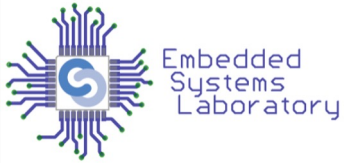


PROIECTAREA CU MICROPROCESOARE

Cursul 4
Timers, RTC, PWM

Facultatea de Automatică și Calculatoare
Politehnica București

"Time is real"

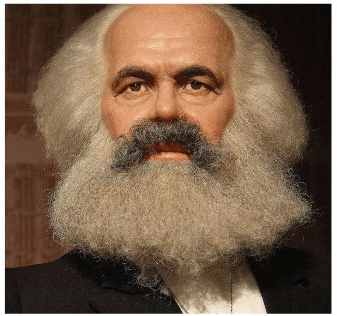
-Aristotle

"Time is an illusion of the mind"

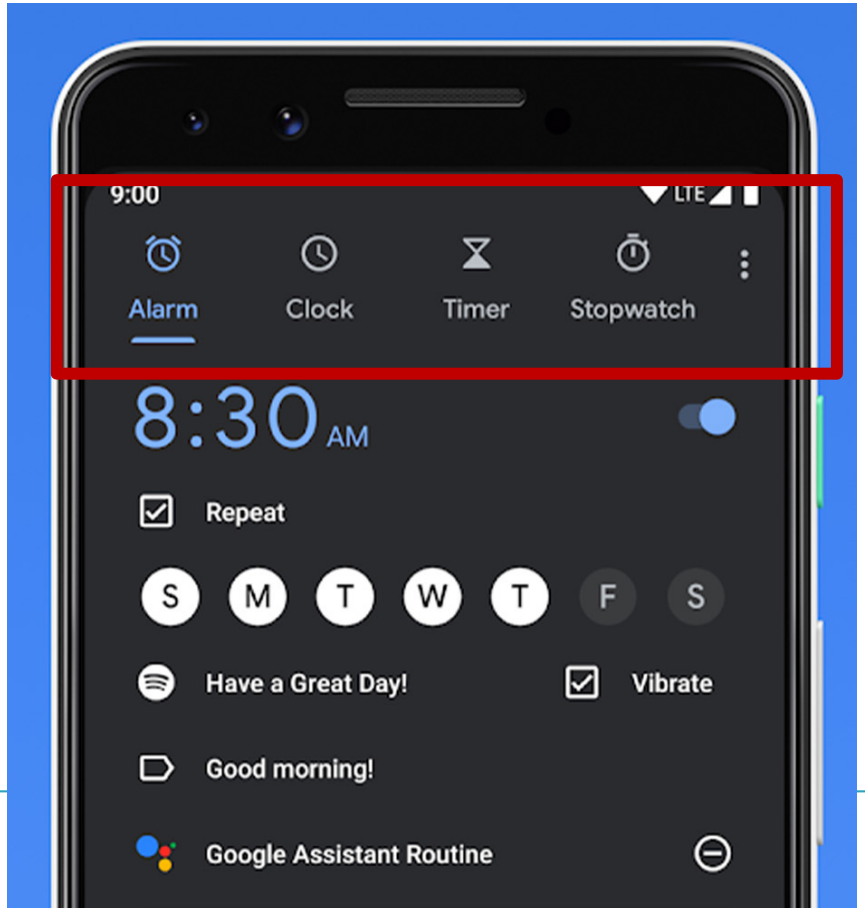
-Immanuel Kant

"Time was invented by clock companies to sell more clocks"

-Karl Marx



Android Clock App



- Alarm – Setează alarma la o anumită oră
- World Clock – afișează ceasul de timp real din mai multe zone
- Stopwatch – măsoară timpul trecut de la un eveniment
- Timer – numără timpul trecut și generează alarmă cand timpul ajunge la zero

Controlul motoarelor/luminilor

- Servomotoare – semnalul PWM controlează rotația și direcția
- Motoare DC – semnalul PWM controlează puterea
- RGB LEDs – semnalul PWM permite variația luminozității

Metode din android.os.SystemClock

Public Methods	
static long	currentThreadTimeMillis () Returns milliseconds running in the current thread.
static long	elapsedRealtime () Returns milliseconds since boot, including time spent in sleep.
static long	elapsedRealtimeNanos () Returns nanoseconds since boot, including time spent in sleep.
static boolean	setCurrentTimeMillis (long millis) Sets the current wall time, in milliseconds.
static void	sleep (long ms) Waits a given number of milliseconds (of uptimeMillis) before returning.
static long	uptimeMillis () Returns milliseconds since boot, not counting time spent in deep sleep.

Biblioteca C standard <time.h>

- avr-gcc are o bibliotecă similară

https://www.nongnu.org/avr-libc/user-manual/group__avr__time.html

Library Functions

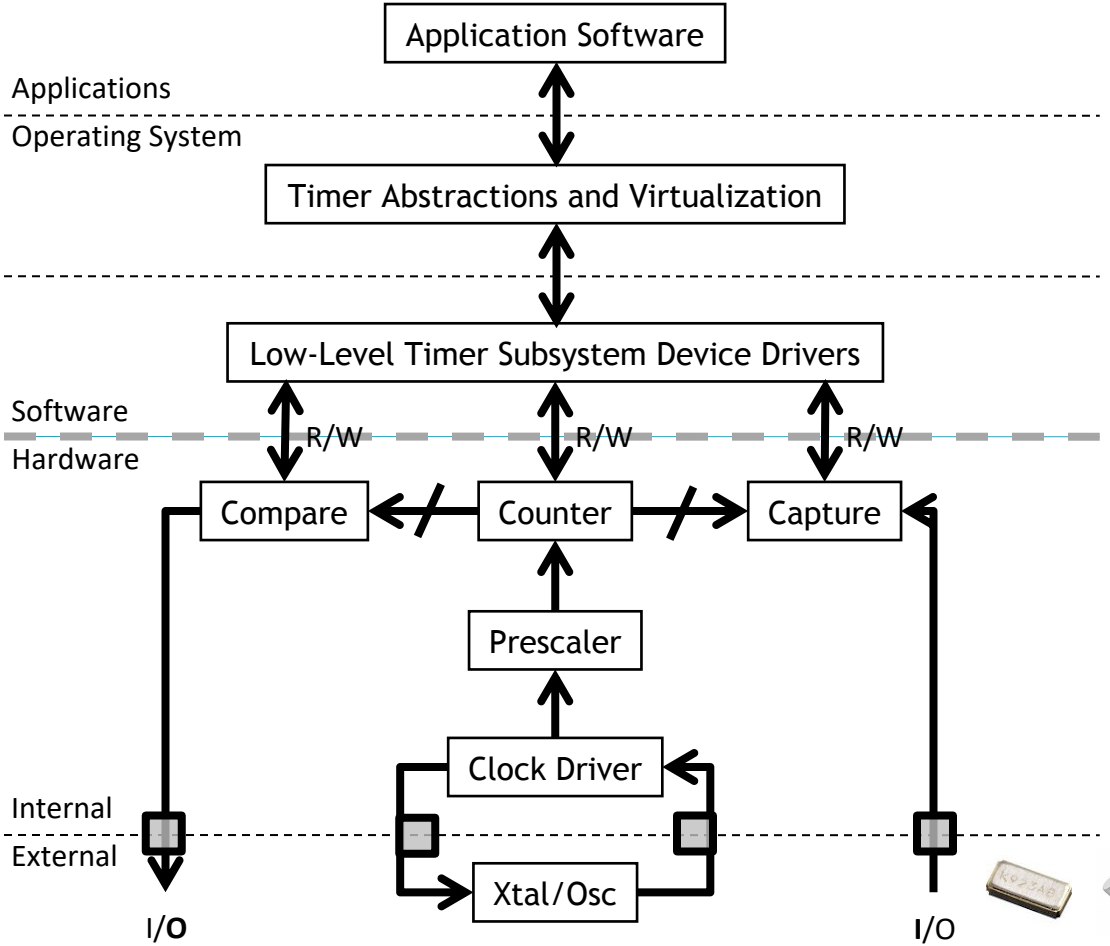
Following are the functions defined in the header time.h:

S.N.	Function & Description
1	char *asctime(const struct tm *timeptr) Returns a pointer to a string which represents the day and time of the structure timeptr.
2	clock_t clock(void) Returns the processor clock time used since the beginning of an implementation-defined era (normally the beginning of the program).
3	char *ctime(const time_t *timer) Returns a string representing the localtime based on the argument timer.
4	double difftime(time_t time1, time_t time2) Returns the difference of seconds between time1 and time2 (time1-time2).
5	struct tm *gmtime(const time_t *timer) The value of timer is broken up into the structure tm and expressed in Coordinated Universal Time (UTC) also known as Greenwich Mean Time (GMT).
6	struct tm *localtime(const time_t *timer) The value of timer is broken up into the structure tm and expressed in the local time zone.
7	time_t mktime(struct tm *timeptr) Converts the structure pointed to by timeptr into a time_t value according to the local time zone.
8	size_t strftime(char *str, size_t maxsize, const char *format, const struct tm *timeptr) Formats the time represented in the structure timeptr according to the formatting rules defined in format and stored into str.
9	time_t time(time_t *timer) Calculates the current calender time and encodes it into time_t format.

<time.h>: struct tm

```
struct tm {
   int tm_sec;         /* seconds,  range 0 to 59          */
   int tm_min;         /* minutes, range 0 to 59           */
   int tm_hour;        /* hours, range 0 to 23             */
   int tm_mday;        /* day of the month, range 1 to 31  */
   int tm_mon;         /* month, range 0 to 11             */
   int tm_year;        /* The number of years since 1900   */
   int tm_wday;        /* day of the week, range 0 to 6    */
   int tm_yday;        /* day in the year, range 0 to 365  */
   int tm_isdst;       /* daylight saving time             */
};
```

Anatomia unui sistem cu timer



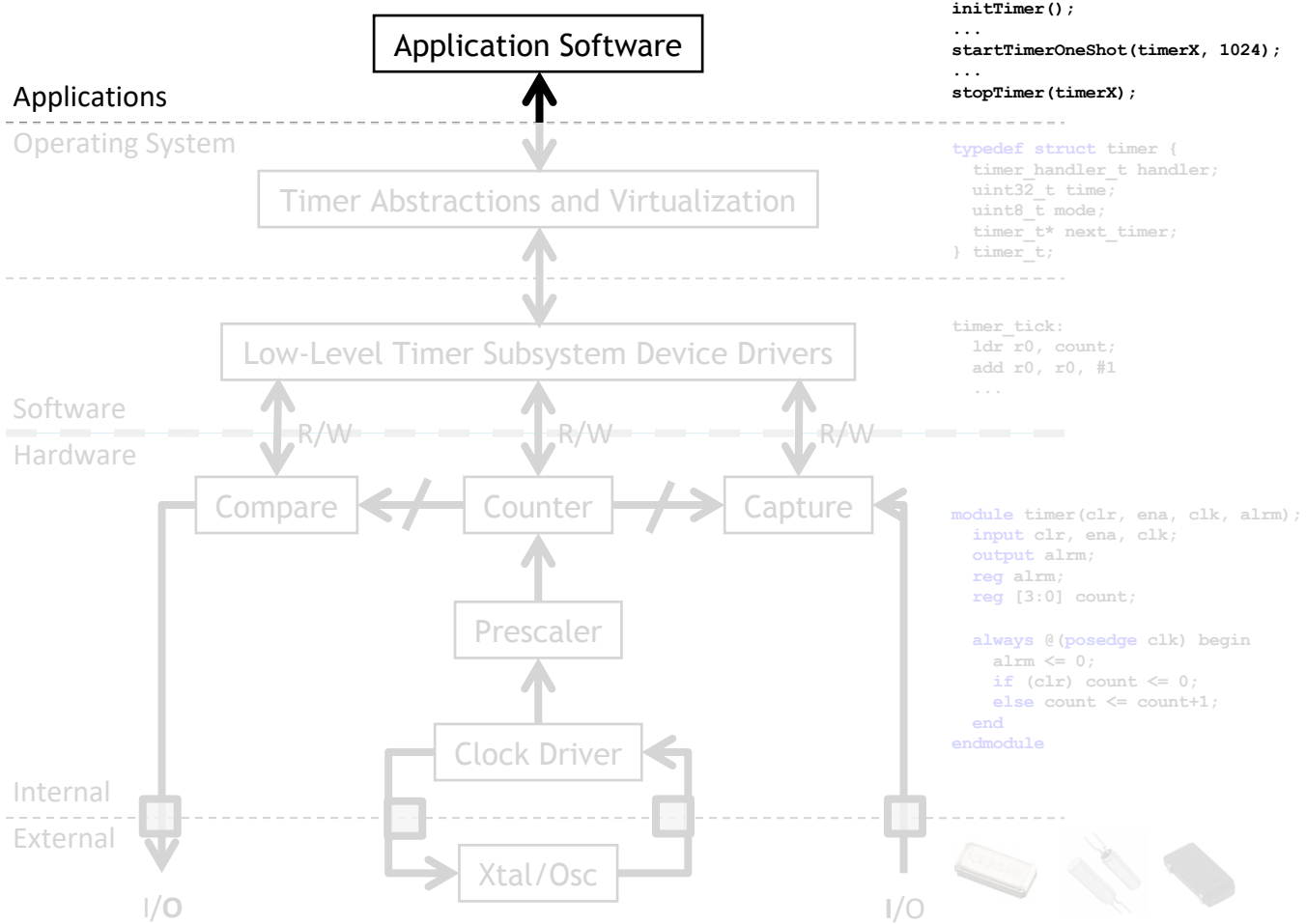

```
...
timer_t timerX;
initTimer();
...
startTimerOneShot(timerX, 1024);
...
stopTimer(timerX);
```

```
typedef struct timer {
  timer_handler_t handler;
  uint32_t time;
  uint8_t mode;
  timer_t* next_timer;
} timer_t;
```

```
timer_tick:
  ldr r0, count;
  add r0, r0, #1
  ...
```

```
module timer(clr, ena, clk, alrm);
  input clr, ena, clk;
  output alrm;
  reg alrm;
  reg [3:0] count;

  always @(posedge clk) begin
    alrm <= 0;
    if (clr) count <= 0;
    else count <= count+1;
  end
endmodule
```

Anatomy of a timer system



Ce dorim să obținem de la un subsistem de timing?

- Ora, minutul, seconda, data curentă
  - Date: Month, Day, Year
  - Time: HH:MM:SS:mmm
  - Dat de un “Real-Time Clock” sau RTC
- Alarme: execută cod după ce a trecut un timp
  - Poate fi măsurat ca o durată de timp din momentul cureng (de ex. Δt)
  - Sau poate fi o coordonată temporală din viitor (de ex. azi la 3pm)
- Stopwatch: măsoară cât a trecut de la declanșare
  - Declanșat de eveniment intern sau extern
- Timer – decrementează un contor și anunță când = 0
  - Poate să execute cod (de ex. un handler)
  - Poate să execute o acțiune (de ex. set/clear o linie de IO)

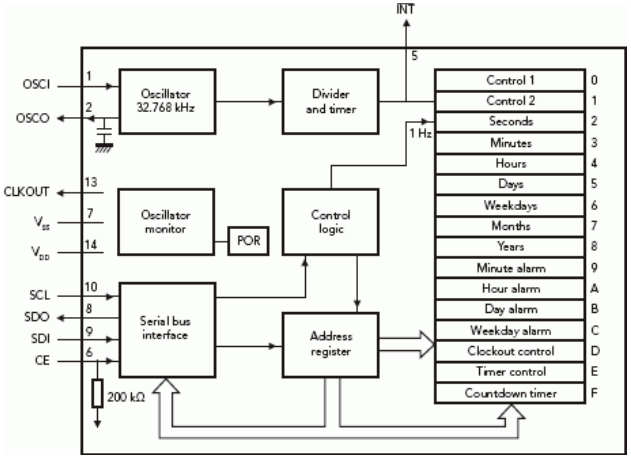
Ce dorim să obținem de la un subsistem de timing?

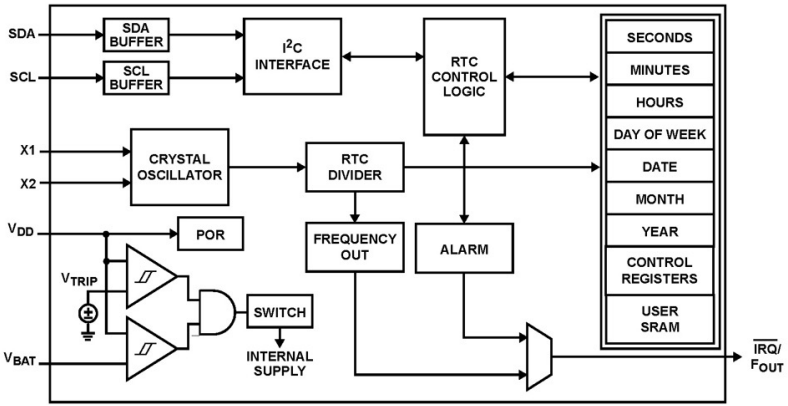
- Ceas de timp real
  - datetime_t getDateTime()
- Alarmă
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: măsoară cât durează un eveniment
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

LDR R1, [R0, #0] % R0=timer address

- Timer – decrementează un contol și anunță când = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

Real-Time Clock (RTC)





- De obicei un modul separat
- Registre dedicate pentru
  - Ani, Luni, Zile
  - Ore, Minute, Secunde
- Alarme: oră, minut, zi
- Accesate prin
  - Memory-mapped I/O
  - Serial bus (I2C, SPI)

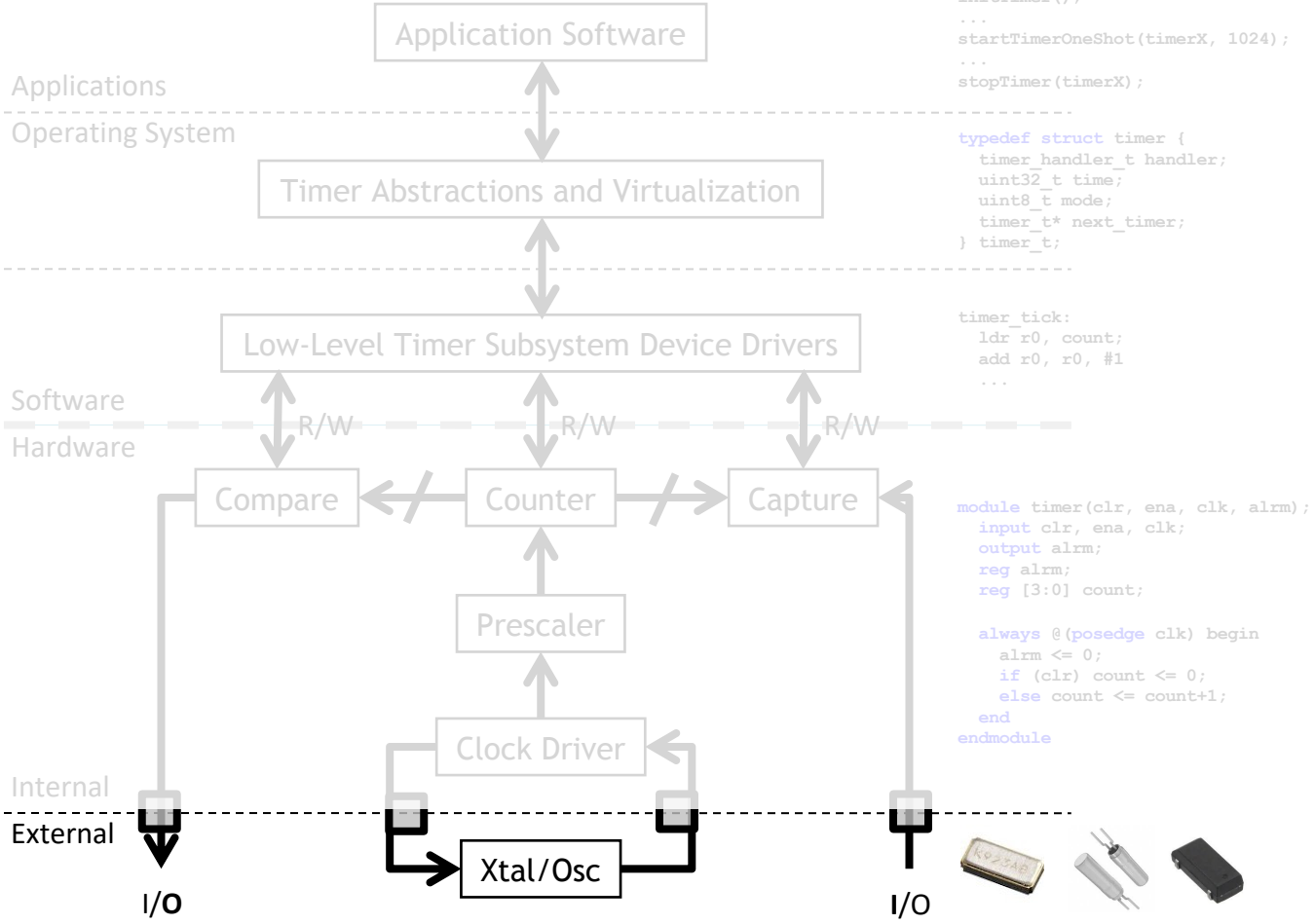
Ce dorim să obținem de la un subsistem de timing?

- Ceas de timp real
  - datetime_t getDateTime()
- Alarmă
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: măsoară cât durează un eveniment
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

```
LDR R1, [R0, #0]      % R0=timer address
```

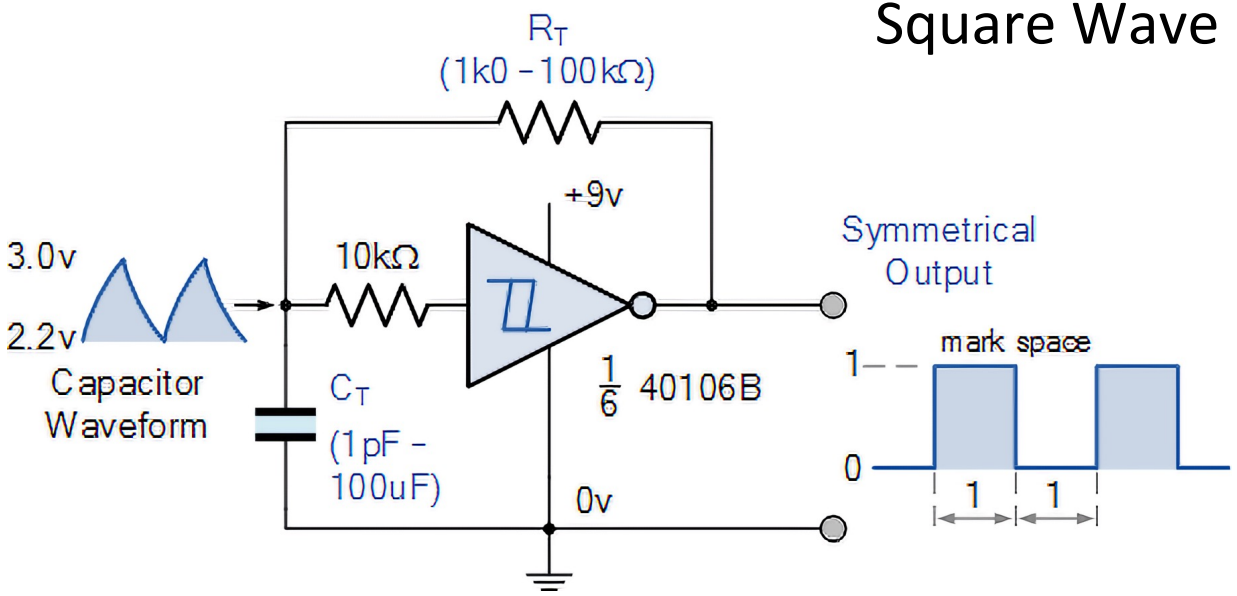
- Timer – decrementează un contol și anunță când = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

Anatomy of a timer system

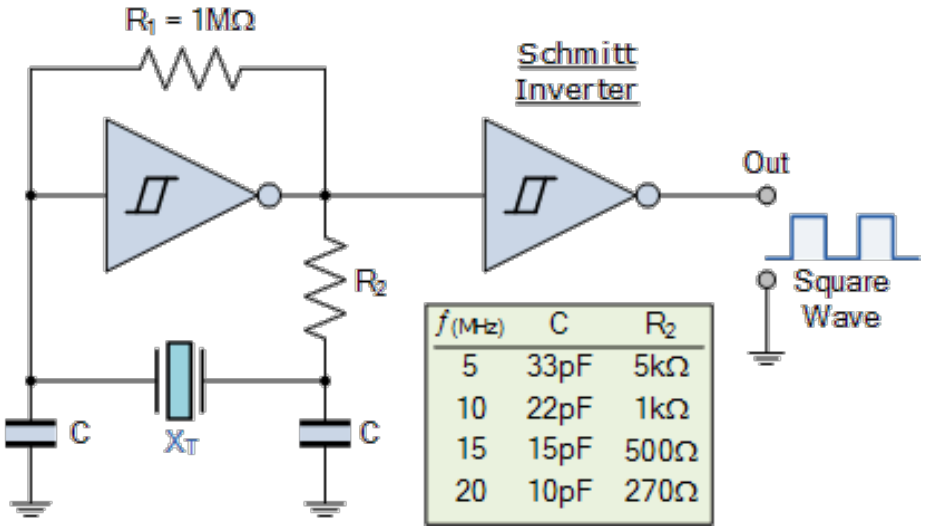


Oscilatoare – RC

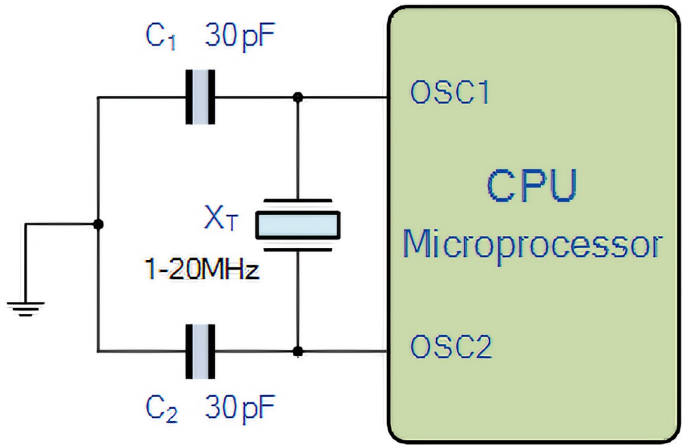
Square Wave Oscillator



Oscilatoare – Cristal Cuarț

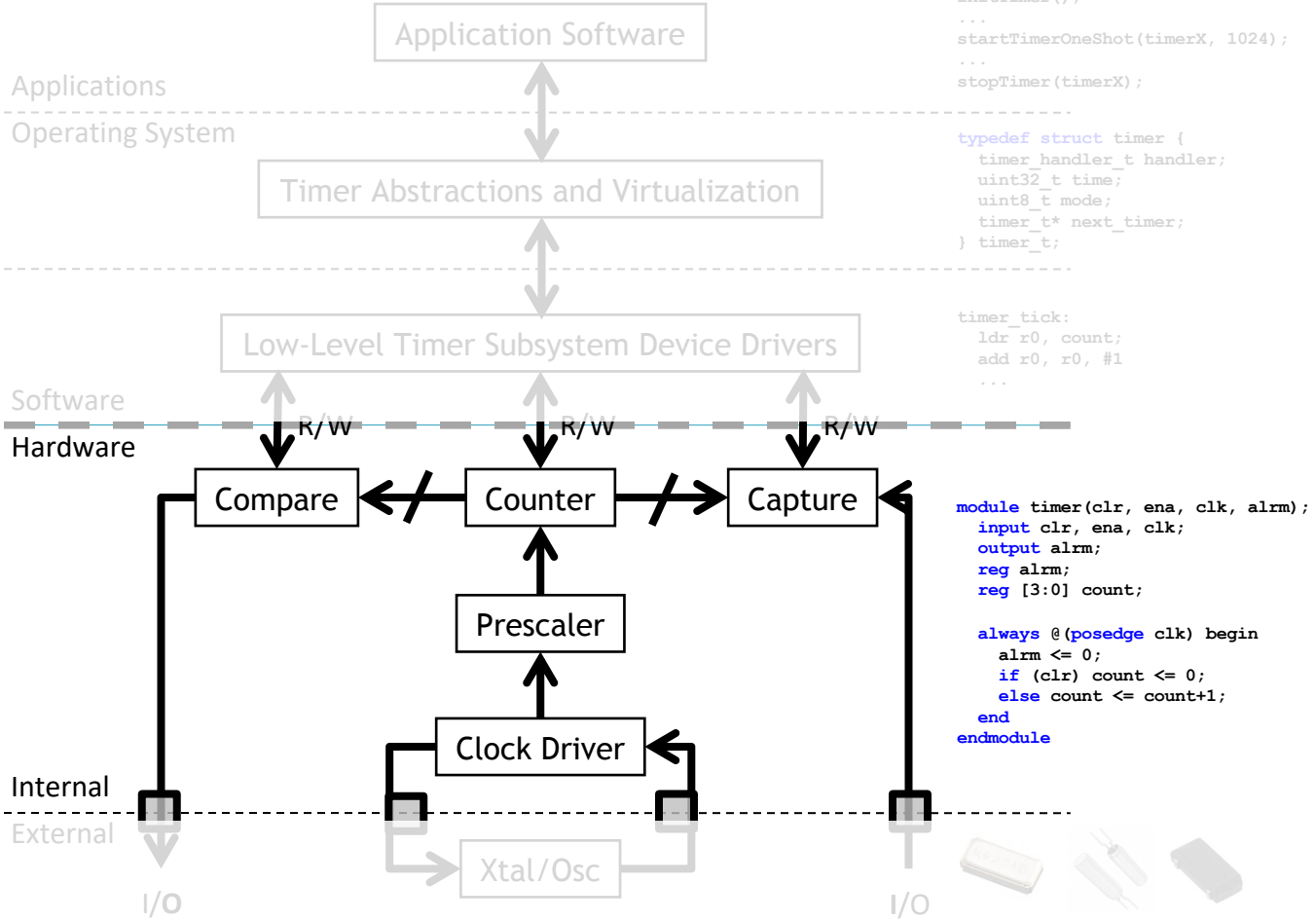


f (MHz)	C	R2
5	33pF	5kΩ
10	22pF	1kΩ
15	15pF	500Ω
20	10pF	270Ω



Pierce Oscillator

Anatomy of a timer system



Ce dorim să obținem de la un subsistem de timing?

- Ceas de timp real
  - datetime_t getDateTime()
- Alarmă
  - void alarm(callback, delta)
  - void alarm(callback, datetime_t)
- Stopwatch: măsoară cât durează un eveniment
  - t1 = now(); … ; t2 = now(); dt = difftime(t2, t1);
  - GPIO_INT_ISR:

    LDR R1, [R0, #0] % R0=timer address
- Timer – decrementează un contol și anunță când = 0
  - void timer(callback, delta)
  - Timer fires → Set/Clear GPIO line (using DMA)

De ce ne pasă?

- Sunt două moduri de bază în care poate fi folosit un timer :
  - Măsoară cât durează un eveniment
    - “Capture”
  - Fă ceva să se întâmple la fiecare X secunde
    - “Compare”

Exemplul 1: Capture

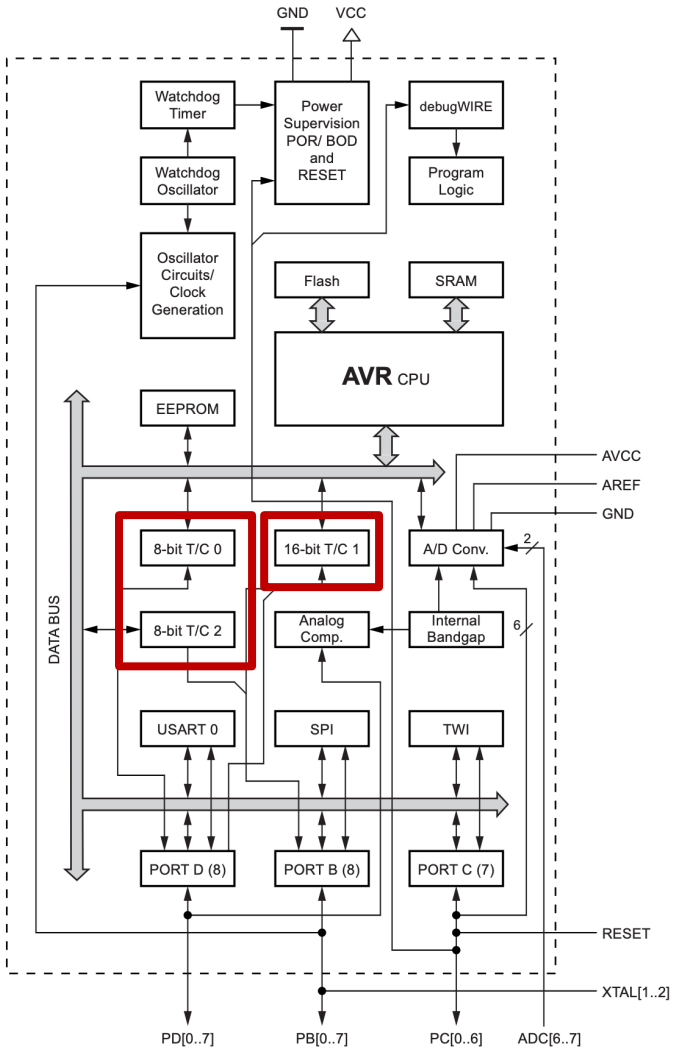
- Ventilator
  - Să zicem că vrem să aflăm cât de repede se învârte un ventilator. O metodă ar fi să generăm o întrerupere de ficare dată când acesta face o rotație
    - Idee bună, dar durează să procesezi întreruperea. Un sistem puternic încărcat e posibil să vadă ventilatorul mai lent decât este
    - Probabil că nu e bine
  - Soluție? Un timper care reține *imediat* cât de mult a durat o rotație și apoi generează o întrerupere. De asemenea, restartează timer-ul imediat.
- Aceeași problemă poate apărea la o mașină care folosește rotația roților pentru a măsura viteza (pentru vitezometru sau pentru sistemul ABS)

Exemplul 2: Compare

- Comanda unui motor cu PWM.
  - Motorul se învârte cu o viteză d.p. cu tensiunea a semnalului
    - Un circuit analogic de comandă este complicat
      - Trebuie să convertim semnalul digital de la procesor în unul analogic
      - Trebuie să amplificăm semnalul (op-amp?)
    - Mai bine alternăm între “Max” și “Off” foarte rapid
      - Valoarea medie a semnalului contează.
      - Acum nu mai avem nevoie de amplificare liniară — doar “on” și “off” (asta face foarte bine un tranzistor)
  - Avem nevoie de un semnal de o anumită frecvență și cu un anumit factor de umplere
    - % de timp cât semnalul este în logic 1

ATmega324P Timer Resources

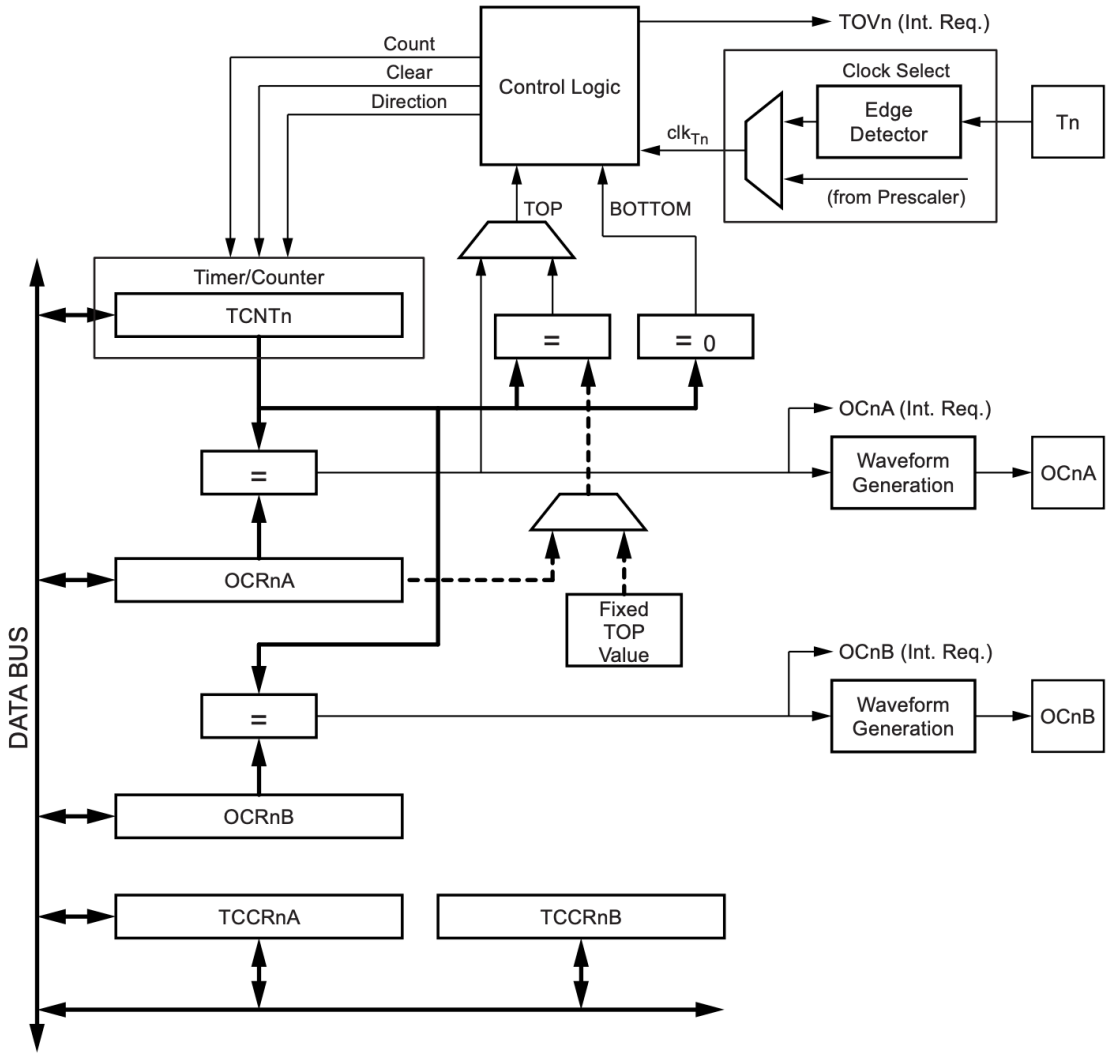
- 8-bit Timer/Counter 0
- 16-bit Timer/Counter 1
- 8-bit Timer/Counter 2



8-bit Timers

Moduri de funcționare

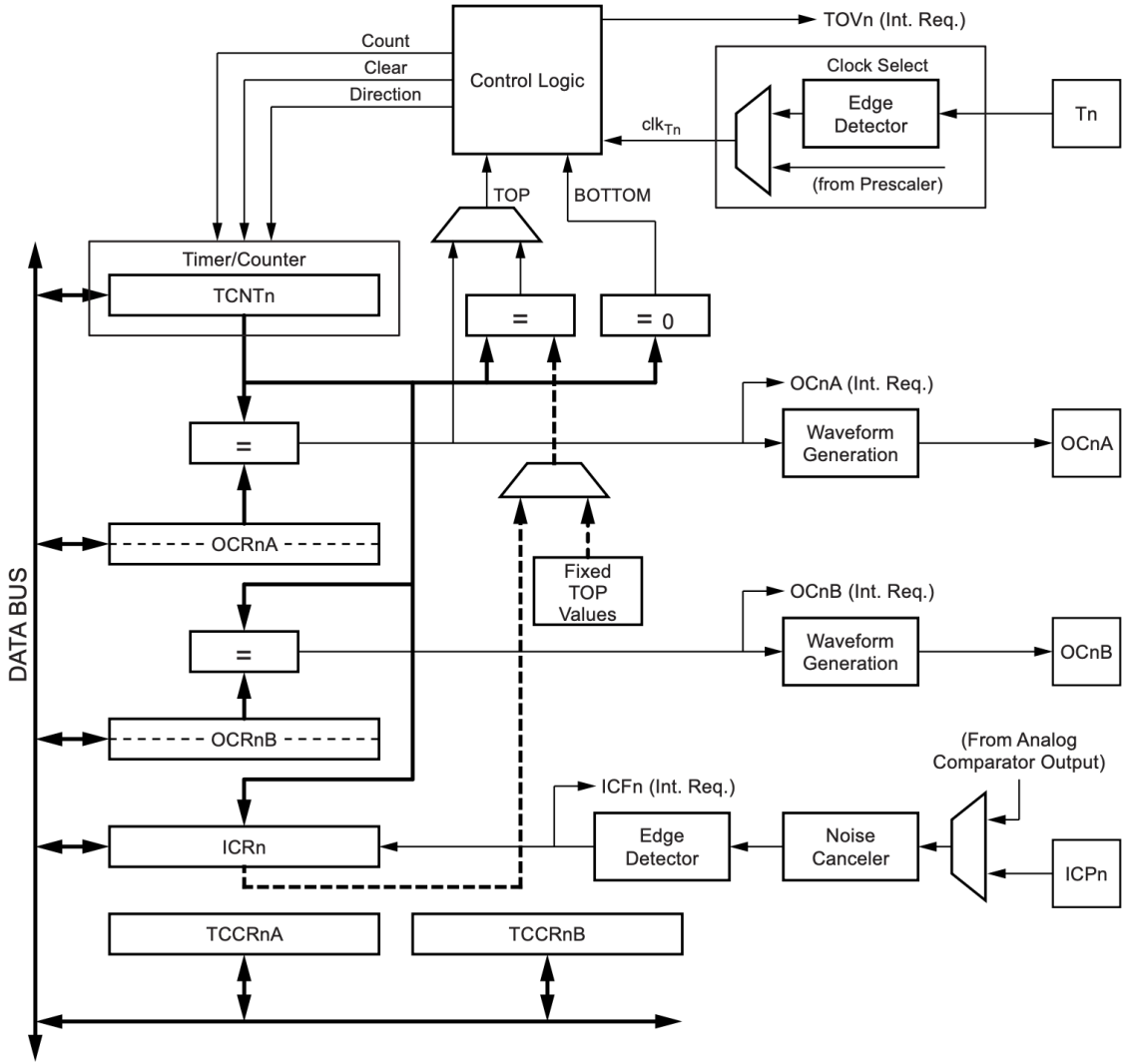
- Overflow (reset at 0xFF)
- Capture (CTC)
- Fast PWM
- Phase-correct PWM



16-bit Timer

Moduri de funcționare

- Overflow (reset at 0xFFFF)
- Capture
  - CTC
  - ICR
- Fast PWM
  - 5 modes
  - 8 to 16-bit resolution
- Phase-correct PWM
  - 7 modes
  - 8 to 16-bit resolution



Inițializare Timer – 1 întrerupere pe secundă

- Vrem să implementăm un ceas de timp real în cod pentru ATmega324P
- Trebuie să inițializăm un timer să genereze o întrerupere pe secundă
- Minutele, orele etc. le vom incrementa în cod
  - Nici un timer nu are rezoluția necesară pentru a temporiza intervale lungi de timp
- Dacă setăm prescaler-ul la 1024 și folosim un timer de 8 biți, atunci putem să temporizăm maxim 12MHz/1024/256 = 45.77Hz (aprox 0.02s)
- Ca să numărăm o secundă ne trebuie să numărăm pe mai mult de 8 biți.
- Dar dacă folosim Timer1 (16 biți)?
  - Dacă folosim un prescaler de 256 trebuie să numărăm până la 12MHz/256 = 46875 ca să ajungem la 1Hz

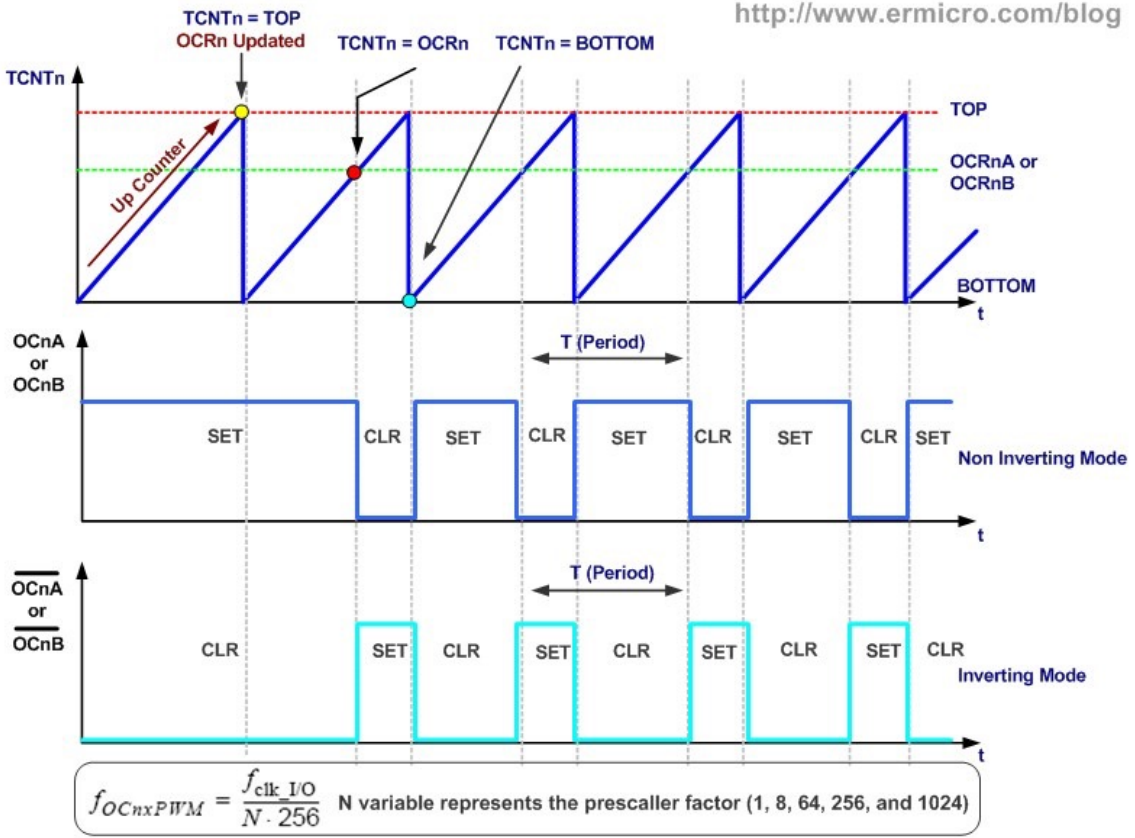
Code Example

- Întrerupere setată la Output Compare pe canalul A
- Setăm prescalerul la 256
- Registrul de Output Compare A (**OCR1A**) este setat la valoarea până la care trebuie să numărăm

```
#define F_CPU 12000000UL
#include <avr/io.h>
#include <avr/interrupt.h>

volatile uint32_t seconds = 0;

ISR(TIMER1_COMPA_vect)
{
    seconds++;
}

int main()
{

    OCR1A = (F_CPU / 256) - 1; /* 46875 */

    /* set prescaler at 256 */
    /* set operation mode at CTC with top at OCR1A */
    TCCR1B = (1 << CS12) | (1 << WGM12);

    /* set interrupt at output compare on channel A */
    TIMSK = (1 << OCIE1A);

    sei();
    while(1);
    return 0;
}
```

PWM

- Oricare dintre cele trei timere ale ATmega324P poate fi folosit pentru a genera semnale PWM
- Avantajul este că putem modifica din cod factorul de umplere al unui semnal digital
  - Factorul de umplere este d.p. cu valoarea medie a semnalului
  - Un timer care poate genera semnal PWM este, în esență, un convertor D/A pe un singur bit



Code

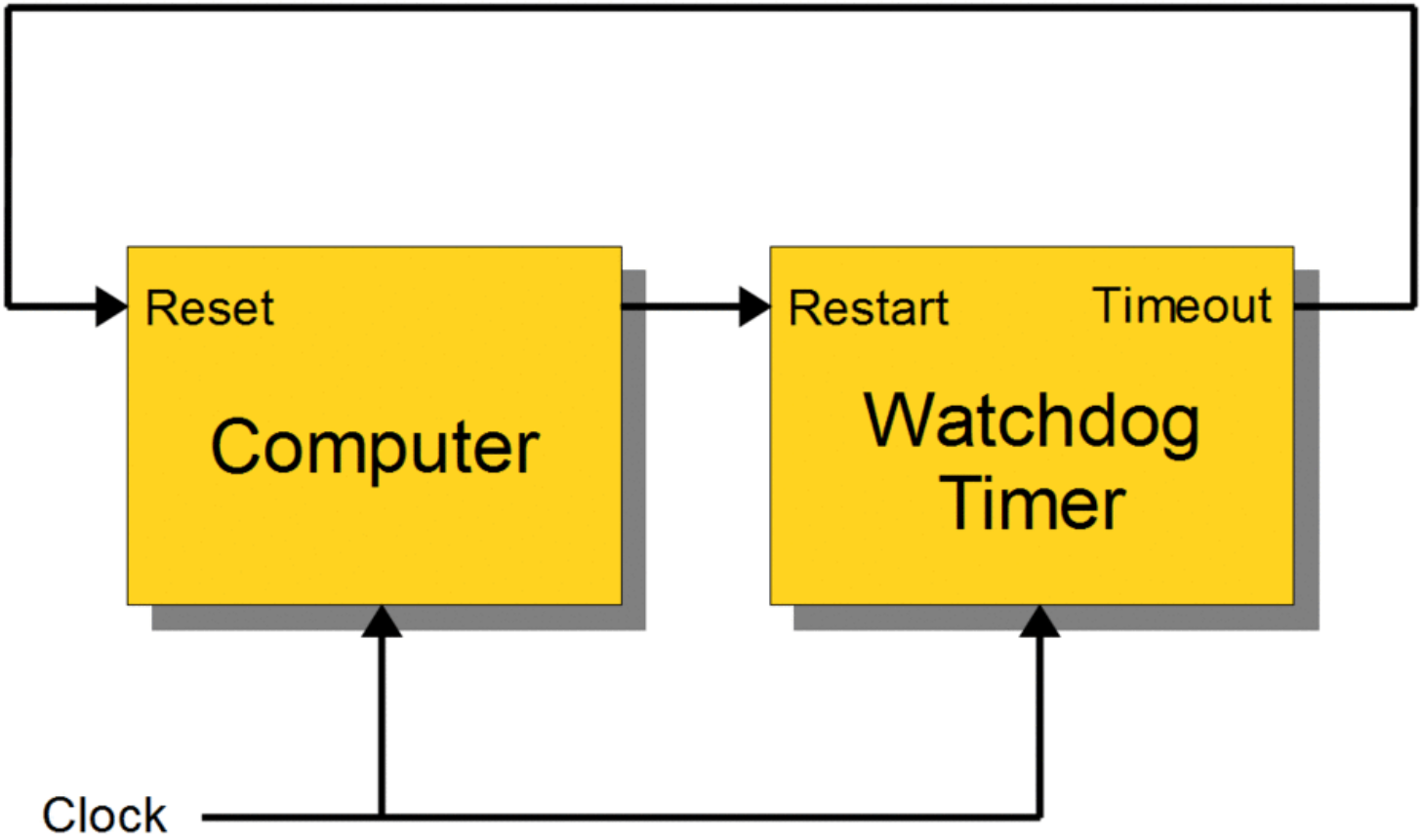
- Timer2 setat în modul FastPWM, non-inverting
- Output semnal pe pinul OC2A (PD7)
- Factorul de umplere este reglat prin scrierea unei valori de la 0 (0%) la 255 (100%) în registrul OCR2A

```
#define F_CPU 12000000UL
#include "avr/io.h"
#include <util/delay.h>

void PWM_set() // PWM setup function
{
    DDRD |= (1<<PD7); //set PD7 (OC2A) as PWM output

    //select Fast PWM mode, non-inverting
    TCCR2A |= (1 << WGM20) | (1<<WGM21) | (1 << COM2A1);

    //set prescaler at F_CPU/1024
    TCCR2B |= (1 << CS20) |(1 << CS21) | (1 << CS22);
}

int main()
{
    unsigned char duty;

    PWM_set();          //call PWM setup function

    while (1)
    {
        for(duty = 0; duty < 255; duty++)   // 0 to max duty cycle
        {
            OCR2A = duty;      //slowly increase the LED brightness
            _delay_ms(10);
        }

        for(duty = 0; duty > 255; duty--)   // max to 0 duty cycle
        {
            OCR2A = duty;      //slowly decrease the LED brightness
            _delay_ms(10);
        }
    }
}
```


WHAT TYPE OF DOG
IS BEST AT TIMEKEEPING?
A WATCHDOG!
imgflip.com

Watchdog Timer

- Este tot un timer dar cu rol special pentru sistem
- Dacă este inițializat, el va număra până va ajunge la o valoare anume
  - În acel moment va reseta microcontrolerul
- Pentru a preveni asta, codul trebuie să reseteze periodic timerul
  - Petting the watchdog
- În acest fel se garantează că programul nu intră într-o stare necunoscută sau se blochează


Reset
Computer
Restart
Timeout
Watchdog
Timer
Clock

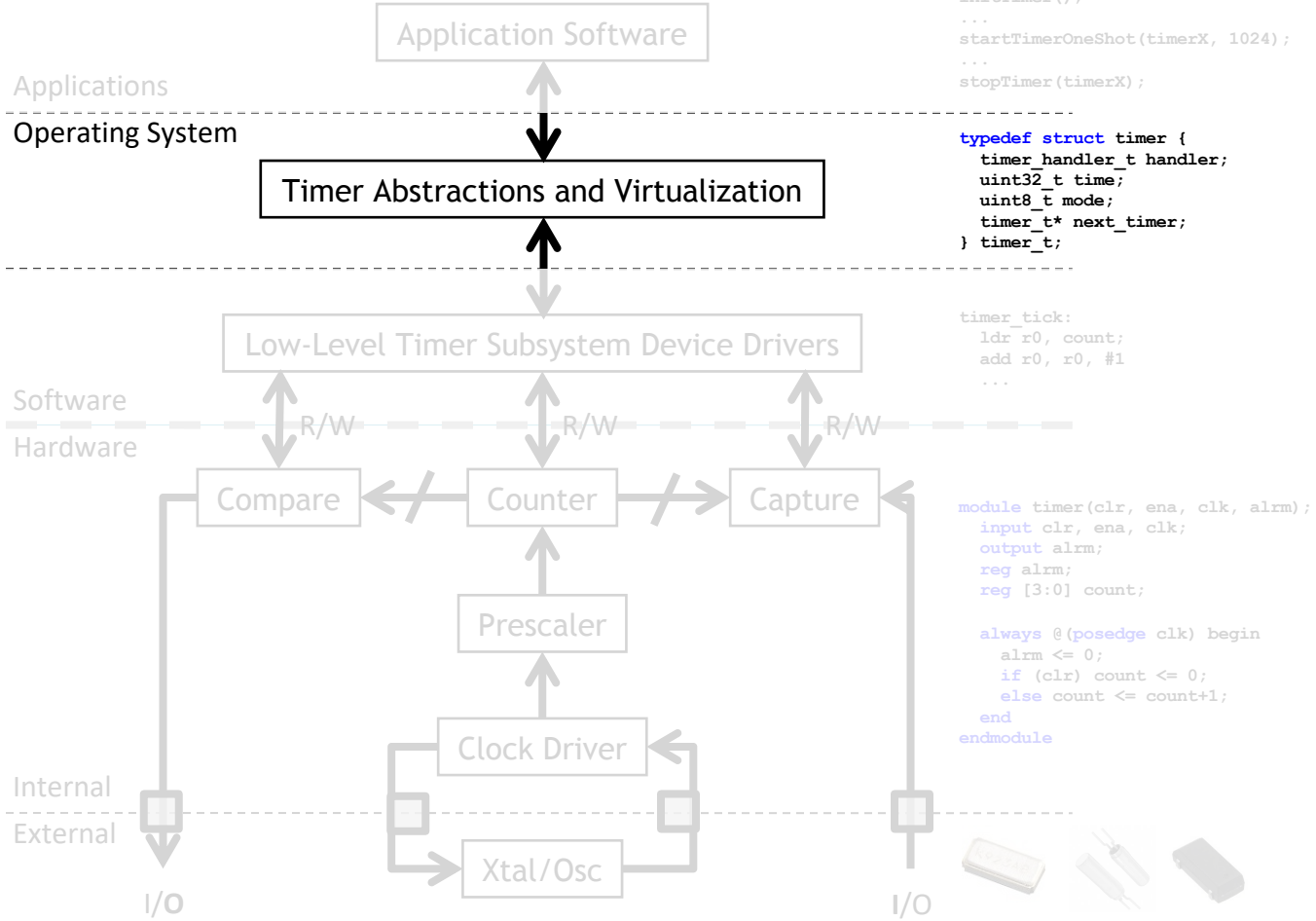
Code Example

- Watchdog inițializat să genereze întrerupere apoi să producă un Reset dacă nu este resetat în 4 secunde
- Întreruperea este utilă dacă vrem să executăm cod înainte de Reset
  - De exemplu să salvăm anumiți parametri din program
- Folosim *wdt_reset()* din *avr/wdt.h* pentru a reseta timerul

https://www.nongnu.org/avr-libc/user-manual/group__avr__watchdog.html

```
#define F_CPU 12000000UL
#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/interrupt.h>

volatile uint32_t seconds = 0;

ISR(WDT_vect)
{
    /* save program variables and states to EEPROM */
    save_stuff();
    /* this is where the processor resets */

}

int main()
{

    /* configure Watchdog to Interrupt and System Reset Mode */
    WDTCSR = (1 << WDTON) | (1 << WDE) | (1 << WDIE);

    /* program a 4s delay */
    WDTCSR |= (1 << WDP3);

    sei();
    while(1)
    {
        wdt_reset();

        do_stuff();

    }
    return 0;
}
```

Anatomy of a timer system



Virtual Timers

- Nu sunt niciodată suficiente timere hardware.
  - Niciodată.
- Ce putem face?
  - Ce-ar fi să încercăm o rezolvare în software?

Virtual Timers

- Idee simplă.
  - Să zicem că avem 10 evenimente pe care dorim să le declanșăm
    - Faci o listă cu ele și setezi timer-ul pentru primul element din listă.
      - După ce expiră și evenimentul este declanșat, setează timer-ul să genereze o întrerupere pentru următorul element.

Probleme?

- Funcționează doar pentru modul “compare” al timer-ului
- Va genera timpi mai lenți de răspuns pentru ISR
  - Poate nu e atât de relevant, poate planifici să ții cont de overhead

Probleme de implementare

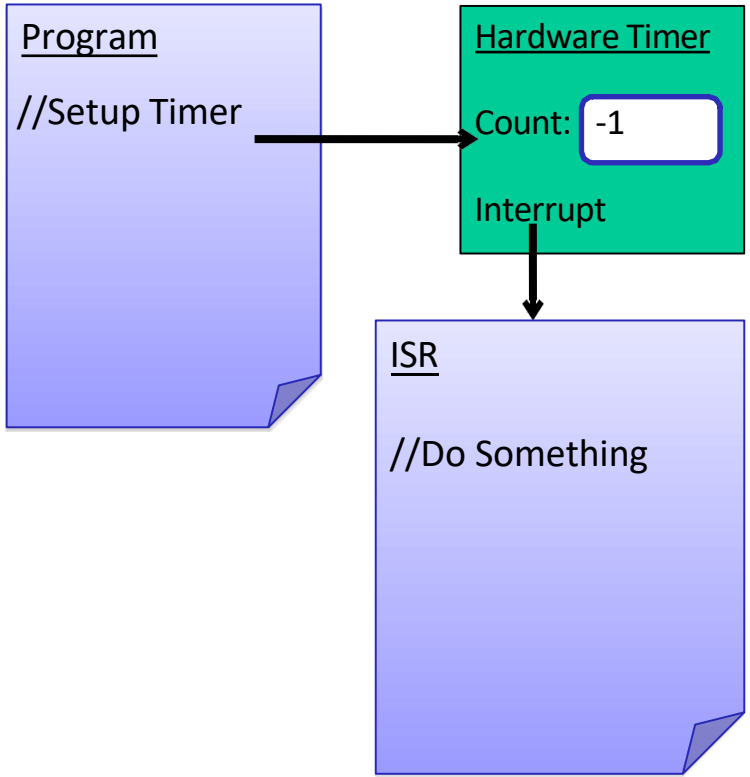
- Structura de date este partajată între ISR și main loop
  - Inserarea de evenimente se petrece cel puțin uneori din main.
  - Ștergerea de elemente se petrece în ISR.
    - Avem nevoie de o secțiune critică (disable interrupt)
- Numărătorul nostru nu e infinit,
  - Se dă peste cap la un moment dat (overflow)
  - De ce e asta o problemă?
- Ce funcționalitate ar fi ok de implementat?
  - În general, one-shot vs. evenimente repetitive
  - Dar poate vrem și altă funcționalitate
- Ce se întâmplă dacă două evenimente trebuie să se petreacă simultan?
  - Alege o ordine, execută pe amândouă…

Structuri de date

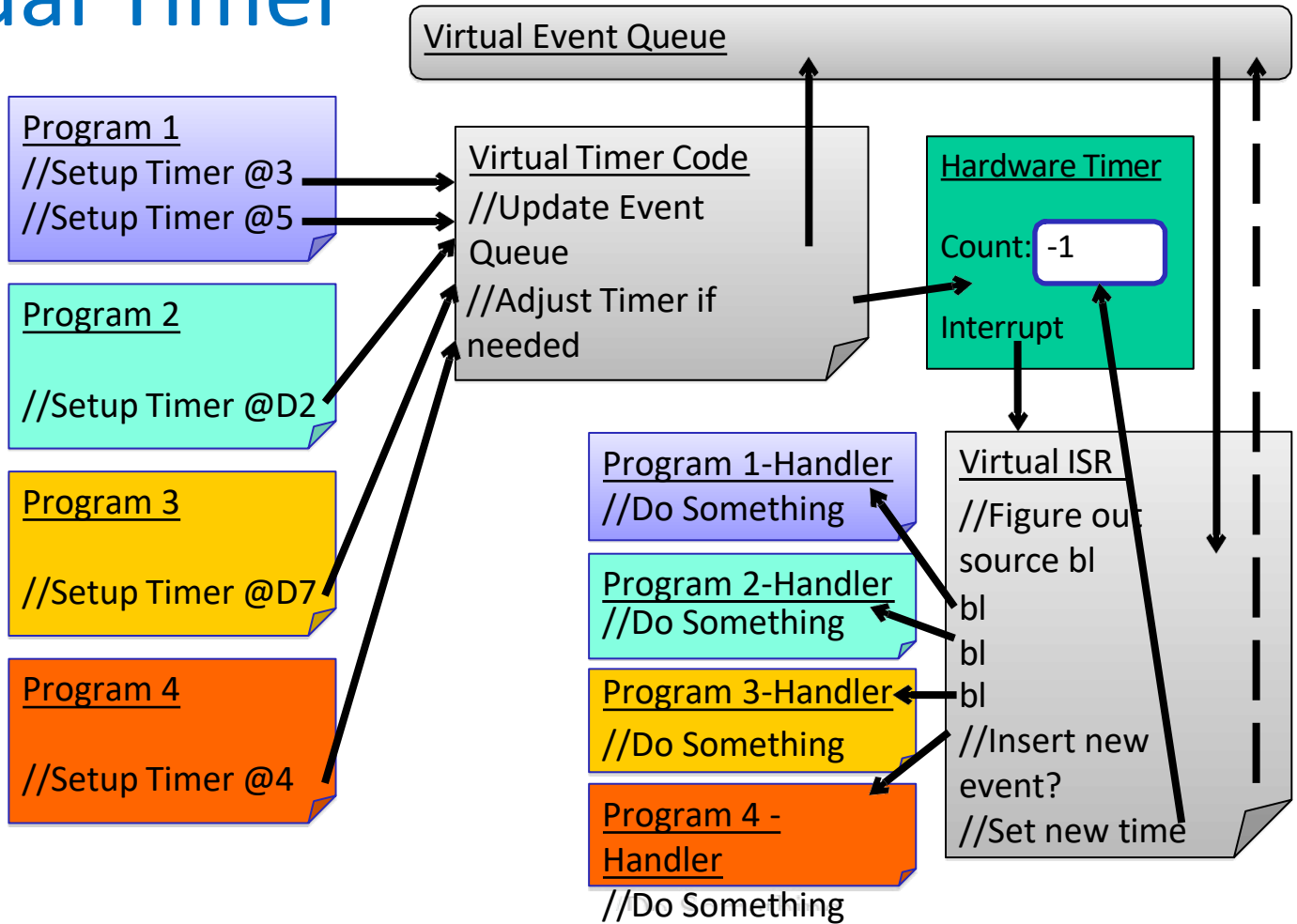
- Cum arată structura de date?
  - Datele trebuie sortate
    - Inserăm câte un element odată
  - Scoatem întotdeauna un element de la final
  - Dar adăugăm în ordine sortată

```
typedef struct timer
{
    timer_handler_t handler;
    uint32_t        time;
    uint8_t         mode;
    timer_t*        next_timer;
} timer_t;

timer_t* current_timer;

void initTimer() {
    setupHardwareTimer();
    initLinkedList();
    current_timer = NULL;
}

error_t startTimerOneShot(timer_handler_t handler, uint32_t t) {
    // add handler to linked list and sort it by time
    // if this is first element, start hardware timer
}

error_t startTimerContinuous(timer_handler_t handler, uint32_t dt) {
    // add handler to linked list for (now+dt), set mode to continuous
    // if this is first element, start hardware timer
}

error_t stopTimer(timer_handler_t handler) {
    // find element for handler and remove it from list
}
```

HW Timer


Program
//Setup Timer
Hardware Timer
Count:
-1
Interrupt
ISR
//Do Something

Virtual Timer



Acknowledgements

These slides contain materials from Prabal Dutta, Mark Brehob and Thomas Schmid (UMich)