CPE 323
Intro to Embedded Computer Systems
Clock, Timers

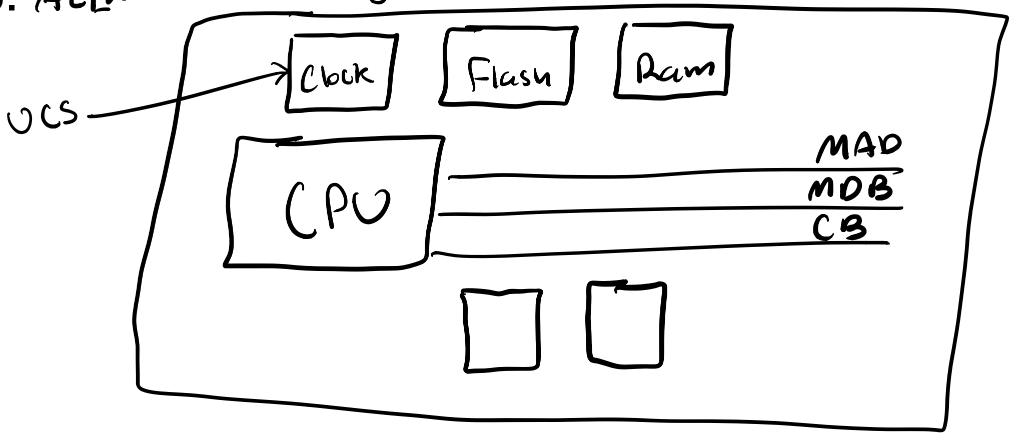
Aleksandar Milenkovic

milenka@uah.edu

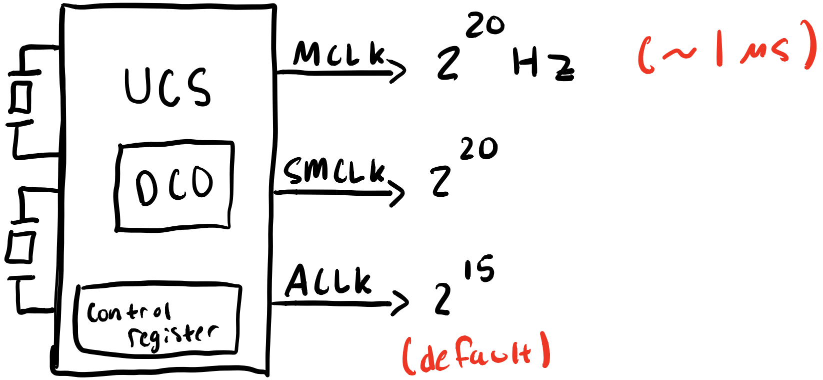
Admin

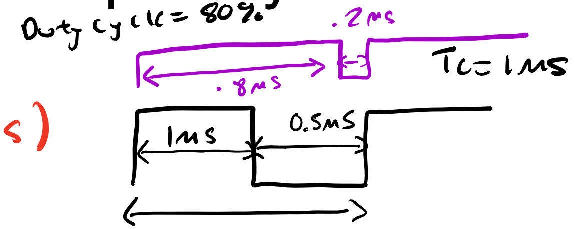
Clocks

- 1. MCLk – main clock, CPU uses
- 2. SMCLk – Sub-main clock
- 3. ACLk – Auxilary Clock



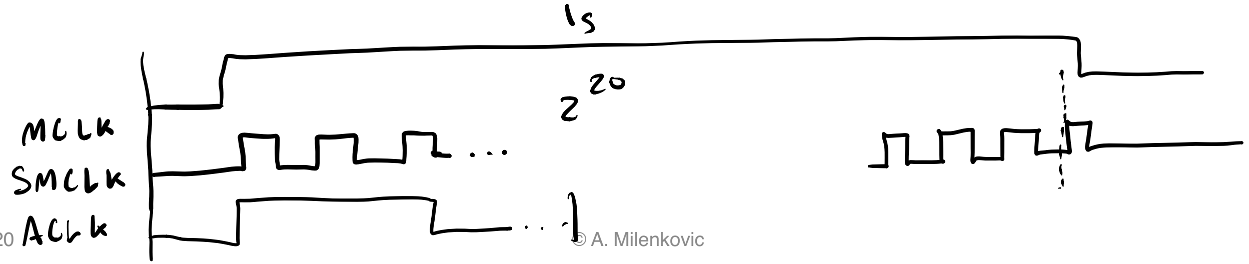
Clock Cycle Time, Frequency



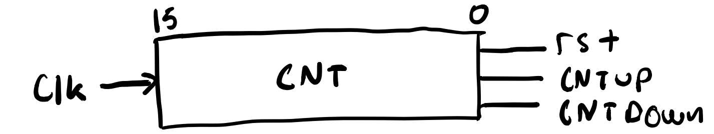


$T_c = 1\mu s$

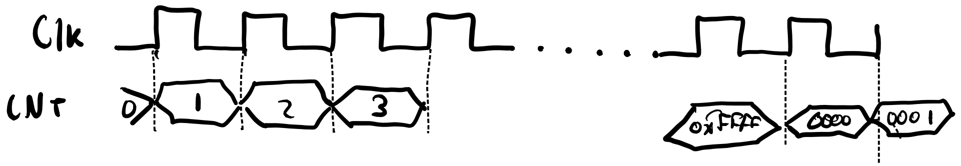
$F = \frac{1}{T_c} = 1 \times 10^6$ Hz



Counters



(CNT=0)



Watchdog Timers, Timers A/B

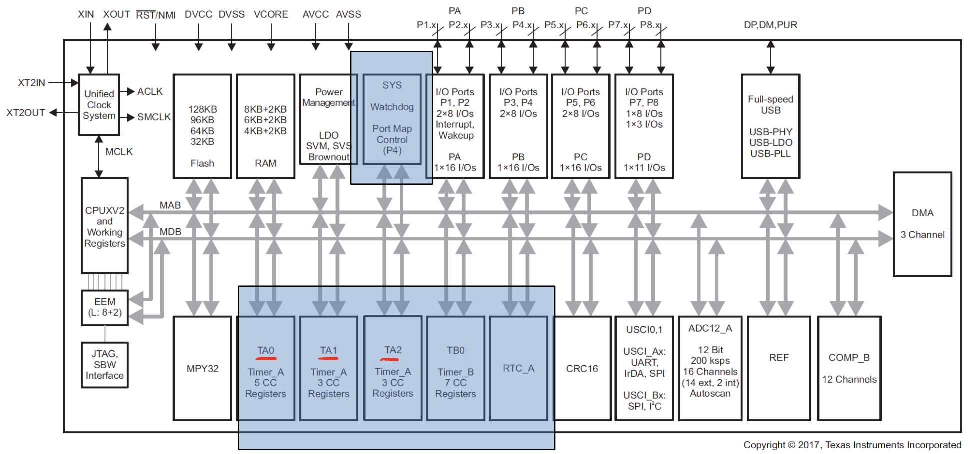

XIN
XOUT
RST/NMI
DVCC
DVSS
VCORE
AVCC
AVSS
PA
PB
PC
PD
P1.x
P2.x
P3.x
P4.x
P5.x
P6.x
P7.x
P8.x
DP,DM,PUR
XT2IN
XT2OUT
Unified Clock System
ACLK
SMCLK
MCLK
128KB 96KB 64KB 32KB Flash
8KB+2KB 6KB+2KB 4KB+2KB RAM
Power Management LDO SVM, SVS Brownout
SYS Watchdog Port Map Control (P4)
I/O Ports P1, P2 2×8 I/Os Interrupt, Wakeup PA 1×16 I/Os
I/O Ports P3, P4 2×8 I/Os PB 1×16 I/Os
I/O Ports P5, P6 2×8 I/Os PC 1×16 I/Os
I/O Ports P7, P8 1×8 I/Os 1×3 I/Os PD 1×11 I/Os
Full-speed USB USB-PHY USB-LDO USB-PLL
CPUXV2 and Working Registers
MAB
MDB
DMA 3 Channel
EEM (L: 8+2)
JTAG, SBW Interface
MPY32
TA0 Timer_A 5 CC Registers
TA1 Timer_A 3 CC Registers
TA2 Timer_A 3 CC Registers
TB0 Timer_B 7 CC Registers
RTC_A
CRC16
USCI0,1 USCI_Ax: UART, IrDA, SPI USCI_Bx: SPI, I²C
ADC12_A 12 Bit 200 ksps 16 Channels (14 ext, 2 int) Autoscan
REF
COMP_B 12 Channels
Copyright © 2017, Texas Instruments Incorporated

Figure 1-1. Functional Block Diagram – MSP430F5529IPN, MSP430F5527IPN, MSP430F5525IPN, MSP430F5521IPN

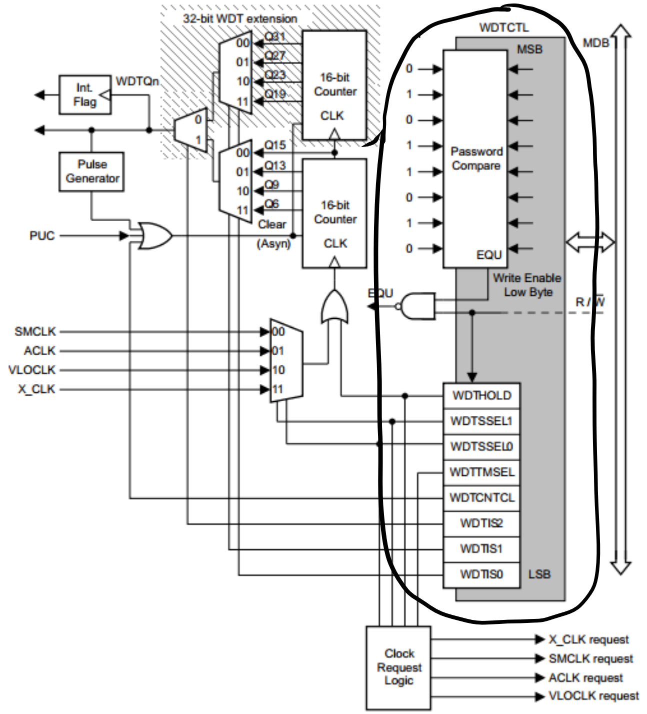
Timer A (5cc)

Timer B (7cc)

Watchdog Timer (WDT_A)

→ Watchdog mode [32 ms]

→ Regular interval mode



Watchdog Timer Registers

15	14	13	12	11	10	9	8
WDTPW							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
WDTHOLD	WDTSSEL		WDTTMSEL	WDTCNTCL	WDTIS		
rw-0	rw-0	rw-0	rw-0	r0(w)	rw-1	rw-0	rw-0

Bit	Field	Type	Reset	Description
15-8	WDTPW	RW	69h	Watchdog timer password. Always read as 069h. Must be written as 5Ah; if any other value is written, a PUC is generated.
7	WDTHOLD	RW	0h	Watchdog timer hold. This bit stops the watchdog timer. Setting WDTHOLD = 1 when the WDT is not in use conserves power. 0b = Watchdog timer is not stopped. - by default WDT is active 1b = Watchdog timer is stopped.
6-5	WDTSSEL	RW	0h	Watchdog timer clock source select 00b = SMCLK 01b = ACLK 10b = VLOCLK 11b = X_CLK; VLOCLK in devices that do not support X_CLK
4	WDTTMSEL	RW	0h	Watchdog timer mode select 0b = Watchdog mode 1b = Interval timer mode
3	WDTCNTCL	RW	0h	Watchdog timer counter clear. Setting WDTCNTCL = 1 clears the count value to 0000h. WDTCNTCL is automatically reset. 0b = No action 1b = WDTCNT = 0000h
2-0	WDTIS	RW	4h	Watchdog timer interval select. These bits select the watchdog timer interval to set the WDTIFG flag and/or generate a PUC. 000b = Watchdog clock source /(2^{31}) (18h:12m:16s at 32.768 kHz) 001b = Watchdog clock source /(2^{27}) (01h:08m:16s at 32.768 kHz) 010b = Watchdog clock source /(2^{23}) (00h:04m:16s at 32.768 kHz) 011b = Watchdog clock source /(2^{19}) (00h:00m:16s at 32.768 kHz) 100b = Watchdog clock source /(2^{15}) (1 s at 32.768 kHz) 101b = Watchdog clock source /(2^{13}) (250 ms at 32.768 kHz) 110b = Watchdog clock source /(2^{9}) (15.625 ms at 32.768 kHz) 111b = Watchdog clock source /(2^{6}) (1.95 ms at 32.768 kHz)

source clock for WDT select.

$$T_{WDT\,period} = 2^{15} \frac{1}{2^{20}} = \frac{1}{2^5} = \frac{1}{32}\,s$$

If you don't stop WDT after 32 ms...

THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Watchdog Timer Registers

IE1, Interrupt Enable Register 1

7	6	5	4	3	2	1	0
			NMIIE				WDTIE
			rw–0				rw–0

	Bits 7-5	These bits may be used by other modules. See device-specific data sheet.
NMIIE	Bit 4	NMI interrupt enable. This bit enables the NMI interrupt. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled
	Bits 3-1	These bits may be used by other modules. See device-specific data sheet.
WDTIE	Bit 0	Watchdog timer interrupt enable. This bit enables the WDTIFG interrupt for interval timer mode. It is not necessary to set this bit for watchdog mode. Because other bits in IE1 may be used for other modules, it is recommended to set or clear this bit using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 Interrupt not enabled 1 Interrupt enabled

IFG1, Interrupt Flag Register 1

7	6	5	4	3	2	1	0
			NMIIFG				WDTIFG
			rw–(0)				rw–(0)

	Bits 7-5	These bits may be used by other modules. See device-specific data sheet.
NMIIFG	Bit 4	NMI interrupt flag. NMIIFG must be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear NMIIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending
	Bits 3-1	These bits may be used by other modules. See device-specific data sheet.
WDTIFG	Bit 0	Watchdog timer interrupt flag. In watchdog mode, WDTIFG remains set until reset by software. In interval mode, WDTIFG is reset automatically by servicing the interrupt, or it can be reset by software. Because other bits in IFG1 may be used for other modules, it is recommended to clear WDTIFG by using `BIS.B` or `BIC.B` instructions, rather than `MOV.B` or `CLR.B` instructions. 0 No interrupt pending 1 Interrupt pending

1s Toggle LEDs (ASM) Using Software Delay

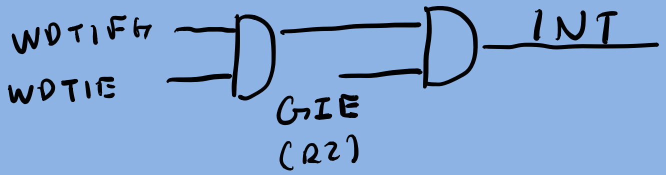
```
RESET:      mov.w   #__STACK_END,SP              ; Initialize stackpointer
StopWDT:    mov.w   #WDTPWIWDTHOLD,&WDTCTL       ; Stop watchdog timer
Setup:      bis.b   #0x06,&P2DIR                 ; Set P2.2 and P2.1 to output
                                                 ; direction (0000_0110)
            bic.b   #0x0,&P2OUT                  ; Set P2OUT to 0x0000_0100 (LEDS off)
InfLoop:    mov.w   #0xFFFF, R5                  ; Software delay (65,535*16cc/2^20 ~ 1s)
SWDelay1:   nop                                  ; 1cc (total delay is 16 cc)
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            dec.w  R5                            ; 1cc
            jnz    SWDelay1                      ; 2cc
            xor.b  #0x06, P2OUT                  ; toggle LEDs
            jmp    InfLoop                       ; goto InfLoop
; Stack Pointer definition
             .global__STACK_END
            .sect   .stack
; Interrupt Vectors
             .sect  ".reset"                     ; MSP430 RESET Vector
            .short  RESET
```

1s Toggle LEDs (ASM) Using WDT (Polling)

```
RESET:      mov.w   #__STACK_END,SP           ; Initialize stackpointer
SetWDT:     mov.w   #WDT_ADLY_1000, &WDTCTL   ; Set watchdog timer, 1 second, int. mode
Setup:      bis.b   #0x06, &P2DIR             ; Set P2.2 and P2.1 to output
                                              ; Direction (0000_0110)
            bic.b   #0x0, &P2OUT              ; Set P2OUT to 0x0000_0100 (LEDS off)

WaitWDT:    bit.b   #WDTIFG, &IFG1            ; Test WDTIFG bit (is it set or no)
            jz      WaitWDT                   ; If zero, wait
            xor.b   #0x06,&P2OUT              ; Period expired, toggle LEDs
            bic.b   #WDTIFG, &IFG1            ; Clear WDTIFG
            jmp     WaitWDT                   ; Go to WaitWDT

; Stack Pointer definition
            .global __STACK_END
            .sect   .stack
; Interrupt Vectors
            .sect   ".reset"                  ; MSP430 RESET Vector
            .short  RESET
```

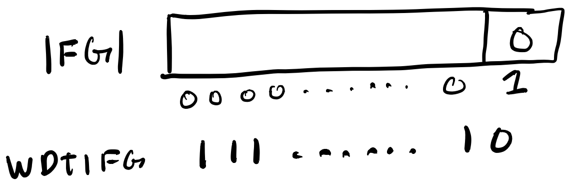
1s Toggle LEDs (ASM) Using WDT (ISR)

```
RESET:      mov.w    #__STACK_END,SP              ; Initialize stackpointer
SetWDT:     mov.w    #WDT_ADLY_1000, &WDTCTL      ; Stop watchdog timer
Setup:      bis.b    #0x06, &P2DIR                ; Set P2.2 and P2.1 to output
                                                  ; Direction (0000_0110)
            bic.b    #0x0, &P2OUT                 ; Set P2OUT to 0x0000_0100 (LEDS off)
            bis.b    #WDTIE, &IE1                 ; Enable interrupts
            nop
            bis.b    #GIE, SR
            nop

InfLoop:    jmp $ -> go to LPM


WDTISR:     xor.b    #0x06, &P2OUT
              reti
; Stack Pointer definition
              .global __STACK_END
            .sect    .stack
; Interrupt Vectors
              .sect  ".reset"                     ; MSP430 RESET Vector
            .short   RESET
              .sect ".int26"
            .short   WDTISR
```



1s Toggle LEDs (C) Using WDT Polling

```
#include  <msp430.h>

void main(void) {
    WDTCTL = WDT_ADLY_1000;              // 1 s interval timer
    P2DIR |= BIT2 + BIT1;                // Set P2.1 and P2.2 to output direction
    P2OUT = 0x00;                        // LEDs are off
    // use polling on WDTIFG (it's set every 1s)
    for (;;) {
        while ((IFG1 & WDTIFG) == 0);    wait for WDTIFG to become set
        P2OUT ^= (BIT1 | BIT2);
        IFG1 &= ~WDTIFG;                 // Clear WDTIFG in IFG1
    }
}
```



1s Toggle LEDs (ASM) Using WDT (ISR)

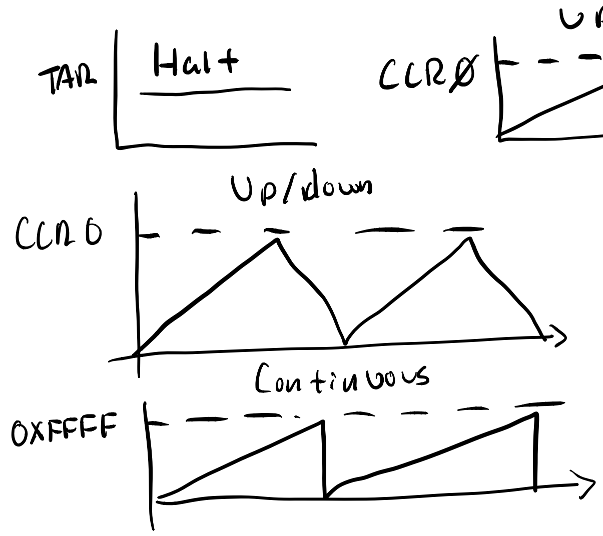
```
#include <msp430xG46x.h>

void main(void) {
    WDTCTL = WDT_ADLY_1000;              // 1s interval
    P2DIR |= BIT2 + BIT1;                // Set P2.2 and P2.1 to output direction
    P2OUT = 0x00;                        // LEDs are off
    IE1 |= WDTIE;                        // Enable WDT interrupt (WDTIE is set)

    _BIS_SR(LPM0_bits + GIE);            // Enter LPM0(CPU is off); Enable interrupts
}
```

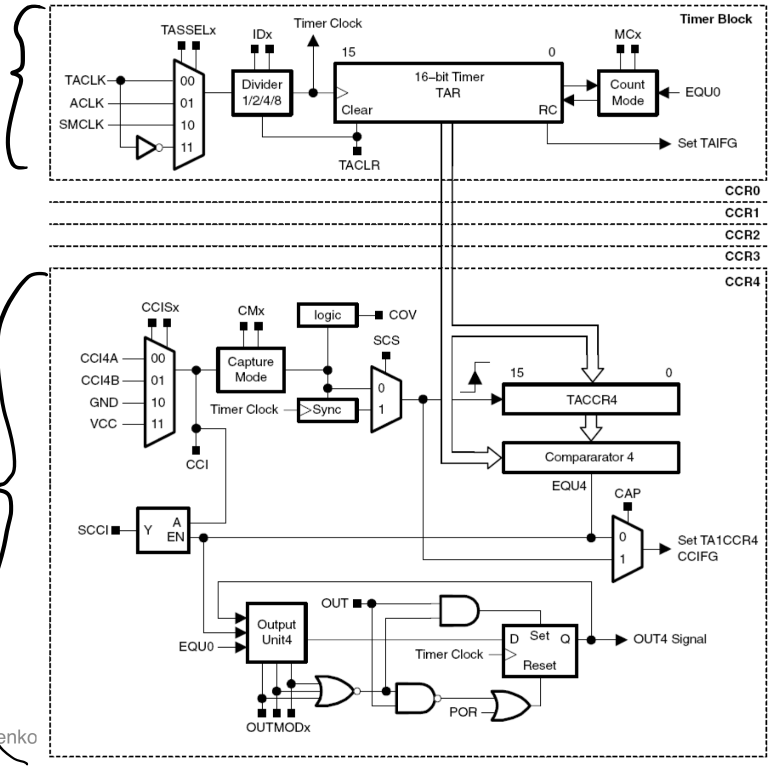
$\underbrace{\text{\_BIS\_SR(LPM0\_bits + GIE);}}$ ↳ low power mode 0, CPU no longer executing.

```
// Watchdog Timer interrupt service routine
#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void) {
    P2OUT ^= (BIT2 | BIT1);              // Toggle P2.1 and P2.2 using exclusive-OR
}
```

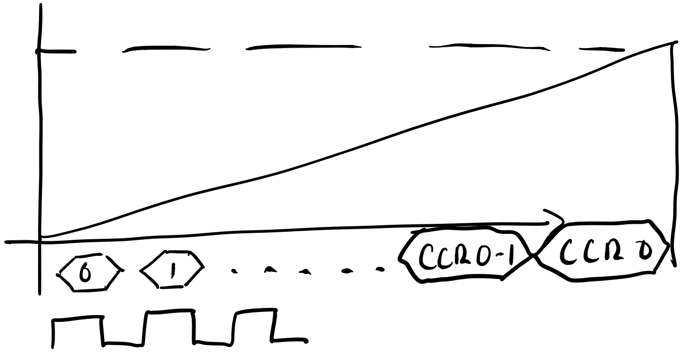
Timers

- Timer Block (counter)
- Capture/Compare Blocks





UP Mode



Timer clock 1μs

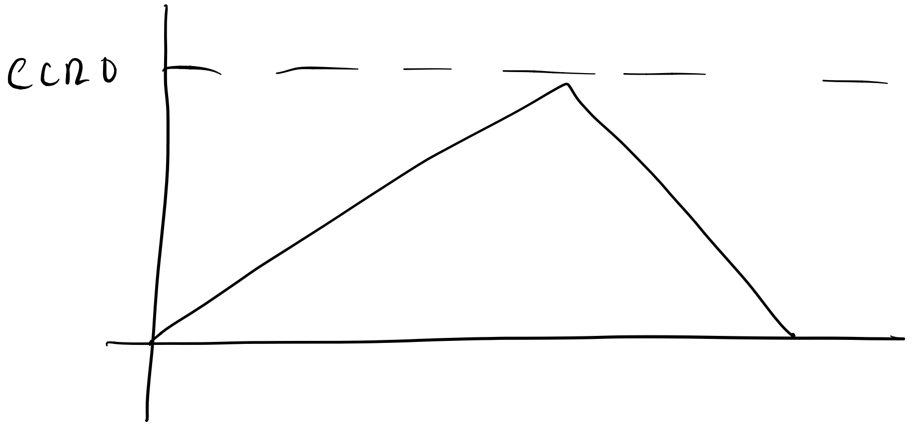
1μs

CCR0+1

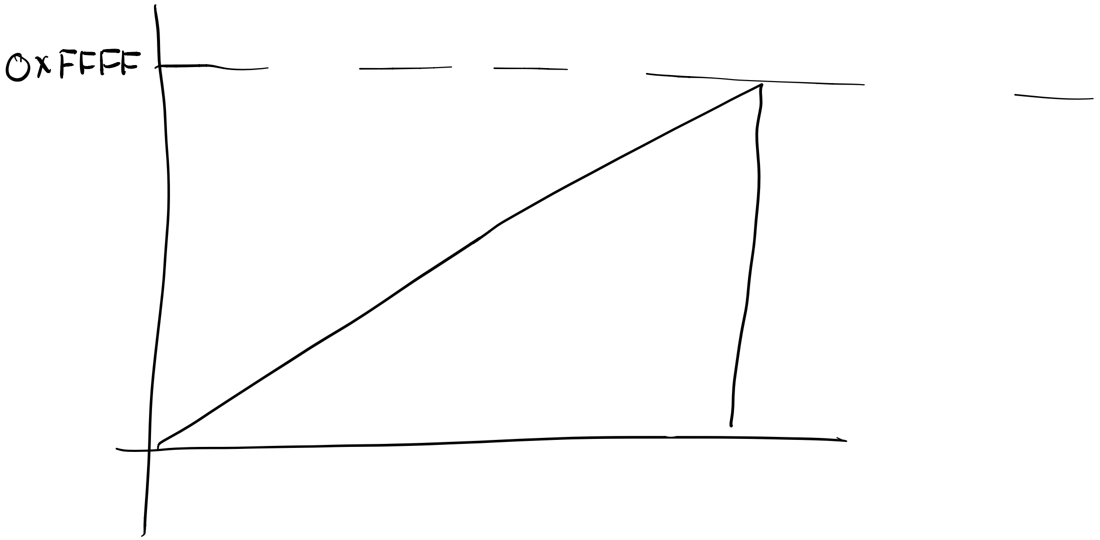
$$\frac{1\,\text{ms}}{1\,\mu\text{s}} = 1{,}000$$

$$CCR0 = 1000 - 1 = 999$$

UP/Down Mode

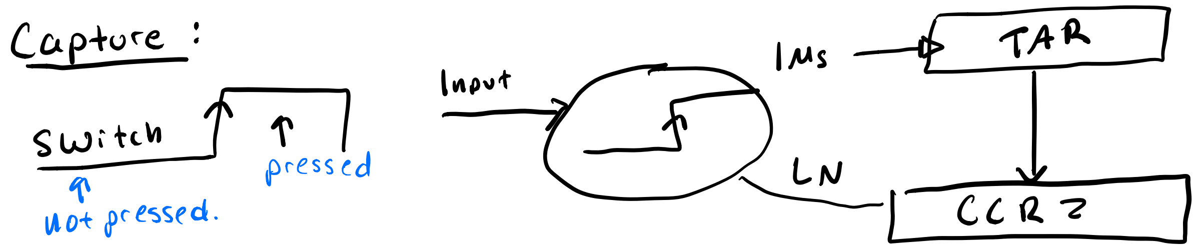


Continuous Mode

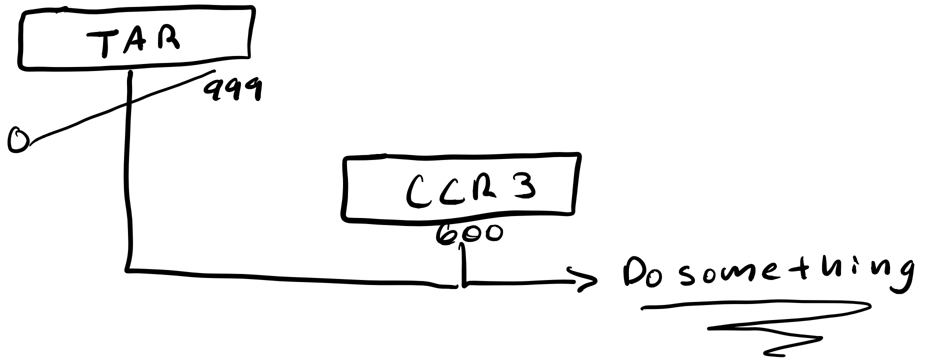


Capture & Compare Block

Capture :



Compare :


TAR
999
0
CCR3
600
Do something