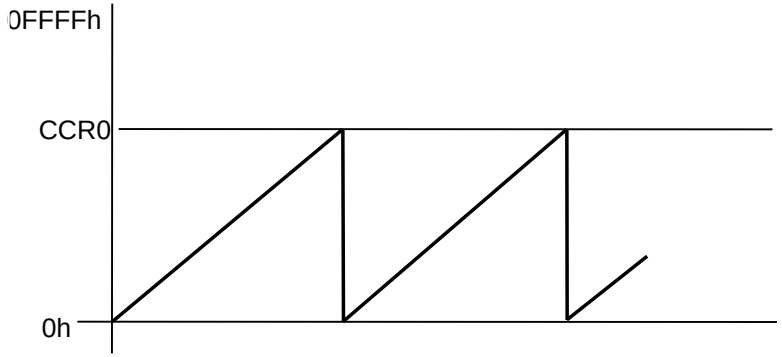
Χρονιστές/Μετρητές
MSP430 – Timer_A

Stop/Halt
Timer is halted

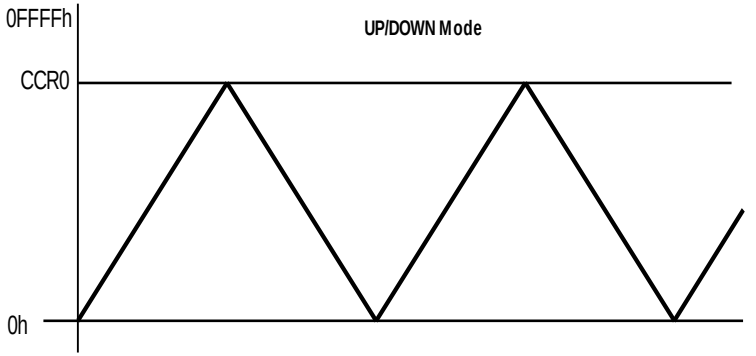
Continuous
Timer continuously counts up



Up
Timer counts between 0 and CCR0



Up/Down
Timer counts between 0 and CCR0 and 0



CCR – Count Compare Register

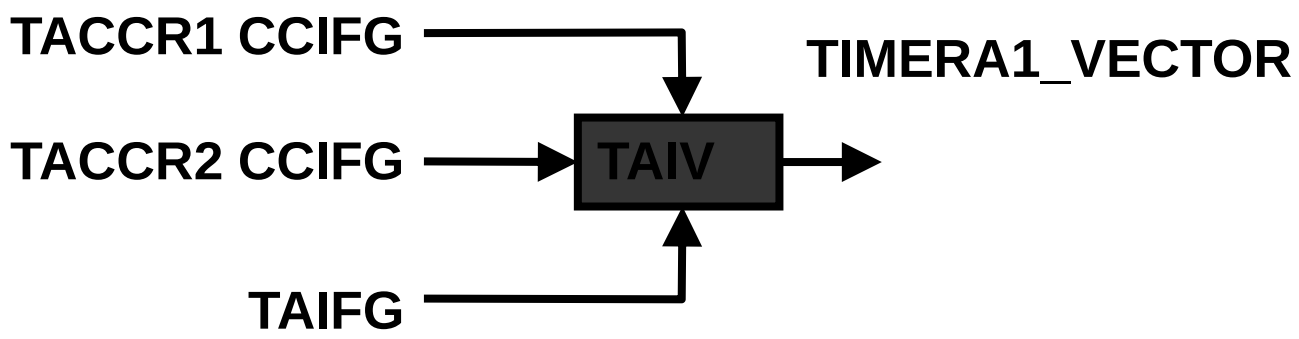
Χρονιστές/Μετρητές
MSP430 – Timer_A Interrupts

The Timer_A Capture/Comparison Register 0 Interrupt Flag (TACCR0) generates a single interrupt vector:

TACCR0 CCIFG → **TIMERA0_VECTOR**

No handler required

TACCR1, 2 and TA interrupt flags are prioritized and combined using the Timer_A Interrupt Vector Register (TAIV) into another interrupt vector

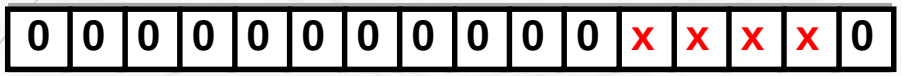


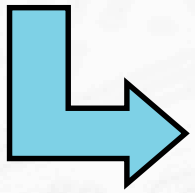
Your code must contain a handler to determine which Timer_A1 interrupt triggered

Χρονιστές/Μετρητές

MSP430 – Timer_A Interrupts

TAIV





Source	TAIV Contents
No interrupt pending	0
TACCR1 CCIFG	02h
TACCR2 CCIFG	04h
Reserved	06h
Reserved	08h
TAIFG	0Ah
Reserved	0Ch
Reserved	0Eh

IAR C code

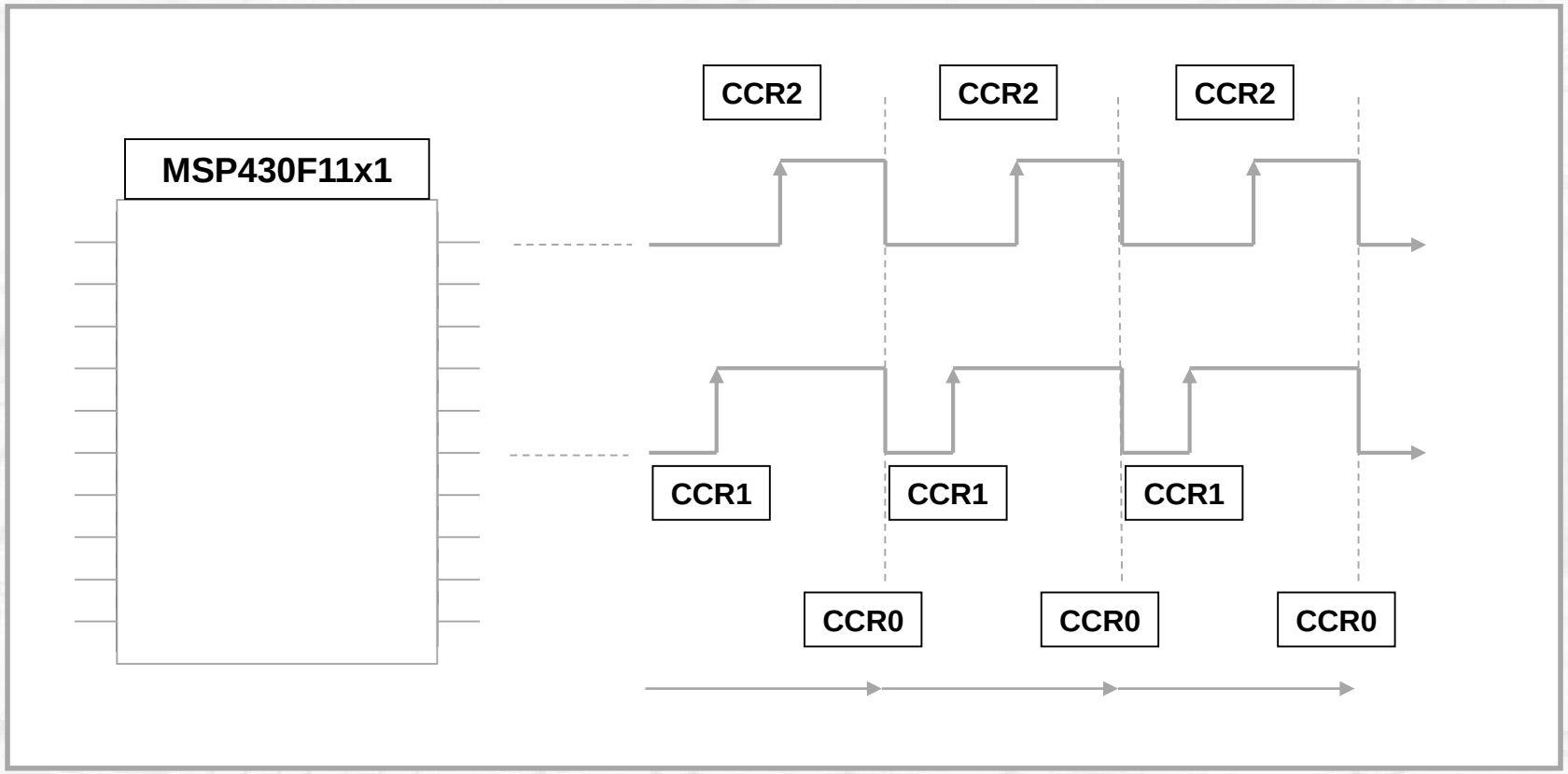
```
#pragma vector = TIMERA1_VECTOR
__interrupt void TIMERA1_ISR(void)
{
  switch(__even_in_range(TAIV,10))
  {
    case  2 :       // TACCR1 CCIFG
      P1OUT ^= 0x04; break;
    case  4 :       // TACCR2 CCIFG
      P1OUT ^= 0x02; break;
    case 10 :       // TAIFG
      P1OUT ^= 0x01; break;
   }
}
```

```
0xF814   add.w   &TAIV,PC
0xF818   reti
0xF81A   jmp     0xF824
0xF81C   jmp     0xF82A
0xF81E   reti
0xF820   reti
0xF822   jmp     0xF830
0xF824   xor.b   #0x4,&P1OUT
0xF828   reti
0xF82A   xor.b   #0x2,&P1OUT
0xF82E   reti
0xF830   xor.b   #0x1,&P1OUT
0xF834   reti
```

Assembly code

Χρονιστές/Μετρητές

MSP430 – Timer_A PWM

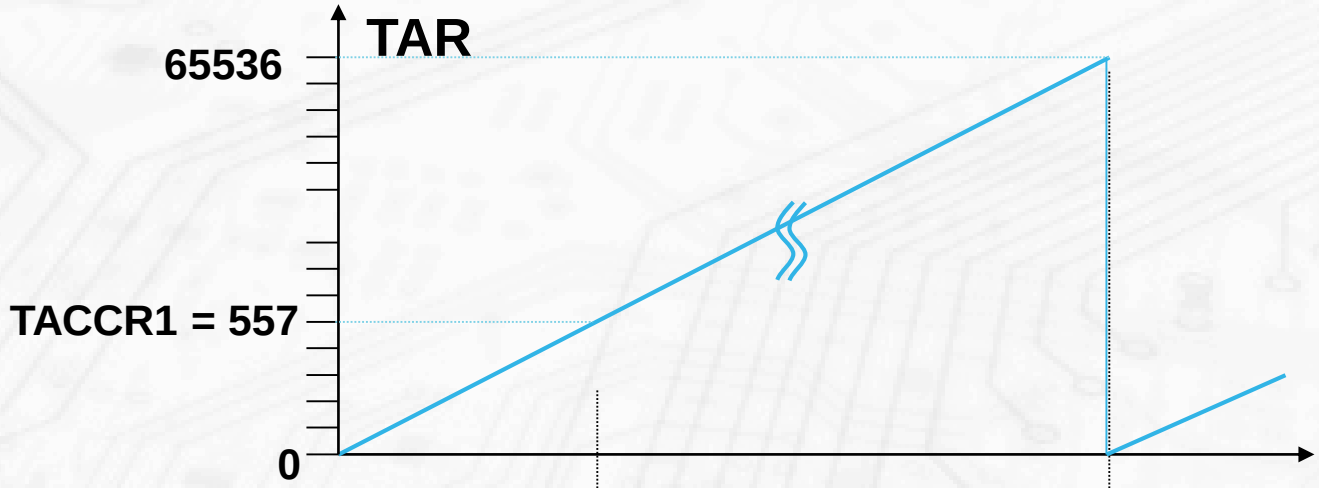


- Completely automatic
- Independent frequencies with different duty cycles can be generated for each CCR
- Code examples on the MSP430 website

Χρονιστές/Μετρητές

MSP430 – Timer_A Hardware control

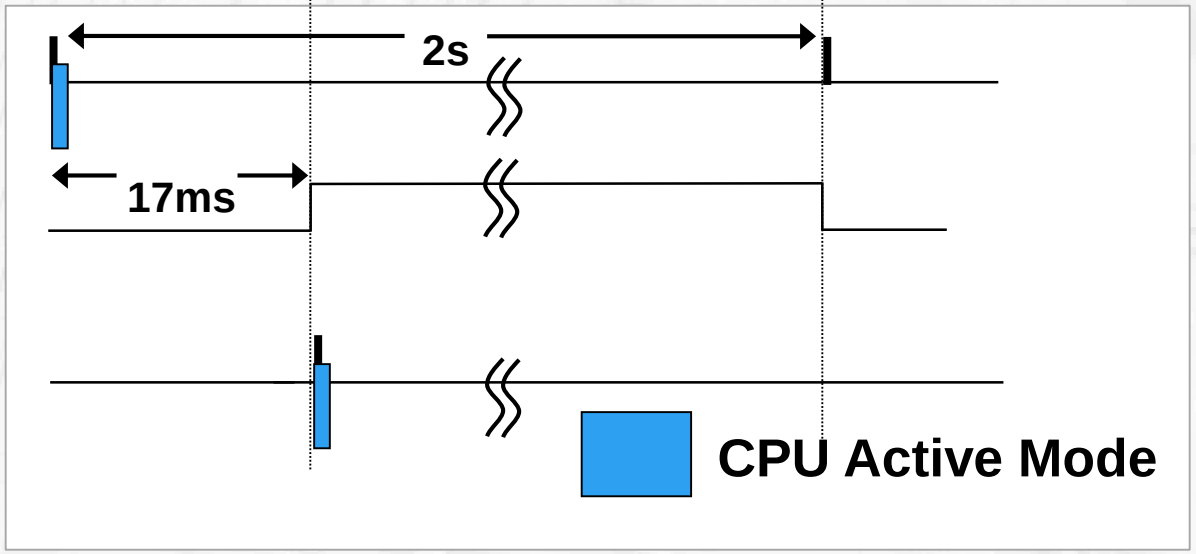
Example: ADC12



TAIFG:
Reference & ADC on

TACCR1:
Ref delay / ADC trigger

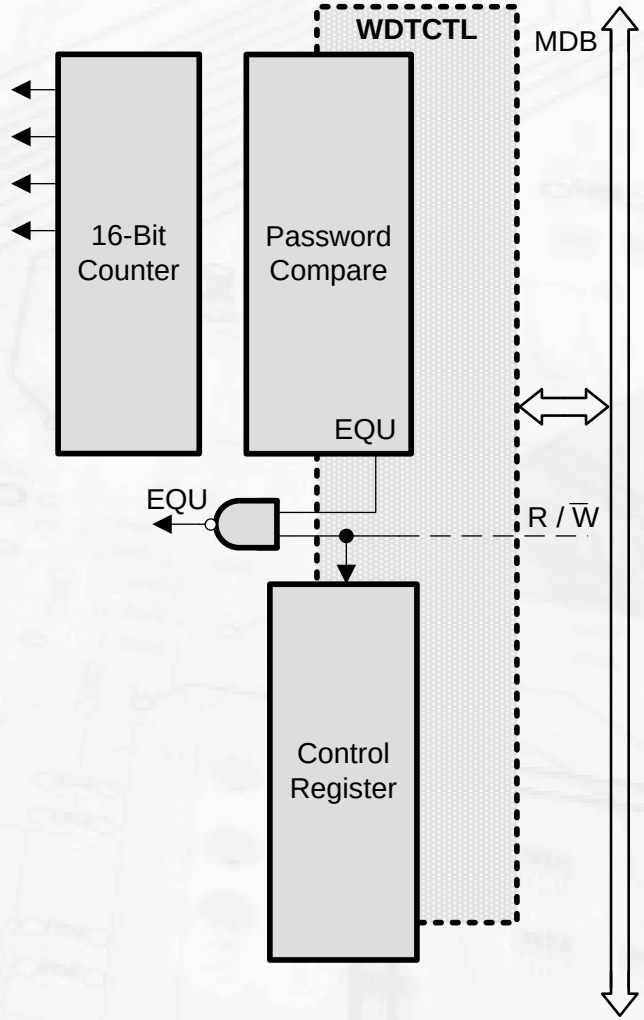
ADC12IFG:
Process ADC result
Ref/ADC Off



Χρονιστές/Μετρητές

MSP430 – Timer_A Hardware control

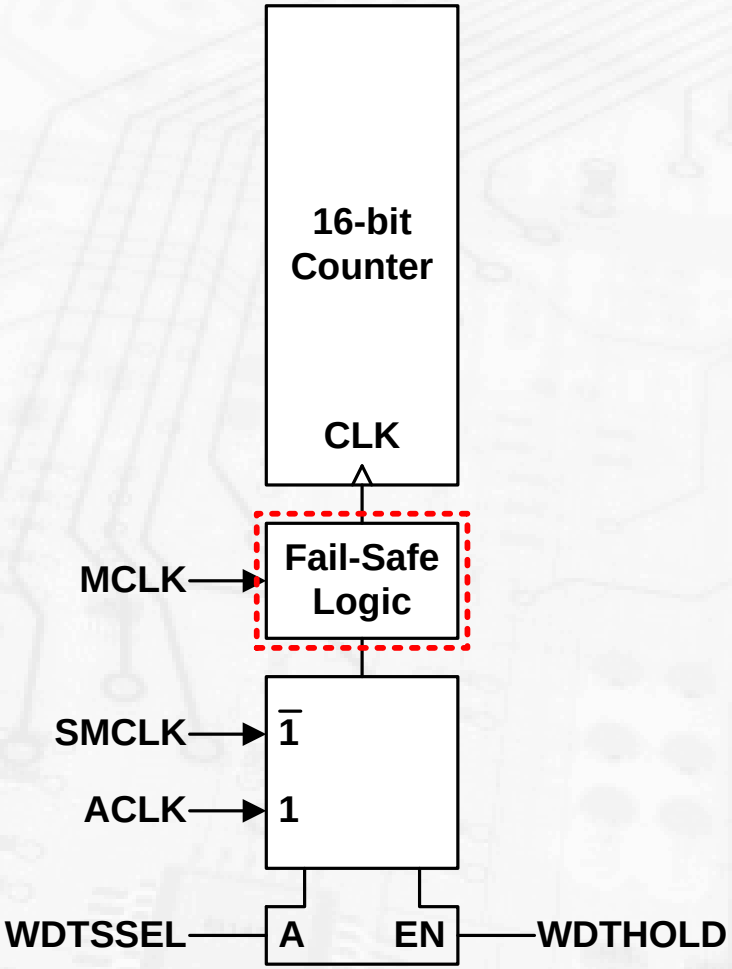
- Found on all MSP430 devices
- Two modes
  - Watchdog
  - Interval timer
- Access password protected
- Separate interrupt vectors for POR and interval timer
- Sourced by ACLK or SMCLK
- Controls RST/NMI pin mode
- WDT+ adds failsafe/protected clock



Χρονιστές/Μετρητές

MSP430 – Watchdog Timer Failsafe

- If ACLK / SMCLK fail, clock source = MCLK
  (WDT+ fail safe feature)
- If MCLK is sourced from a crystal, and the crystal fails, MCLK = DCO
  (XTAL fail safe feature)



WDT clock source …

WDT: Common Design Issues

- Program keeps resetting itself!
- Program acting wacky – how did execution get to that place?
  - Try setting interrupt near beginning of main() to see if code is re-starting
- CPU seems to freeze before even getting to first instruction
  - Is this a C program with a lot of initialized memory?
  - Generally can occur only with very large-memory versions of the device
  - Solution: Use __low_level_init() function, stop watchdog there

```
void main(void)
{
  WDTCTL = WDTPW+WDTHOLD;          // Stop the dog
  .
  .
}
```

ΑΣΚΗΣΗ (Project – II)

- Hello world (LED blink with Timer IRQ)
- Exercises:
  - Reduce blink Frequency
  - Blink RED/Green (alternating)
  - Blink Green twice as fast

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;
}
```