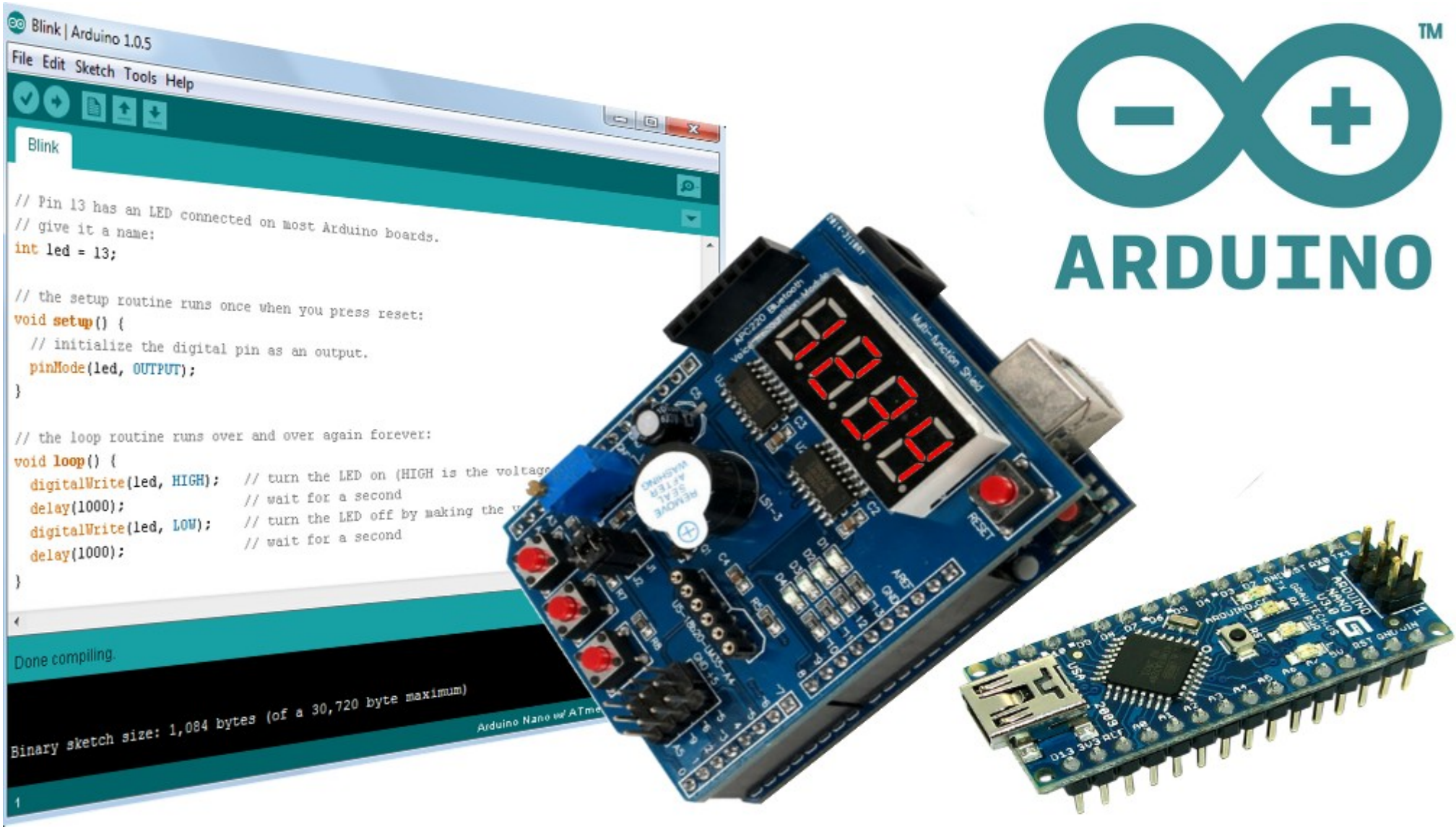
Arduino tanfolyam kezdőknek és haladóknak



6. Időzítők – számlálók (timers)

Felhasznált irodalom

- Microchip (ATMEL): **ATmega328p adatlap**
- Meettechniek: **Arduino Analog measurements**
- Roland Pelayo: **Interrupt Tutorial II - Arduino Timer Interrupt**
- Arduino Playground: **MsTimer2 library (old version)**

Arduino időzítők/számlálók

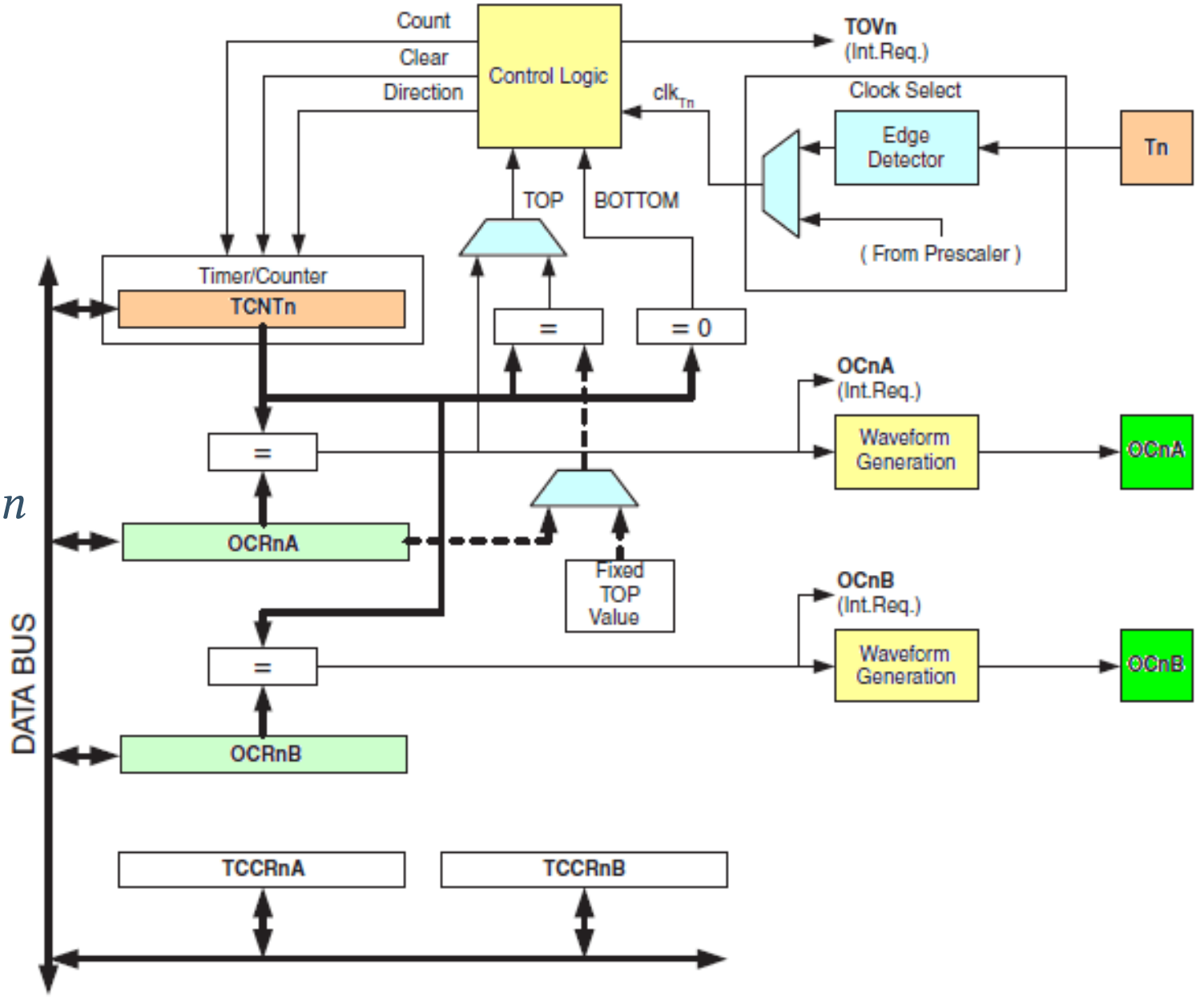
- Az **ATmega328** mikrovezérlő három időzítővel rendelkezik: **Timer0, Timer1** és **Timer2**
- Egyik számláló sem szabad felhasználású, ha átírjuk a regisztereket, akkor az adott időzítőhöz kapcsolódó gyári függvényeket nem használhatjuk (például **Timer2** beállításainak módosítása után nem használhatjuk a **tone()** függvényt)
- Az időzítők órajele lehet ***belső*** (a rendszer órajel /1, /8, /32, /64, /128, /256, /1024 leosztása választható; /32 és /128 csak **Timer2**), vagy ***külső jel*** (csak **Timer0** és **Timer1** esetén)

Időzítő	Bitek	Csatorna	Kivezetés	Frekvencia	Funkció
Timer0	8-bit	OC0A, OC0B	6, 5	980 Hz	millis()
Timer1	16-bit	OC1A, OC1B	9, 10	490 Hz	Servo library
Timer2	8-bit	OC2A, OC2B	11, 3	490 Hz	tone()

Timer0 blokkvázlata

Jellemzők:

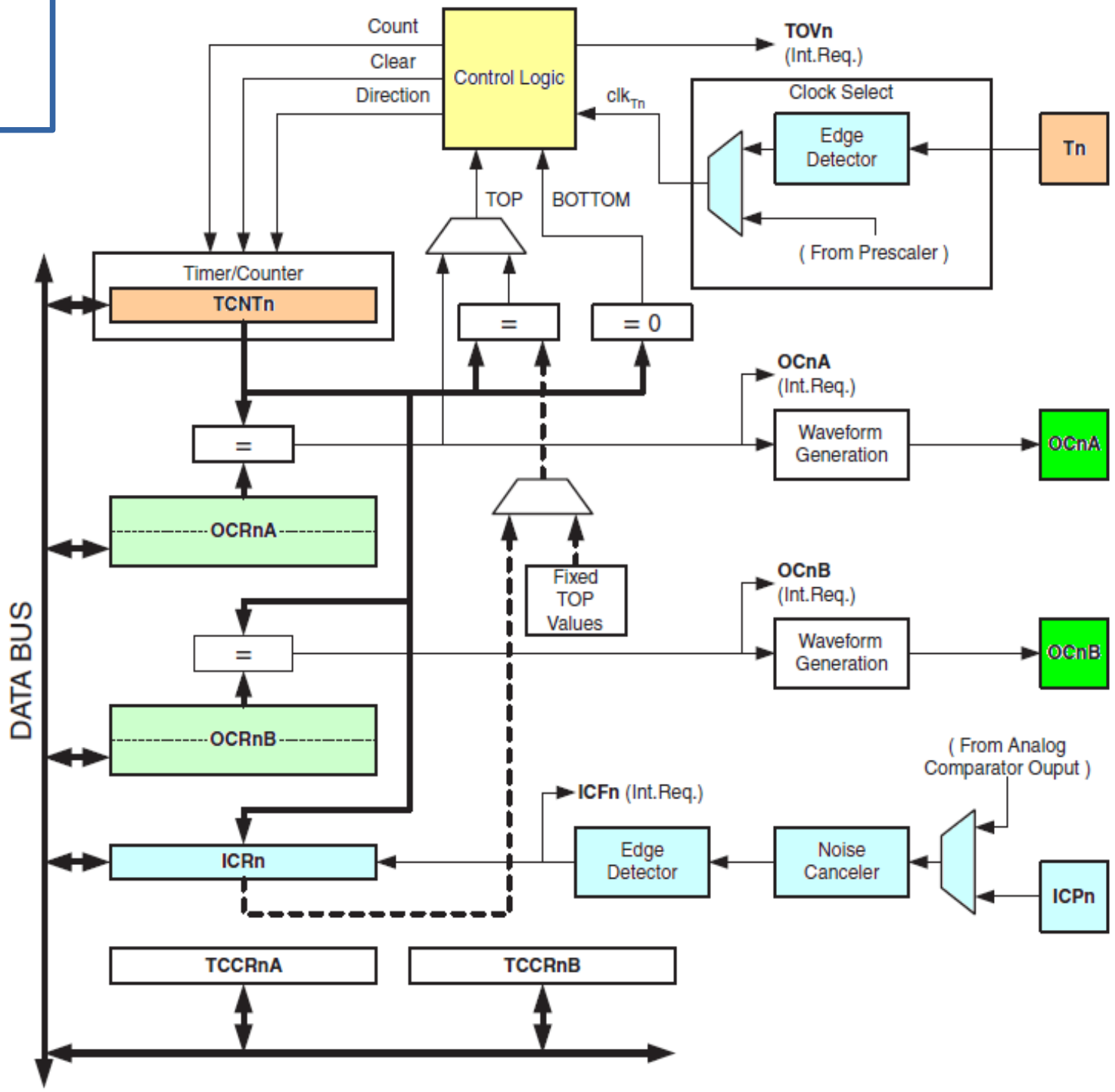
- 8-bites számláló
- Belső órajel, vagy külső jel fogadása
- 2 PWM csatorna
- Kettős bufferelés
- CTC mód *(Clear on Terminal Count)*
- Él igazított és fázishelyes PWM
- Megszakítási források: (TOV0, OCF0A, és OCF0B)



Timer1 blokkvázlata

Jellemzők:

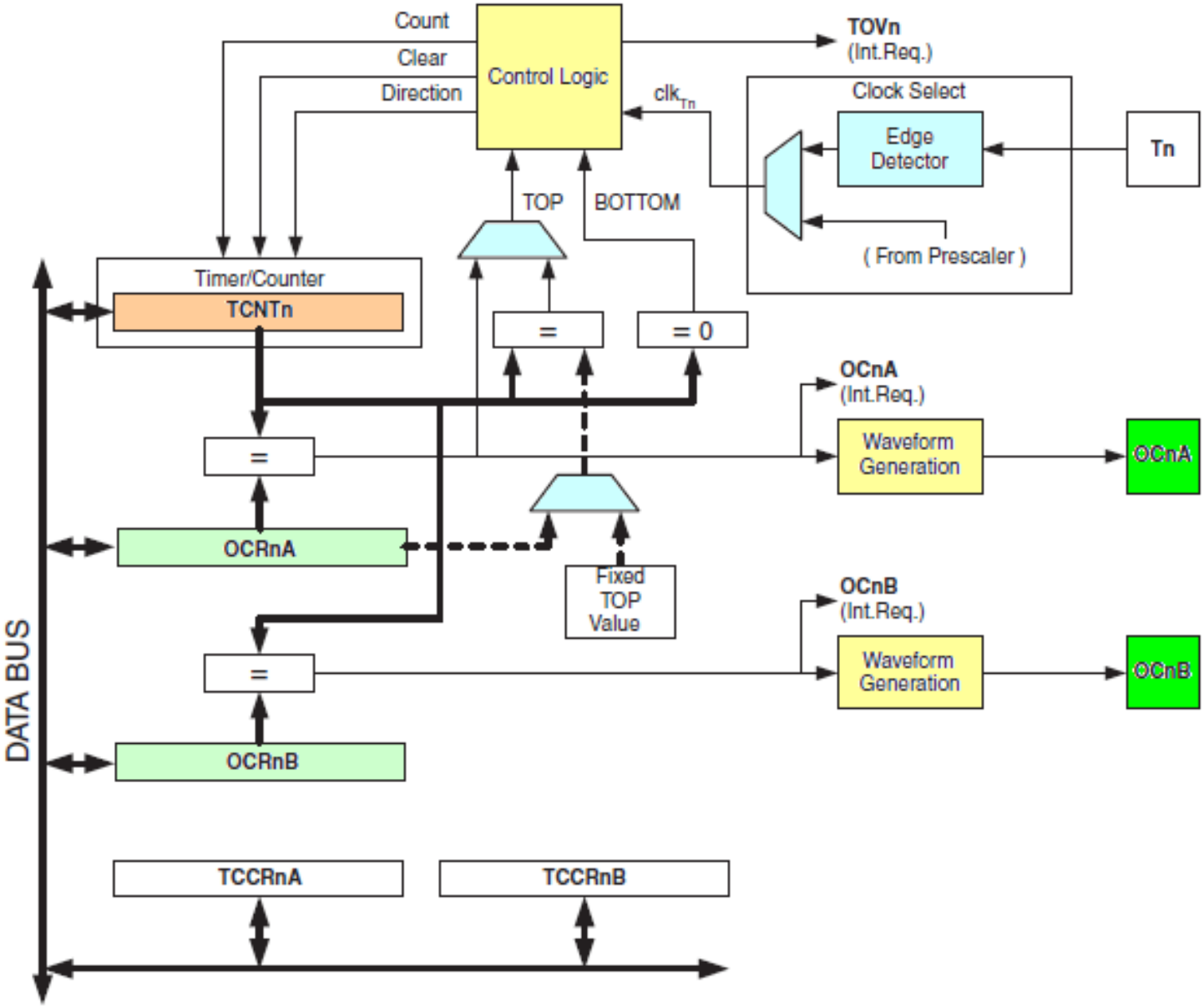
- 16-bites számláló
- Belső órajel, vagy külső jel fogadása
- 2 PWM csatorna
- Kettős bufferelés
- CTC mód *(Clear on Terminal Count)*
- Él igazított és fázishelyes PWM
- Bemeneti jelfogás
- Megszakítási források: (TOV1, ICF1, OCF1A, és OCF1B)



Timer2 blokkvázlata

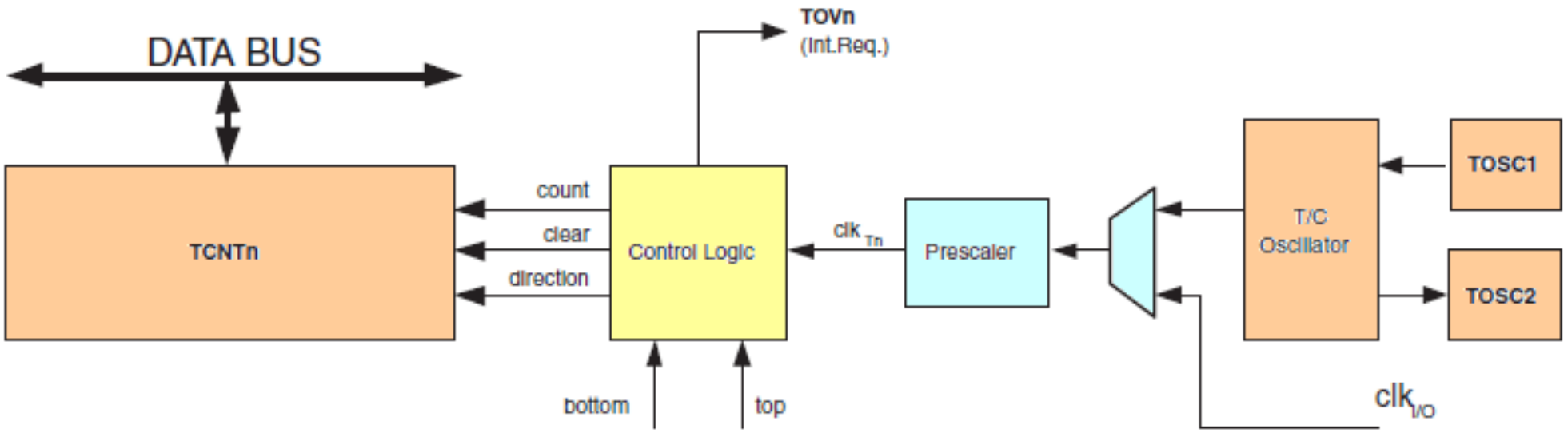
Jellemzők:

- 8-bites számláló
- 2 PWM csatorna
- Kettős bufferelés
- CTC mód *(Clear on Terminal Count)*
- Él igazított és fázishelyes PWM
- Megszakítási források: (TOV2, OCF2A, és OCF2B)



Timer2 számláló egységének blokkvázlata

- A jelforrás a rendszer órajel, vagy a 32 kHz-es RTC oszcillátor lehet (ha az **AS2** bit in az **ASSR** regiszterben = 1)
- Az előszámláló /1, /8, /32, /64, /128, /256, /1024 leosztású lehet
- **count:** a számláló egyet lép, **clear:** számláló nullázás, **direction:** a számlálás iránya (fel/le), **top:** jelzi, hogy elértük a számlálás felső határát (MAX=255, vagy az **OCR2A** regiszterben adott érték), **bottom:** jelzi, hogy elértük a számlálás alsó határát (0)



Timer2 vezérlő regiszterei

TCCR2A – TImer/Counter Control RegIster A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 18-2. Compare Output Mode, non-PWM Mode

COM2A1	COM2A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC2A on Compare Match
1	0	Clear OC2A on Compare Match
1	1	Set OC2A on Compare Match

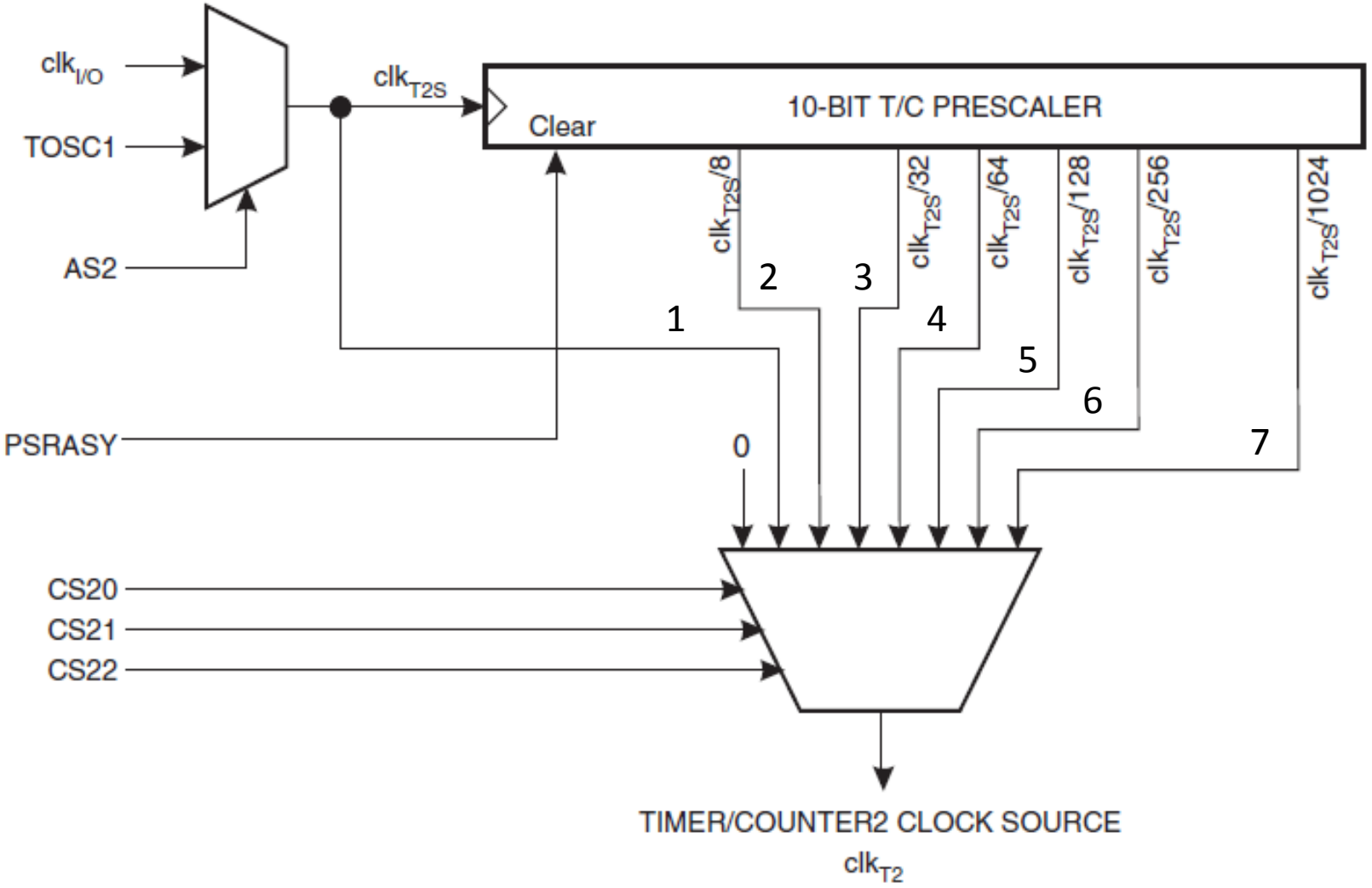
TCCR2B – TImer/Counter Control RegIster B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

FOC2A, FOC2B – force output compare (Compare esemény szoftveres generálása)

Timer2 előosztó (prescaler)

- A **TCCR2B** regiszter **CS22, CS21** és **CS20** bitjei állítják be az előosztási arányt (0 esetén nincs órajel, leáll a számlálás)



Timer2 üzemmódjai

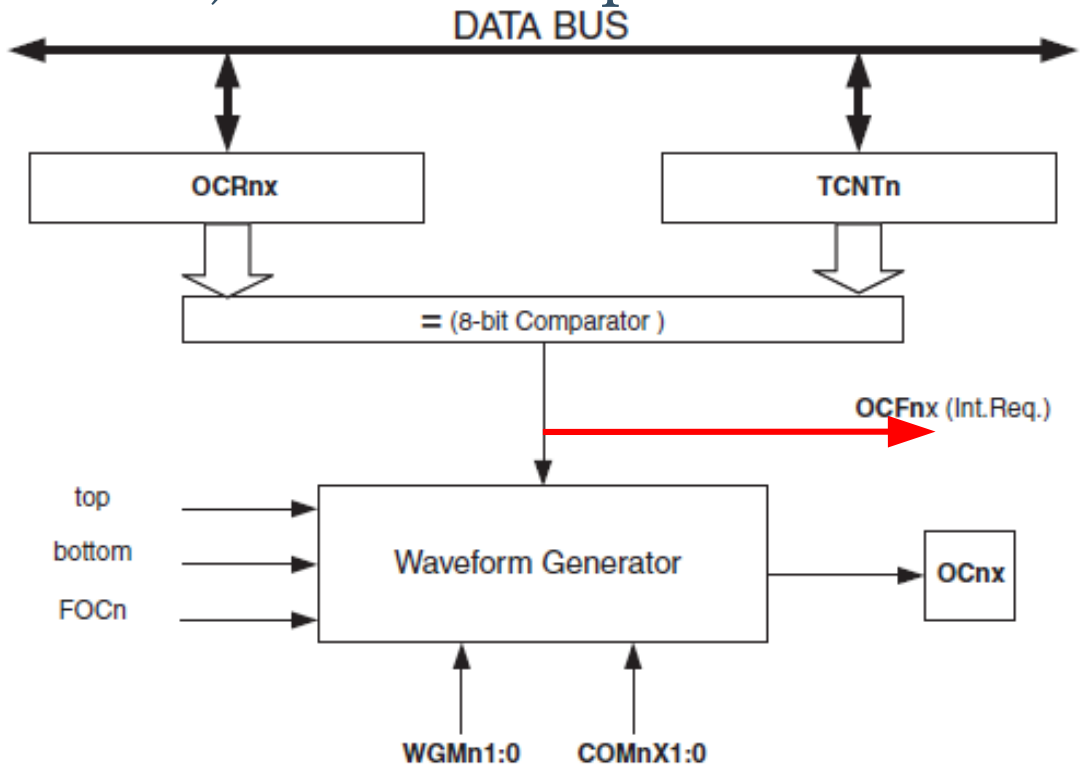
- A vezérlő regiszterek **WGM[22:20]** bitjeivel állíthatjuk be **Timer2** üzemmódját (MAX = 0xFF, Bottom = 0)

Table 18-8. Waveform Generation Mode Bit Description

Mode	WGM22	WGM21	WGM20	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Timer2: Normal mode (0)

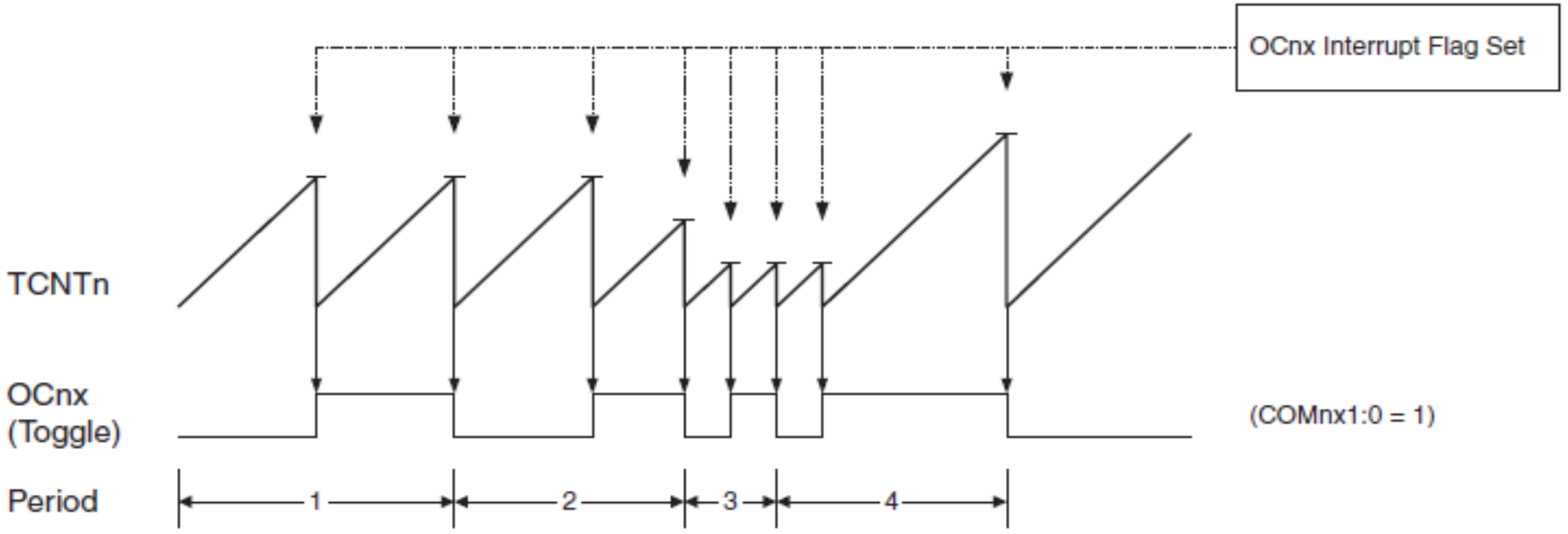
- A számláló fölfelé számlál, a számlálás felső határa 0xFF (255)
- A megszakításjelző bit (**TOV2**) akkor billen 1-be, amikor a számláló túlcsordul és 0-ra lép (**TOV2** felfogható úgy, mint a számláló 9. bitje, bár megszakításkor automatikusan törlődik)
- Megszakítások válthatók ki az összehasonlítási egységgel is (**OCF2A, OCF2B**) a számlálási perióduson belül, egyezéskor



Timer2: CTC mode (2)

- A **Clear Timer on Compare Match** (CTC) módban a számlálás felső határát az **OCR2A** regiszterben adhatjuk meg, s egyezéskor a számláló törlődik, illetve **OCF2A** megszakítást kelthetünk
- A következő TOP értéket a megszakításban átírhatjuk, de ha a számláló már meghaladta a beírt értéket, egy periódust elszalasztunk (ebben a módban nincs kettős bufferelés)

CTC Mode, Timing Diagram

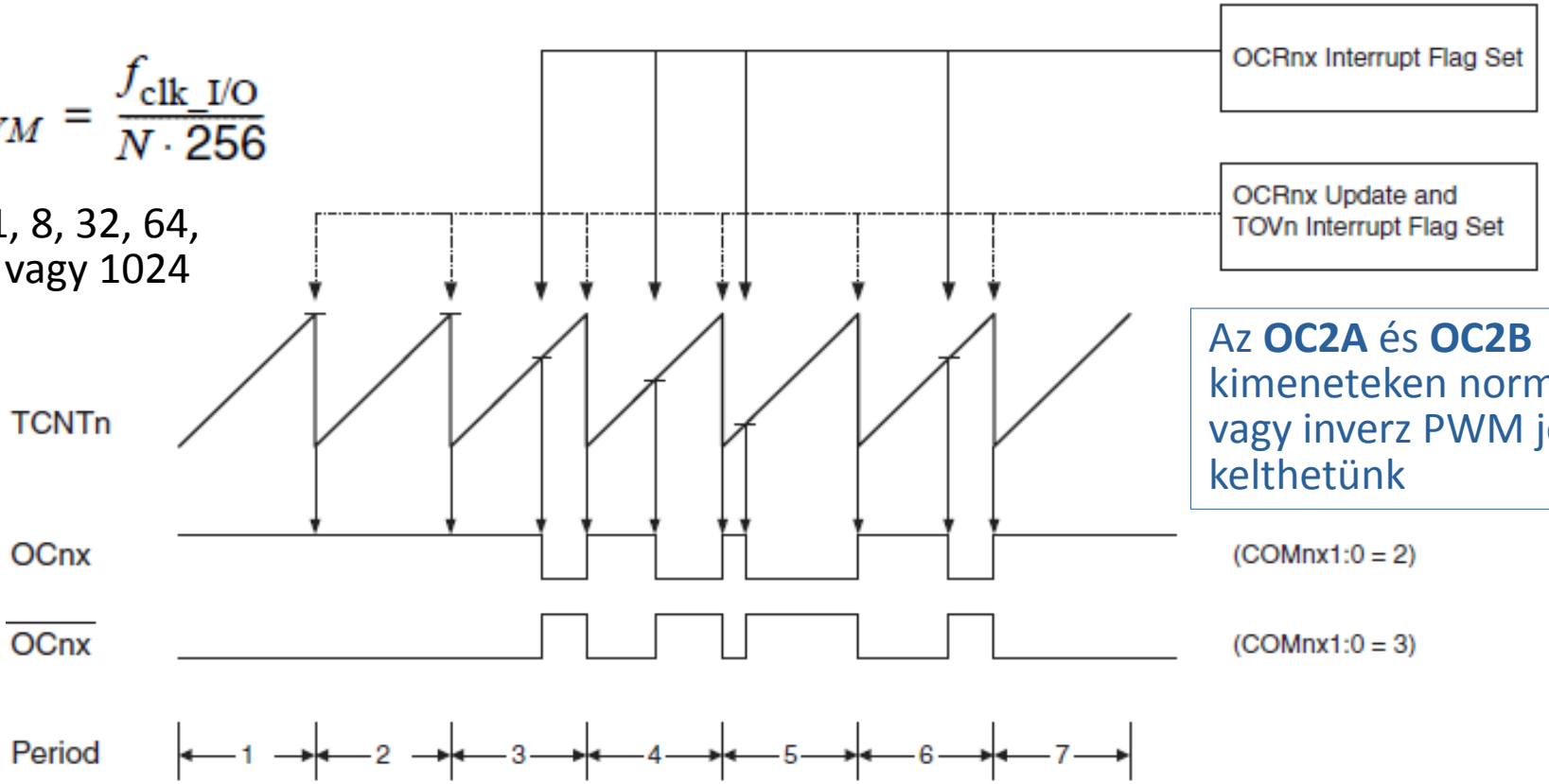


Timer2: Fast PWM mode (3, vagy 7)

- **Gyors PWM** módban a számlálás egyirányú (felfelé), a számlálás felső határa 0xFF (mode 3 esetén), illetve **OCR2A** (mode 7 esetén)
- Az egyirányú számlálás miatt nagyobb frekvenciát érhetünk el, mint a le-fel számláló fázishelyes PWM módban
- Az **OCR2*x*** regiszterek beírása itt kettős bufferelléssel történik

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot 256}$$

ahol N = 1, 8, 32, 64, 128, 256, vagy 1024



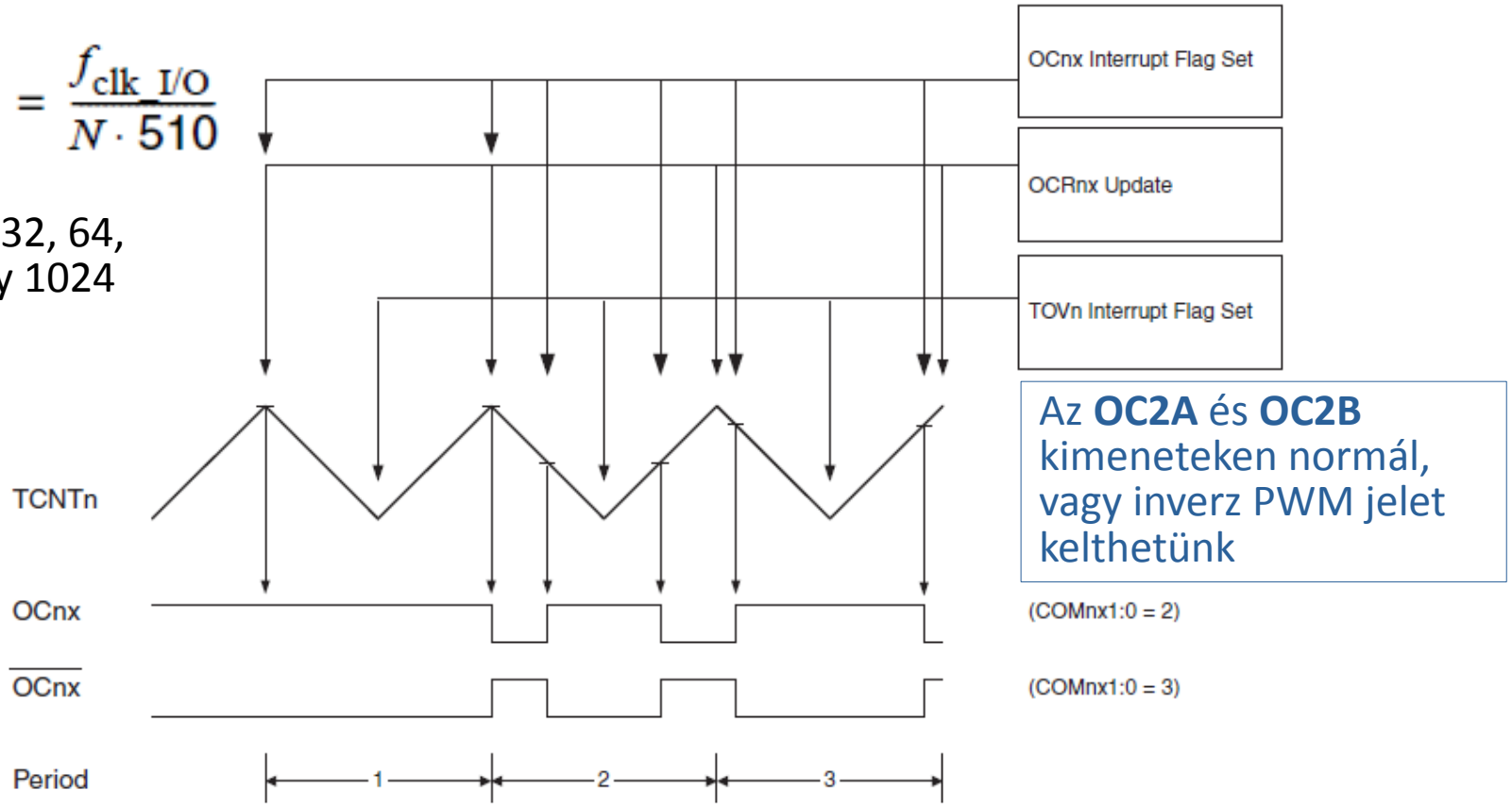
Az **OC2A** és **OC2B** kimeneteken normál, vagy inverz PWM jelet kelthetünk

Timer2: Phase correct PWM mode (1, vagy 5)

- **Fázishelyes PWM** módban a számlálás kétirányú (fel/le), a felső határa 0xFF (mode 1), illetve **OCR2A** (mode 5)
- A kimenő jel a **TOV2** eseményre szimmetrikus, azonos fázisú
- Az **OCR2*x*** regiszterek beírása itt kettős bufferelléssel történik

$$f_{OCnxPCPWM} = \frac{f_{clk\_I/O}}{N \cdot 510}$$

ahol N = 1, 8, 32, 64, 128, 256, vagy 1024



Az **OC2A** és **OC2B** kimeneteken normál, vagy inverz PWM jelet kelthetünk

Timer2 megszakítások kezelése

- **TMR2_OVF** megszakításhoz **TOIE2** = 1 feltétel kell, s beállítandó az előosztási arány. A többi vezérlő bit 0 legyen

TIMSK2 – TImer/Counter2 Interrupt Mask Register

Megszakítás engedélyezése

TOIE2 = 1

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TIFR2 – TImer/Counter2 Interrupt Flag Register

Megszakítási jelzőbitek

Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2A – TImer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2B – TImer/Counter Control Register B

Előosztás beállítása

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Timer2 megszakítások használata

- Nézzünk egy egyszerű példát: ütemezzük a beépített LED (**D13**) kapcsolgatását **Timer2** túlcsordulási megszakítás segítségével!
- Megszakítás engedélyezés: **TIMSK2_TOIE2** = 1
- Előosztó **TCCR2B_CS**[22:20] = 7 így f_{INT} = 16 000 000/(256*1024) ≈ 61 Hz (ilyen gyakorisággal billentjük át a LED állapotát)
- A megszakítási vektor neve **TIMER2_OVF_vect**

```
volatile byte state = LOW;

void setup() {
   pinMode(LED_BUILTIN, OUTPUT);              // LED kimenet inicializálása
   TIMSK2 = (TIMSK2 & B11111110) | 0x01;      // TIMER2_OVF megszakítás engedélyezése
   TCCR2B = (TCCR2B & B11111000) | 0x07;      // Előosztó 1:1024 legyen
}

void loop() {
   digitalWrite(LED_BUILTIN, state);          // A LED állapotának frissítése
}

ISR(TIMER2_OVF_vect){
   state = !state;                            // Az állapotjelző átbillentése
}
```

Forrás: teachmemicro.com/arduino-timer-interrupt-tutorial/

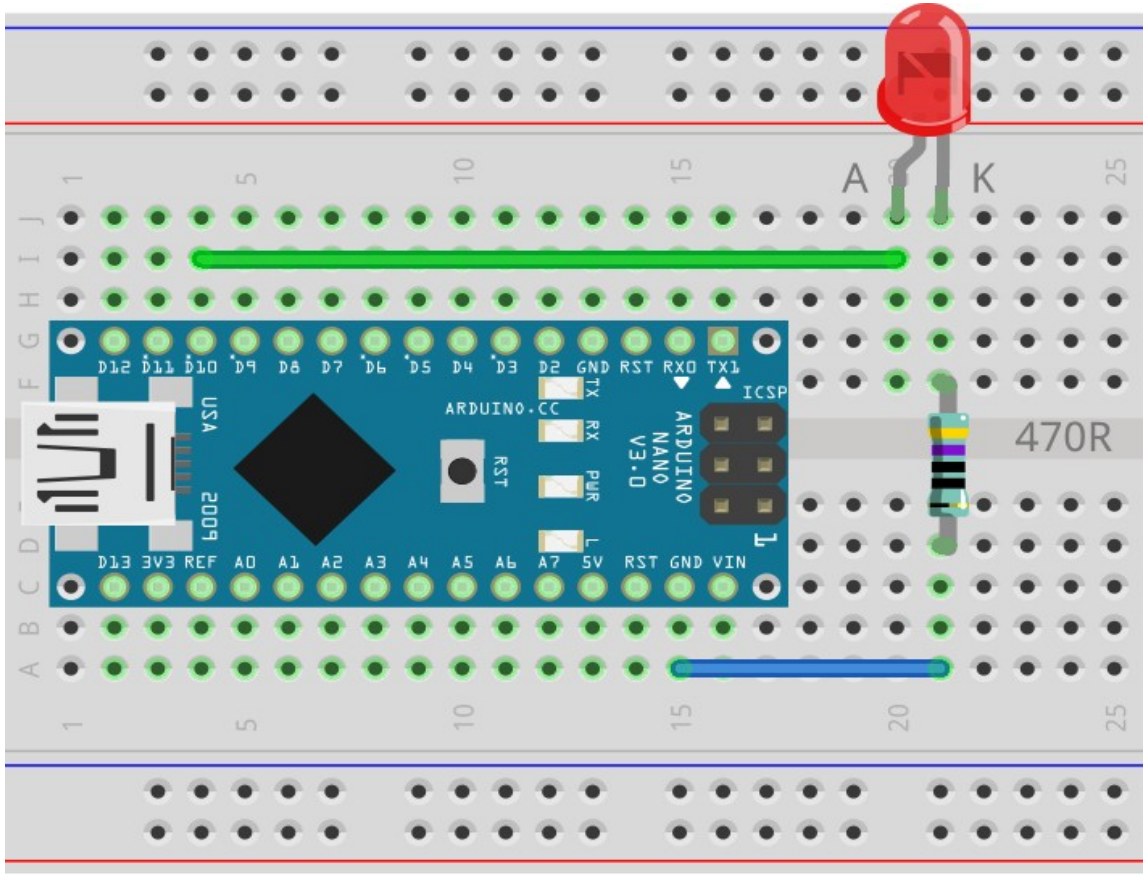
timer2_ovf.ino

- Bővítsük az előző oldali programot egy "életjelző" LED villogtatásával – így két feladat fut párhuzamosan

```
#define ledPin LED_BUILTIN                  // A beépített LED (D13)
#define heartbeat       10                  // A másik LED kivezetése (D10)
volatile byte state = LOW;                  // A beépített LED állapotjelzője

void setup() {
   pinMode(ledPin, OUTPUT);                 // LED vezérlő kimenetek inicializálása
   pinMode(heartbeat, OUTPUT);
   TIMSK2 = (TIMSK2 & B11111110) | 0x01;    // TIMER2_OVF megszakítás engedélyezése
   TCCR2B = (TCCR2B & B11111000) | 0x07;    // Előosztó 1:1024 legyen
}

void loop() {                               // TASK1: az életjelző LED villogtatása
   digitalWrite(heartbeat, LOW);
   delay(1000);
   digitalWrite(heartbeat, HIGH);
   delay(1000);
}

ISR(TIMER2_OVF_vect){                       // TASK2: a beépített LED villogtatása
   state = !state;
   digitalWrite(ledPin, state);
}
```

Kapcsolási elrendezés

- A **timer2_ovf.ino** program futtatása **Arduino Uno** és **Multifunkciós** kártyán futtatható (**LED1** és **LED4** fog villogni)
- Multifunkciós kártya híján **dugaszolós próbapanelon** is megépíthetjük a kapcsolást: egy LED anódját **D10-re,** a katódját pedig egy 470 Ω-os ellenálláson keresztül a GND-re kötjük.
  A másik LED a kártyán a beépített, L jelzésű LED lesz



Multiplex kijelzés ütemezése Timer2-vel

- A 4. előadásban a **thermometer_display.ino** programban már kidolgoztuk a multiplex kijelzés frissítését, a **refresh_display()** függvény rendszeresen meghívásával a **loop()** függvény végén

```
void refresh_display() {
  byte i = idx++ & 0x3;
  digitalWrite(LATCH_DIO, LOW);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, segment_data[i]);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, SEGMENT_SELECT[i] );
  digitalWrite(LATCH_DIO, HIGH);
}
```

- Most két lehetőséget mutatunk arra, hogy a frissítés a **Timer2** által egyenletesen ütemezve történjen, s ne függjön a **loop()** függvényben végrehajtott feladatok mennyiségétől, esetleges blokkoló várakozásaitól
  - Az **MsTimer2** programkönyvtár használatával periodikusan meghívhatunk függvényeket, s előírhatjuk az ismétlődések gyakoriságát
  - **Timer2** regiszterszintű programozásával megszakításokat kelthetünk (az előző mintaprogramhoz hasonlóan), s a megszakítási függvényben elvégezhetjük a kívánt feladatokat, vagy meghívhatunk függvényeket

Az MsTimer2 programkönyvtár

- Az MsTimer2 programkönyvtár segítségével ***ms*** felbontással végezhetünk időzítéseket a Timer2 időzítő felhasználásával. Telepítése az Arduino IDE **Tools→Manage Libraries** menüpontban is elvégezhető: írjuk be a kereső sávba MsTimer2 nevét, majd ha felbukkan az ajánlat, kattintsunk az Install gombra!
- Szerzője: *Javier Valencia*, karbantartója: *Paul Stoffregen*

Publikus függvények:

- **MsTimer2::set(***unsigned long ms, void (*f)()***)** – ezzel beállíthatjuk az ismétlődési periódust (*ms* paraméter) és hozzárendeljük a visszahívási függvényt (*f* paraméter)
- **MsTimer2::start()** – engedélyezi **Timer2** megszakításait, s elindítja az időzítési ciklust
- **MsTimer2::stop()** – letiltja **Timer2** megszakításait, s leállítja az időzítési ciklust

Mintapélda MsTimer2 használatára

- Az **MsTimer2** programkönyvtár dokumentációjában található alábbi példaprogram bemutatja annak használatát: a beépített LED-et villogtatjuk az **MsTimer2** programkönyvtár felhasználásával, az állapotváltások 500 *ms*-onként történnek

```
#include <MsTimer2.h>

void flash() {                          // A visszahívási függvény
  static boolean output = HIGH;         // statikus állapotjelző változó
  digitalWrite(LED_BUILTIN, output);    // LED állapot frissítése
  output = !output;                     // Az állapotjelző átbillentése
}

void setup() {
  pinMode(LED_BUILTIN, output);         // Beépített LED konfigurálása
  MsTimer2::set(500, flash);            // 500ms periódus
  MsTimer2::start();                    // Időzítés indítása
}

void loop() { }                         // Nincs feladat
```

1. Multiplex kijelzés MsTimer2 használatával

- A 4. előadásban bemutatott **thermometer_display.ino** programot most átdolgozzuk úgy, hogy egy kicsit általánosabb legyen:
  - Az **A4** analóg bemenetre kapcsolt feszültséget mérjük, átlagoljuk és a mV-ra átszámolt feszültséget a **Multifunkciós kártya** 7-szegmenses kijelzőjén íratjuk ki
  - a multiplex kijelzés frissítését (a **refresh_display()** függvény rendszeres meghívását) az **MsTimer2** programkönyvtár segítségével ütemezzük
- Lényegében egy digitális (milli)voltmérőt készítünk ...

voltmeter_display.ino 2/1. oldal

```
#include <MsTimer2.h>

/* A kijelzőt vezérlő lábak definiálása */
#define LATCH_DIO 4
#define CLK_DIO 7
#define DATA_DIO 8

/* Szegmensrajzolat a 0-9 számjegyekhez */
const byte SEGMENT_MAP[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90};
/* Az 1-4 számjegyek kiválasztása */
const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};
byte segment_data[] = {0xFF,0xFF,0xFF,0xFF}; // adattároló
byte counter = 50;                           // Hány mérést végezzünk
byte idx = 0;                                // Kijelzés számlálója
long nexttime;                               // Időbélyeg mérés indításhoz
long sum = 0;                                // Összegző változó átlagoláshoz

void setup () {
  analogReference(DEFAULT);
  pinMode(LATCH_DIO, OUTPUT);
  pinMode(CLK_DIO, OUTPUT);
  pinMode(DATA_DIO, OUTPUT);
  nexttime = millis();
  MsTimer2::set(5, refresh_display);         // 5 ms period
  MsTimer2::start();
}
```

voltmeter_display.ino 2/2. oldal

```
void loop() {
  long voltage;
  if (millis() > nexttime) {          // TASK1 100 ms-onként fut le
    sum += analogRead(A4);
    if (--counter == 0) {
      voltage = sum * 100 / 1024;     // Feszültség mV-ban (sum 50 mérés összege)
      sum = 0;                        // Töröljük az összegző változót
      counter = 50;                   // 50 konverzió eredményét átlagoljuk
      segment_data[0] = SEGMENT_MAP[voltage / 1000];
      segment_data[1] = SEGMENT_MAP[(voltage / 100) % 10];
      segment_data[2] = SEGMENT_MAP[(voltage / 10) % 10];
      segment_data[3] = SEGMENT_MAP[voltage % 10];
    }
    nexttime += 10;                   // 10 ms-onként végzünk konverziót
  }
}

/* A soron következő számjegy megjelenítése */
void refresh_display() {
  byte i;
  i = idx++ & 0x3;
  digitalWrite(LATCH_DIO, LOW);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, segment_data[i]);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, SEGMENT_SELECT[i] );
  digitalWrite(LATCH_DIO, HIGH);
}
```

2. Multiplex kijelzés Timer2 megszakításban

- A feladat hasonló az előző programhoz, csak most mellőzzük az **MsTimer2** programkönyvtár használatát és közvetlenül programozzuk a **Timer2** időzítőt:
  - Az **A4** analóg bemenetre kapcsolt feszültséget mérjük, átlagoljuk és a mV-ra átszámolt feszültséget a **Multifunkciós kártya** 7-szegmenses kijelzőjén íratjuk ki
  - a multiplex kijelzés frissítését most a **Timer2** időzítő túlcsordulási megszakításai felhasználásával ütemezzük
- Nem tartozik a kitűzött feladathoz, de a programban „életjelző funkcióként” (heartbeat) villogtatjuk a beépített LED-et (ennek villogásából láthatjuk, hogy nem akadt el a program)
- Mellékesen azt is megmutatjuk, hogy hogyan tudunk tizedespontot kiírni, azaz hogyan tudjuk *mV* helyett az eredmény *V*-ban kiíratni

voltmeter_display2.ino 3/1. oldal

```
#define LATCH_DIO 4                          /* A kijelzőt vezérlő lábak definiálása */
#define CLK_DIO   7
#define DATA_DIO  8
#define ledPin   LED_BUILTIN                 /* Heartbeat LED */
/* Szegmensrajzolat a 0-9 számjegyekhez */
const byte SEGMENT_MAP[] = {0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8, 0X80, 0X90};
const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8}; // számjegyek kiválasztása
volatile byte segment_data[] = {0,0,0,0}; // adattároló buffer
byte counter = 50;                           // Ennyi mérést átlagolunk
byte idx = 0;                                // Kijelzés számlálója
byte ledstate = 0;                           // Heartbeat LED állapota
long nexttime;                               // Időbélyeg mérés indításhoz
long blinktime;                              // Időbélyeg hearbeat LED átkapcsoláshoz
long sum = 0;                                // Összegző változó átlagoláshoz

void setup() {
  pinMode(ledPin, OUTPUT);
  analogReference(DEFAULT);
  pinMode(LATCH_DIO, OUTPUT);                /* Vezérlő lábak konfigurálása */
  pinMode(CLK_DIO, OUTPUT);
  pinMode(DATA_DIO, OUTPUT);
  nexttime = millis();
  blinktime = millis();
  /* Timer2 konfigurálás */
  TIMSK2 = (TIMSK2 & B11111110) | 0x01;      // TIM2_OVF megszakítás engedélyezés
  TCCR2B = (TCCR2B & B11111000) | 0x05;      // előosztó = 1 : 128
}
```

voltmeter_display2.ino 3/2. oldal

```
void loop() {
  long thistime = millis();                 // Ciklusonként csak egyszer olvassuk ki...
  if (thistime > nexttime) {                // TASK1 100 ms-onként fut le
    sum += analogRead(A4);                  // Újabb mérés indítása
    if (--counter == 0) {                   // Ha összegyűlt 50 adat...
      long voltage = sum * 100 / 1024;      // Feszültség mV-ban
      sum = 0;                              // Töröljük az összegző változót
      counter = 50;                         // újabb 50 márást jegyzünk elő
      /* Eredmény kiírása a bufferbe */
      segment_data[0] = SEGMENT_MAP[voltage / 1000] & 0x7F; // Tizedesponttal...
      segment_data[1] = SEGMENT_MAP[(voltage / 100) % 10];
      segment_data[2] = SEGMENT_MAP[(voltage / 10) % 10];
      segment_data[3] = SEGMENT_MAP[voltage % 10];
    }
    nexttime += 10;                         // 10 ms-onként végzünk konverziót
  }

  if (thistime > blinktime ) {              // TASK2: heartbeat LED villogtatása
    blinktime += 1000;                      // következő állapotváltás 1 s múlva
    ledstate = !ledstate;                   // állapotváltás
    digitalWrite(ledPin, ledstate);         // állapotváltás érvényesítése
  }
}
```

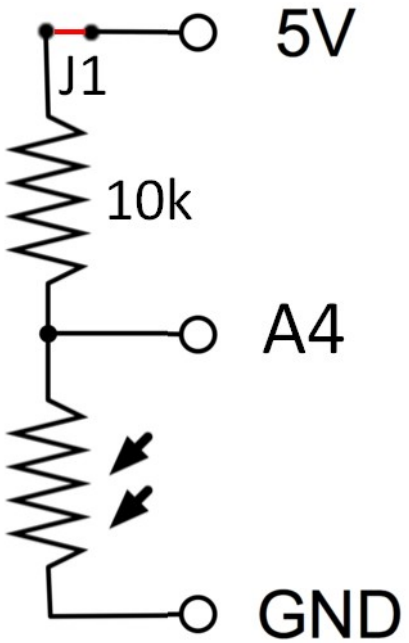
voltmeter_display2.ino 3/3. oldal

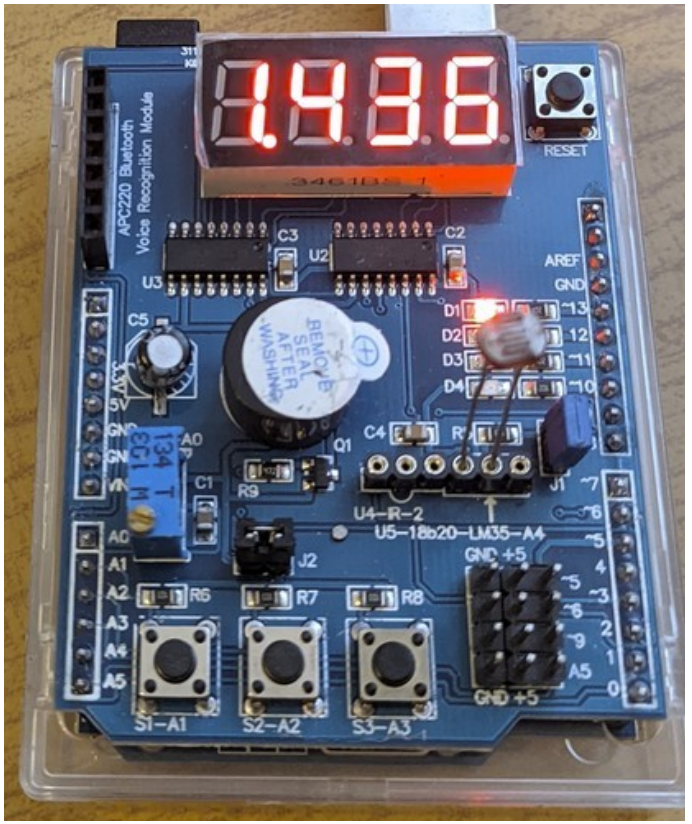
```
/* Timer2 túlcsordulás megszakításkiszolgáló eljárás
 * amelyben felvillantjuk a számkijelző egyik számjegyét
 */
ISR(TIMER2_OVF_vect) {
  byte i = idx++ & 0x3;
  digitalWrite(LATCH_DIO, LOW);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, segment_data[i]);
  shiftOut(DATA_DIO, CLK_DIO, MSBFIRST, SEGMENT_SELECT[i] );
  digitalWrite(LATCH_DIO, HIGH);
}
```

- A **refresh_display()** függvény törzsét nem muszáj bemásolni a megszakításkiszolgáló eljárásba – elég lenne meghívni
- A függvényhívás azonban a visszatérési cím elmentése majd visszaállítása miatt fölösleges többletterhet jelentene

Fénymérés fényérzékeny ellenállással

- Az előző előadásban bemutatott fényérzékeny ellenállást a **Multi-funkciós kártyához** is csatlakoztathatjuk, az **LM35** hőmérő helyére
- A kapcsolás feszültségosztóként működik, amelyikben az alsó tag egy CdS fényérzékeny ellenállás, a felső tag pedig a kártyára épített 10 kΩ-os ellenállás (a **J1** átkötést zárni kell!)
- Az osztó közös pontja az **A4** analóg bemenetre van kötve





Az Arduino nano kártya kivezetései