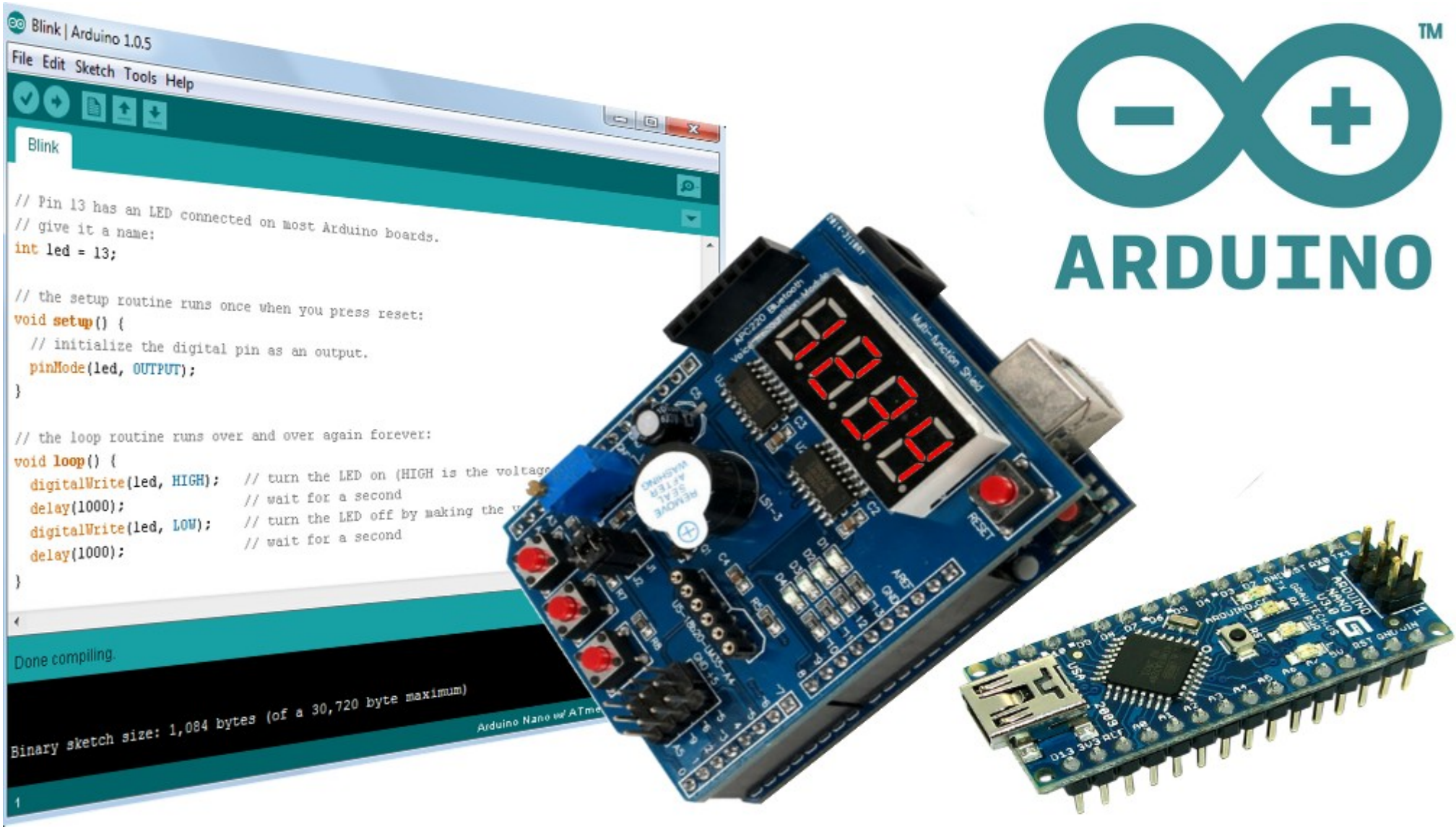
Arduino tanfolyam kezdőknek és haladóknak



5. Az analóg-digitális átalakító (ADC) használata – 2. rész

Felhasznált irodalom

- Microchip (ATMEL): **ATmega328p adatlap**
- Meettechniek: **Arduino Analog measurements**
- Glenn Sweeney:
  Interrupt-Driven Analog Conversion With an ATMega328p
- Nick Gammon: **ADC conversion on the Arduino (analogRead)**

A switch utasítás

- A **switch** utasítás egy kifejezés értékét több, egész értékű állandó értékével hasonlítja össze, és azt az ágat hajtja végre, amelyiknél egyezést talál. Ha nincs egyezés, a `default:` ág kerül végrehajtásra
- A switch utasítás általános felépítése:

```
switch (kifejezés) {
    case állandó kifejezés: utasítások; break
    case állandó kifejezés: utasítások; break
    . . .
    default: utasítások
}
```

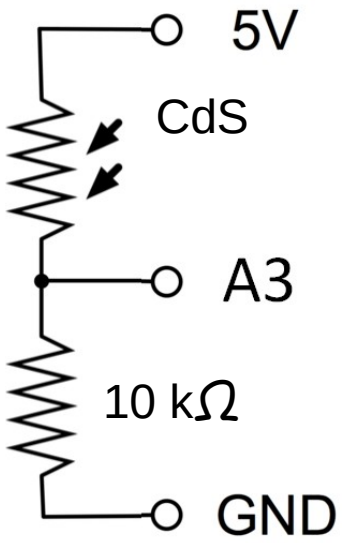
A **break** szerepe az, hogy kiugorjon a **switch** utasítás törzséből

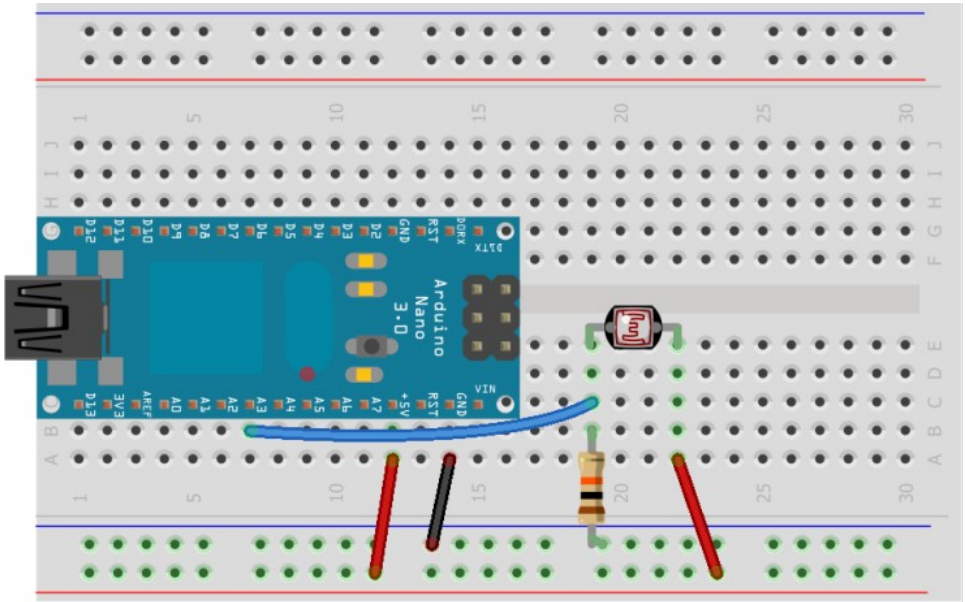
- Egy példa: LED ki-, be-, vagy átkapcsolása

```
switch (led_state) {
  case 0: digitalWrite(RED_LED, LOW); break;
  case 1: digitalWrite(RED_LED, HIGH); break;
  default: digitalWrite(RED_LED, !digitalRead(RED_LED));
}
```

Fénymérés fényérzékeny ellenállással

- A kapcsolás feszültségosztóként működik, amelyikben a felső tag egy CdS fényérzékeny ellenállás, melynek ellenállása a megvilágítástól függően széles határok között változik. A megvilágítás hatására az ellenállása csökken…
- Az ellenállásosztó közös pontját az **A3** analóg bemenetre kötöttük
- Az eredményt itt most szöveges értékelés formájában a soros porton kiküldjük a számítógépre, s közben a **switch** utasítást használatát is gyakoroljuk





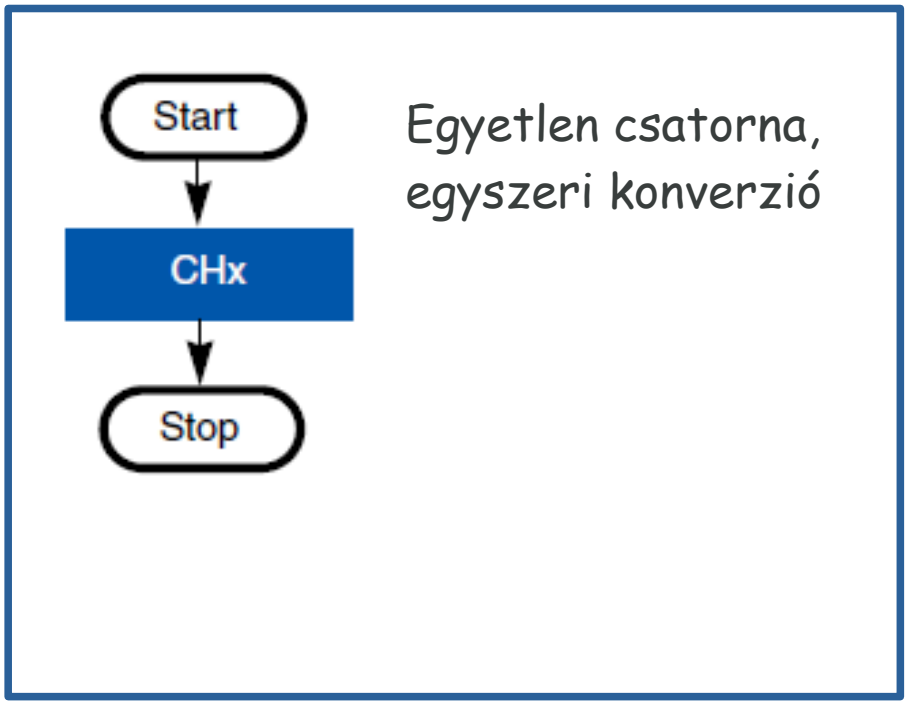
switch_case.ino

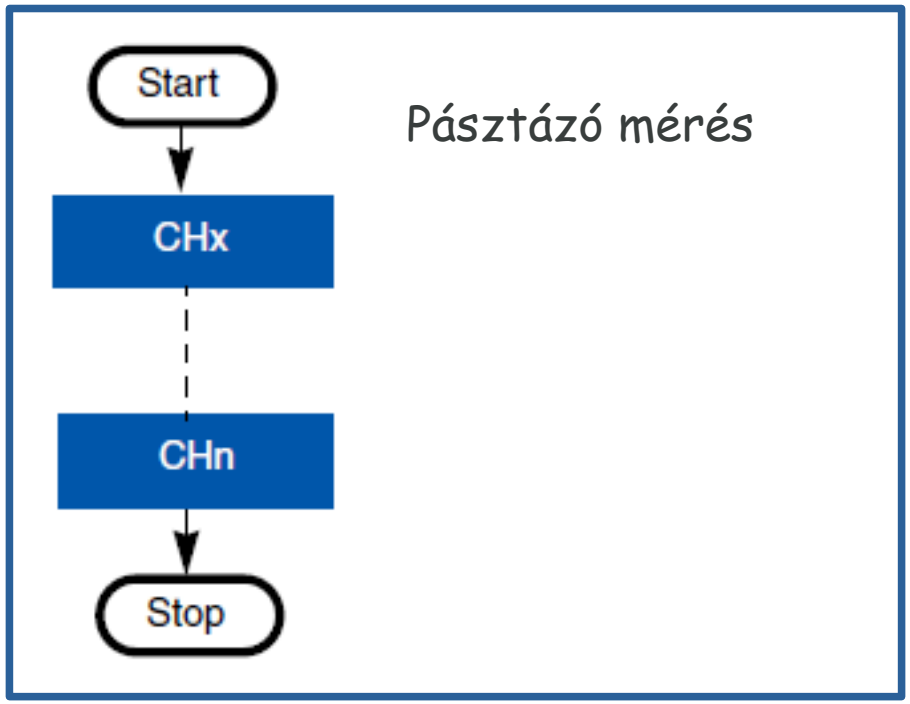
- A fénymérés eredményének szöveges kiértékelése

```
const int sensorMin = 0;          // szenzor minimum (tapasztalat alapján)
const int sensorMax = 800;        // szenzor maximum (tapasztalat alapján)

void setup() {
  Serial.begin(9600);             // Soros port inicializálása
}

void loop() {
  int reading = analogRead(A3); // Szenzor kiolvasás
  //--- A mérési tartomány leképezése négy esetre ---
  int range = map(reading,sensorMin,sensorMax, 0, 3);
  //--- Elágazás az eredmény szerint ----------------
  switch (range) {
    case 0:     // Eltakart szenzor esetén
      Serial.println("sötét"); break;
    case 1:     // kezed közel a szenzorhoz
      Serial.println("derengő fény"); break;
    case 2:     // kezed 10 cm-re a szenzortól
      Serial.println("közepes fény"); break;
    case 3:     // kezed nincs a fény útjában
      Serial.println("erős fény"); break;
  }
  delay(1000);  // Késleltetés a kiíratások között
}
```

Pásztázó mérés több csatornában

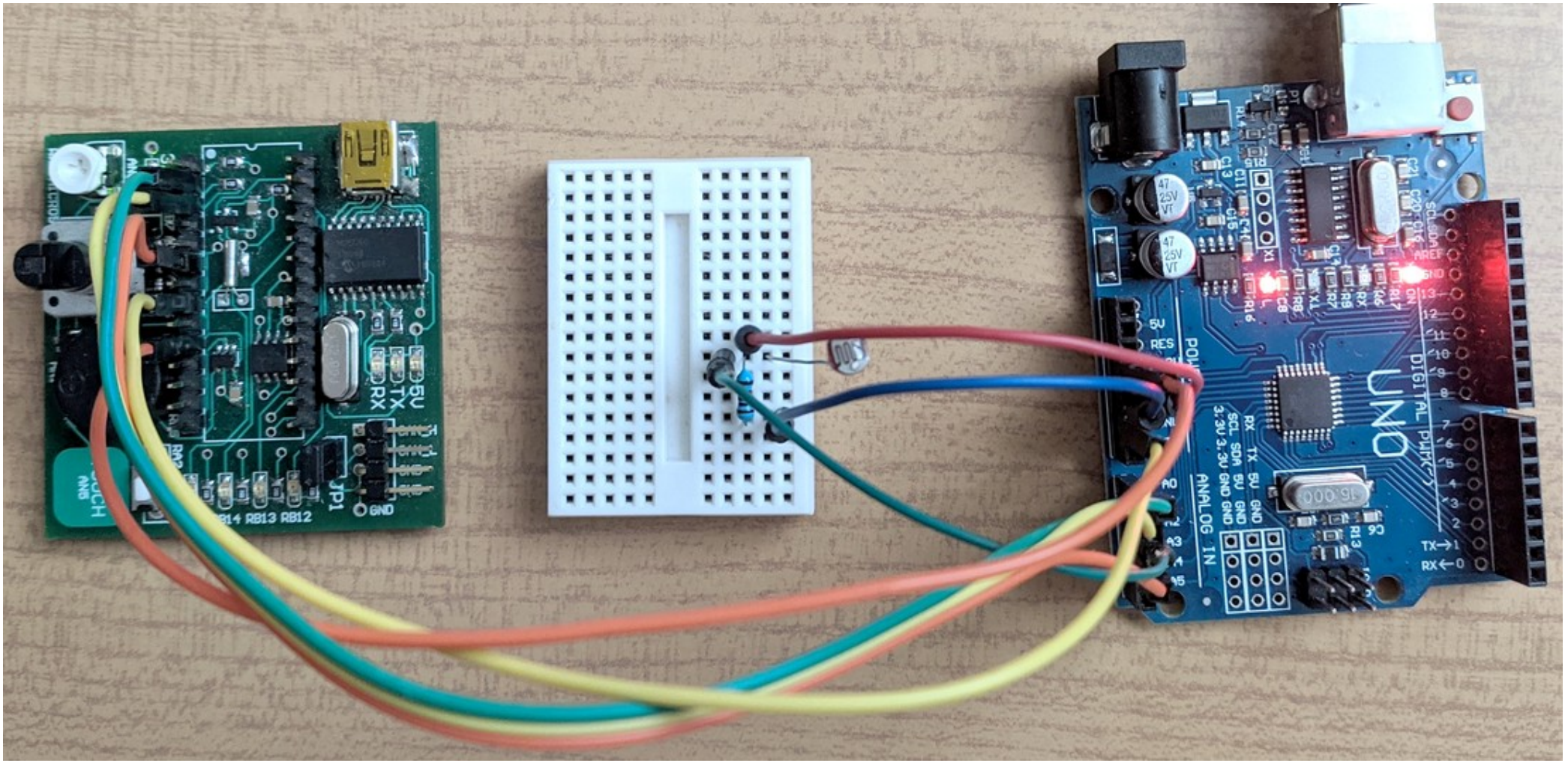
- Az eddigiekben egyetlen csatorna egyszeri konverzióját ismételgettük az **analogRead()** függvény hívásával
- Egymás után több csatornában is mérhetünk, ha az **analogRead()** függvényt más-más csatornaszámmal hívjuk, így több csatornát is pásztázhatunk – kvázi szimultán





adc_scan: pásztázó mérés több csatornában

- **A0:** potméterrel beállított fesz., **A1: MCP9700** analóg hőmérő, **A2:** LDR és 10 kΩ osztó (fénymérő), **A3:** 2.5 V referencia forrás (a képen egy **Microstick Plus** kártya, kiegészítve az LDR-es osztóval)



adc_scan.ino 2/1

- Az alábbi programban négy csatorna jelét mérjük pásztázó módban

```
/* adc_scan

   Pásztázó mérés az ADC-vel: több csatornát pásztázunk
    - az A0 bemeneten egy potméterrel leosztott feszültséget mérjük
    - az A1 bemeneten egy MCP9700 analóg hőmérő jelét mérjük
    - az A2 bemeneten egy LDR segítségével a fényt mérjük
    - az A3 bemeneten pedig egy referencia feszültséget mérünk

   Az eredményeket feszültségre átszámítva a soros porton kiíratjuk.
*/

#define VDD   4.97                  // Vdd feszültség [V]
int data[4] = {0, 0, 0, 0};         // Mérési adatok tárolója

void setup() {
  Serial.begin(115200);             // Adatsebesség = 115 200 baud
  analogReference(DEFAULT);         // Vref = kb. 5V (a tápfeszültség)
  Serial.println("  adc_scan: Pásztázó mérés ADC-vel");
  Serial.println("  - az A0 bemeneten egy potméterrel leosztott feszültségét mérjük");
  Serial.println("  - az A1 bemeneten egy MCP9700 analóg hőmérő jelét mérjük");
  Serial.println("  - az A2 bemeneten egy LDR segítségével a fényt mérjük");
  Serial.println("  - az A3 bemeneten pedig egy referencia feszültséget mérünk");
}
```

Folytatás a következő oldalon ...

adc_scan.ino 2/2

```
void loop() {
  for (int i = 0; i < 4; i++) {
    data[i] = analogRead (i);                    // Pásztázás
  }
  for (int i = 0; i < 4; i++) {
    float voltage = (data[i] * VDD) / 1024.0;
    Serial.print("  A");
    Serial.print(i);
    Serial.print(" = ");
    Serial.print(voltage, 2);
    Serial.print(" V ");
    switch (i) {
      case 1:  { float tempC = (voltage - 0.5) / 0.01; // MCP9700 (500mV, 10 mV/°C)
        Serial.print(tempC, 1);
        Serial.println(" °C");
        break; }
      case 2:  { float ldr = 10 * VDD / voltage - 10;  // LDR ellenállás kiszámítása
        Serial.print(ldr, 3);
        Serial.println(" k");
        break;}
      default: Serial.println(" ");
    }
  }
  Serial.println("  ----------");
  delay(2000);
}
```

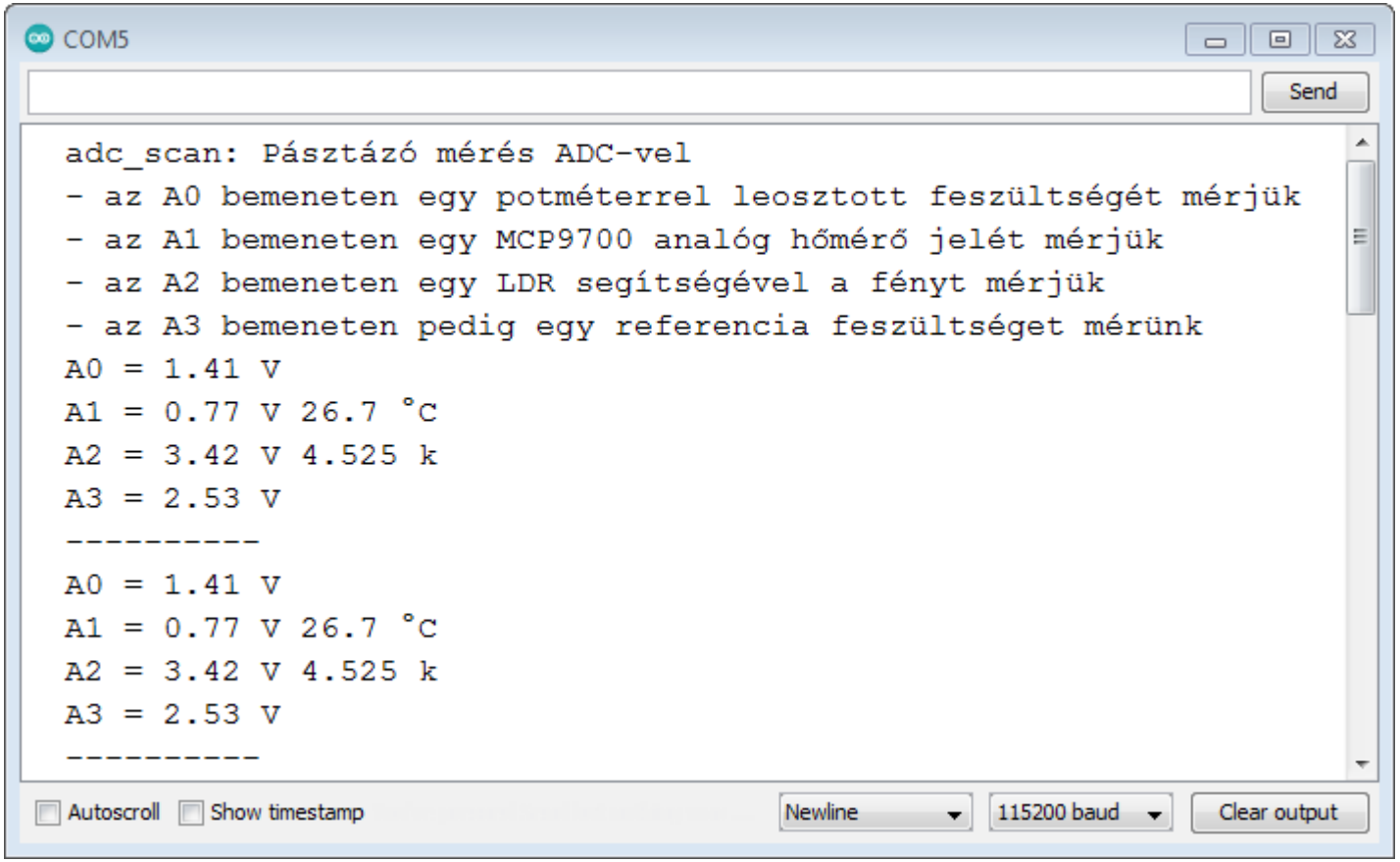
Itt is használjuk a switch utasítást

Így is irhattuk volna:

```
if (i == 1) { utasítások1 }
else if (i == 2) { utasítások2 }
else { utasítások3 }
```

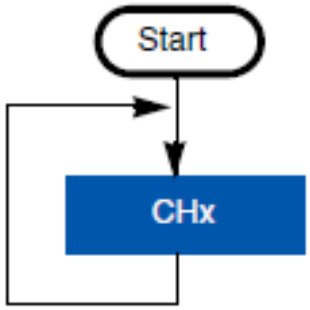
adc_scan.ino futási eredmény

- Az alábbi ábrán láthatjuk a program futási eredményét, ahogy az Arduino IDE **Serial terminal** ablakában megjelenik
- **A0** értéke a potméter állásától függ, **A1** a hőmérséklettől, **A2** a megvilágítástól, **A3** pedig a 2.50 V névleges feszültségű referencia



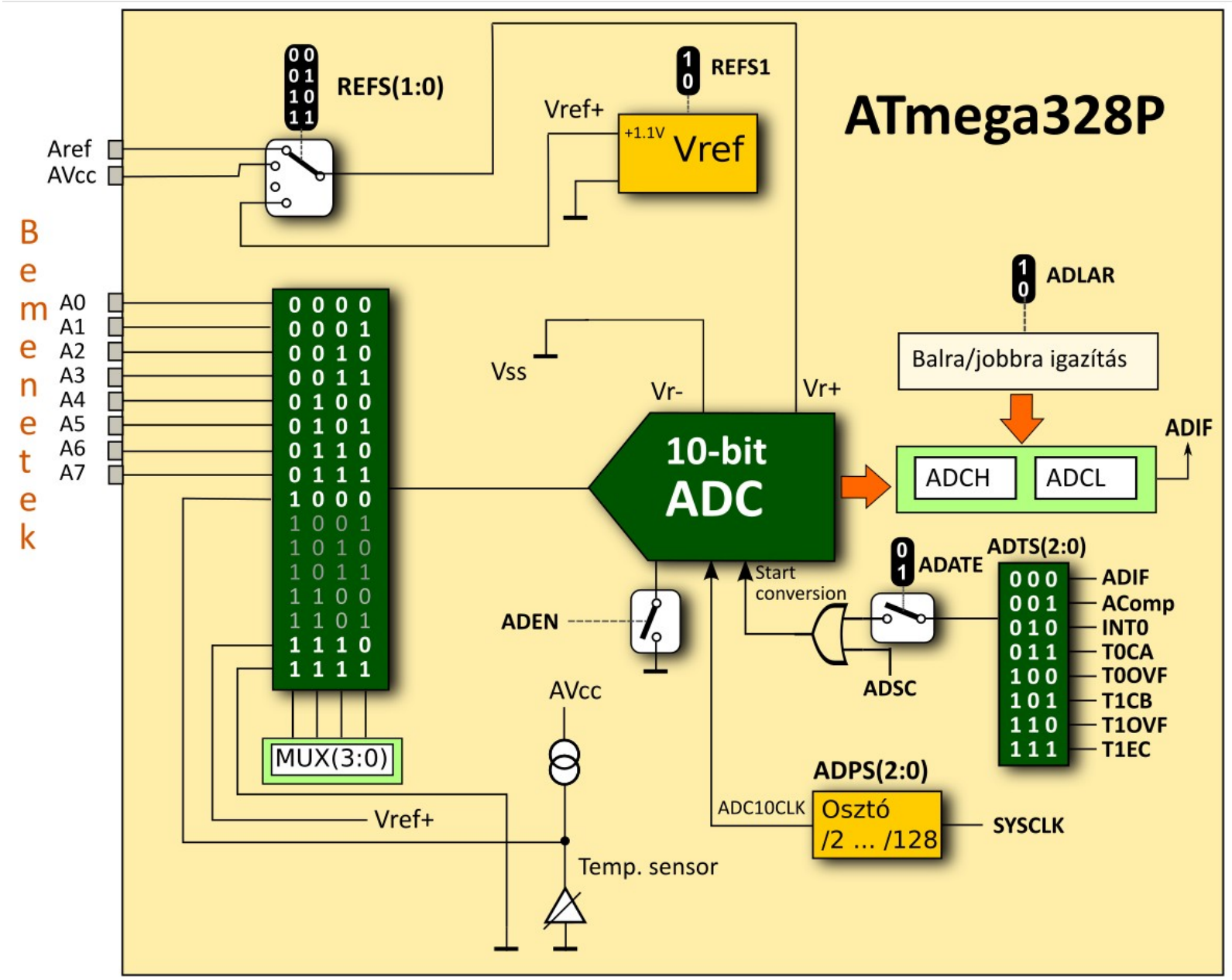
Az ADC regiszterszintű programozása

- Amint láttuk, az **analogRead()** függvénnyel csak szoftveresen indított egyszeri konverziókat tudunk végezni
- Az **ATmega328p** mikrovezérlő regiszterszintű programozásával további **ADC** üzemmódokat tudunk használatba venni:
- **Folyamatos konverzió** - amikor az **ADIF** jel újabb konverziót indít)



- **ADC kiolvasása programmegszakításban** – az ADC a konverzió végén megszakítást kelt, így nem szükséges folyamatosan lekérdezgetni az állapotát, blokkoló vagy nem blokkoló várakozásban
- **Hardver eseménnyel triggerelt konverzió** – amikor külső eseménnyel (**INT0**) vagy belső periféria jelével indítjuk/ütemezzük a konverziót

Az ADC blokkvázlata



Az ADC regiszterei

- **ADMUX** – referenciafeszültség és bemeneti csatorna választó

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

REFS – 00: `EXTERNAL`, 01: `DEFAULT` (Avcc), 11: `INTERNAL` (1,1V)
ADLAR – Az eredmény igazítása **0:** jobbra, **1:** balra
MUX – bemenetválasztás (**0000-0111:** A0-A7, **1000:** belső hőmérő, **1110:** 1,1V-os referencia, **1111:** GND)

- **ADCH** és **ADCL** adatregiszterek

ADLAR = 0

15	14	13	12	11	10	9	8	
–	–	–	–	–	–	ADC9	ADC8	ADCH
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
7	6	5	4	3	2	1	0	

ADLAR = 1

15	14	13	12	11	10	9	8	
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	–	–	–	–	–	–	ADCL
7	6	5	4	3	2	1	0	

Az ADC regiszterei

- **ADCSRA** – vezérlő és állapotregiszter I.

7	6	5	4	3	2	1	0	
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

ADEN: ADC engedélyezése, **ADSC:** konverzió indítás, **ADATE:** automatikus triggerelés engedélyezése, **ADIF:** konverzió vége jelzőbit, **ADIE:** megszakításkérés engedélyezés, **ADPS[2:0]:** előszámláló választás (/2 … /128)

- **ADCSRB** – vezérlő és állapotregiszter II.

7	6	5	4	3	2	1	0	
–	ACME	–	–	–	ADTS2	ADTS1	ADTS0	ADCSRB
R	R/W	R	R	R	R/W	R/W	R/W	
0	0	0	0	0	0	0	0	

ACME – az analóg komparátorhoz rendeli a bemeneti multiplexert (ADEN = 0 esetén)
ADTS[2:0] – a konverziót indító (trigger) jelforrás kiválasztása:

000: Szabadonfutó mód (ADIF)	100: Timer0 Túlcsordulás
001: Analóg komparátor	101: Timer1 Compare esemény B
010: Külső megszakítás (INT0)	110: Timer1 Túlcsordulás
011: Timer0 Compare esemény A	111: Timer1 Capture esemény

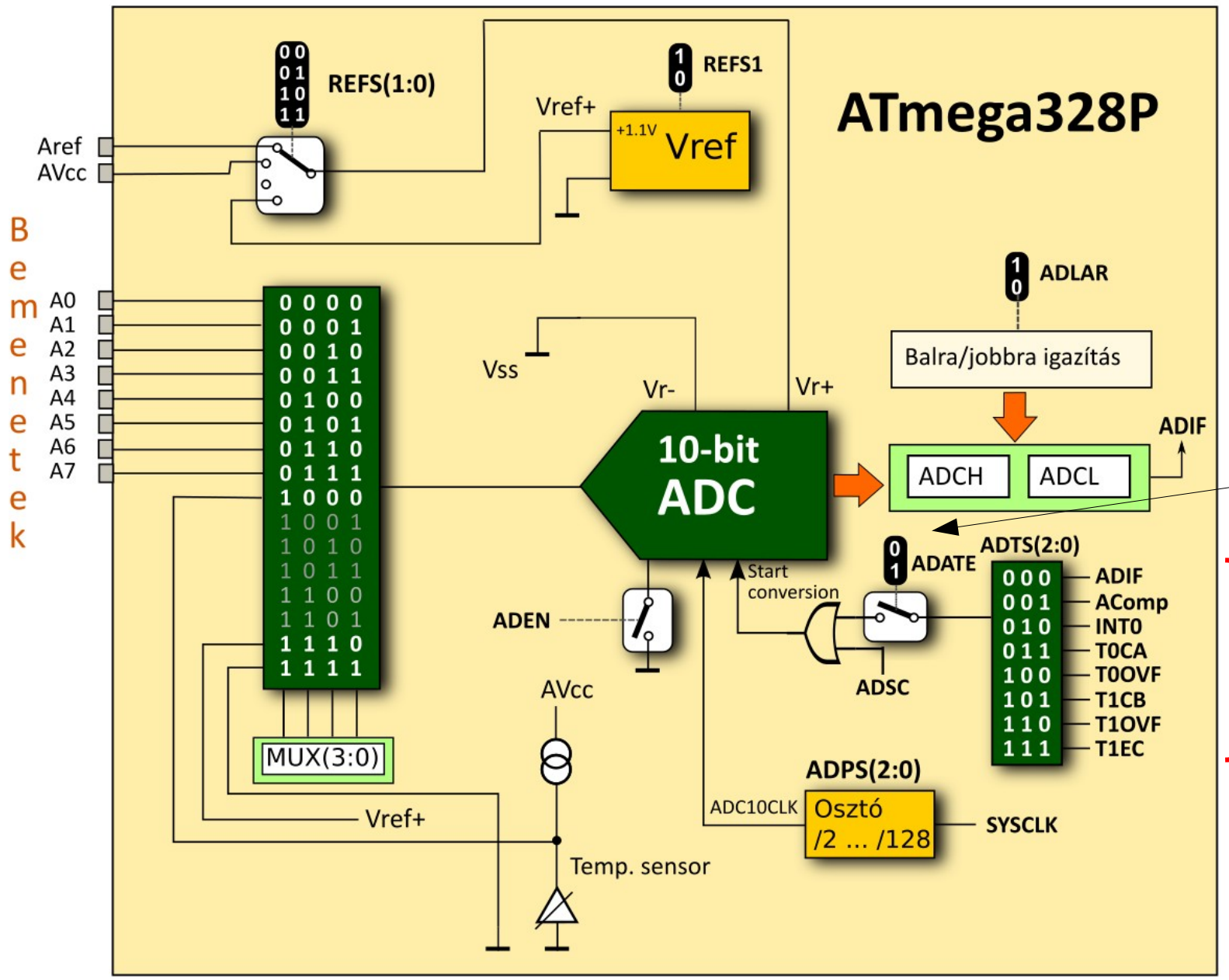
Az ADC regiszterei

DIDR0 – Digital Input Disable Register 0

Bit	7	6	5	4	3	2	1	0	
(0x7E)	–	–	ADC5D	ADC4D	ADC3D	ADC2D	ADC1D	ADC0D	DIDR0
Read/Write	R	R	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Ha 1-et írunk valamelyik bitbe, az letiltja a hozzá tartozó **A0 – A5** kivezetés digitális bemeneti bufferét. **A6** és **A7** csak analóg bemenet, ezeknél nincs mit letiltani…

Folyamatos konverzió: ADIF trigger

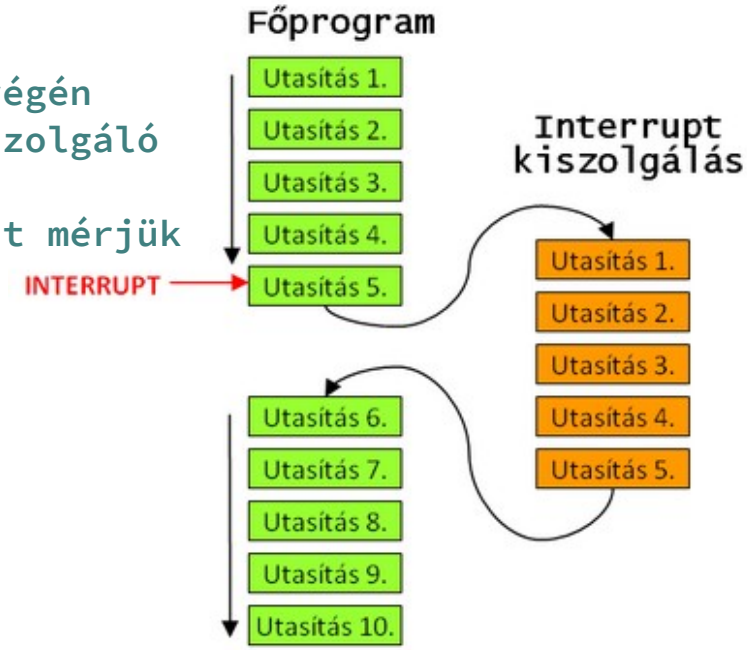


Triggerelés engedélyezése
`ADCSRA.ADATE=1`

Trigger jelforrások

adc_int.ino 2/1

```
/* adc_int

   Folyamatos módban mérünk az ADC-vel, a konverziók végén
   programmegszakítást kérünk, s az ADC megszakításkiszolgáló
   eljárásban olvassuk ki az eredményt.
   Az ADC-vel itt az A0 bemenetre kapcsolt feszültséget mérjük
   A nyers eredményeket a soros porton kiíratjuk.
*/

volatile int value = 0;     // analog value
volatile int eoc = 0;       // End of conversion jelző

void setup() {
  Serial.begin(115200);
  //--- ADC indítás megszakításos módban ---
  DIDR0 = 0x3F;             // A0 - A5 digitális bemeneti mód letiltása
  ADMUX = _BV(REFS0);       // Mérés az A0 csatornában, 5V referenciával
  ADCSRA = _BV(ADEN) |      // ADC enable (bekapcsolás)
           _BV(ADATE)|      // auto trigger enable (triggerelés engedélyezés)
           _BV(ADIE) |      // interrupt enable (megszakítás engedélyezés)
           _BV(ADPS2);      // prescaler = 16  (azaz 2^4)
  ADCSRB = 0x00;            // AD channels MUX off, free running mode (ADIF trigegrel)
  ADCSRA |= _BV(ADSC);      // Konverzió indítása
  sei();                    // megszakítás globális engedélyezése
}
```



Folytatás a következő oldalon ...

adc_int.ino 2/2

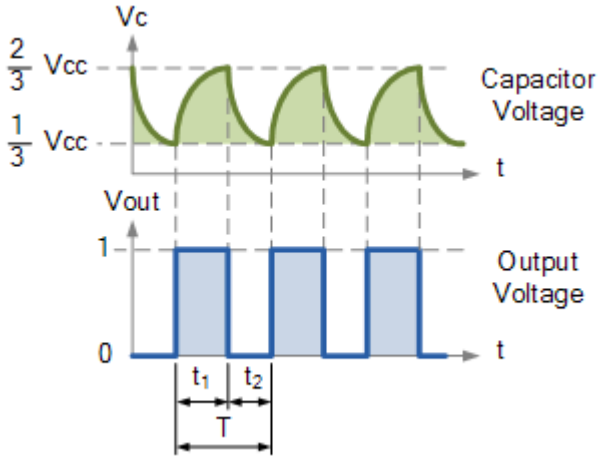
```
void loop() {
  if (eoc) {
    eoc = 0;
    Serial.println(value);
  }
}

//--- 'ADC kész' megszakítás kiszolgálása ---
ISR(ADC_vect) {
  value = ADCL;              // eredmény alsó bájt
  value += ADCH << 8;        // eredmény felső bitek
  eoc = 1;
}
```

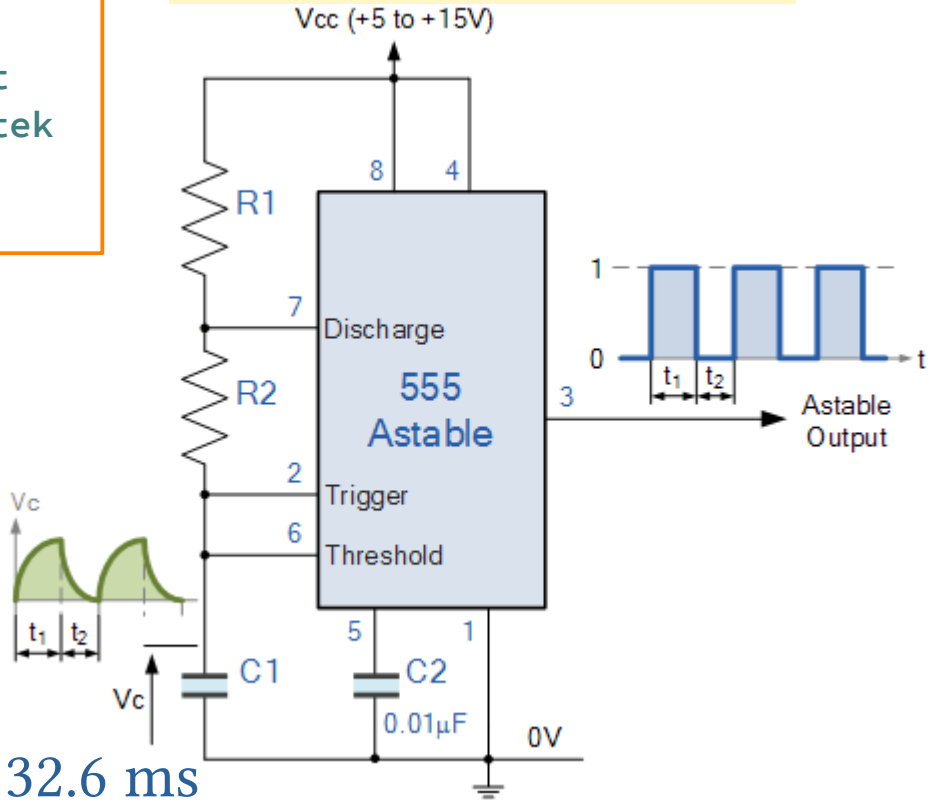
Jelforrásként építsünk egy NE555 astabil multivibrátort!

$$t_1 = 0.693(R_1 + R_2).C$$

and

$$t_2 = 0.693 \times R_2 \times C$$



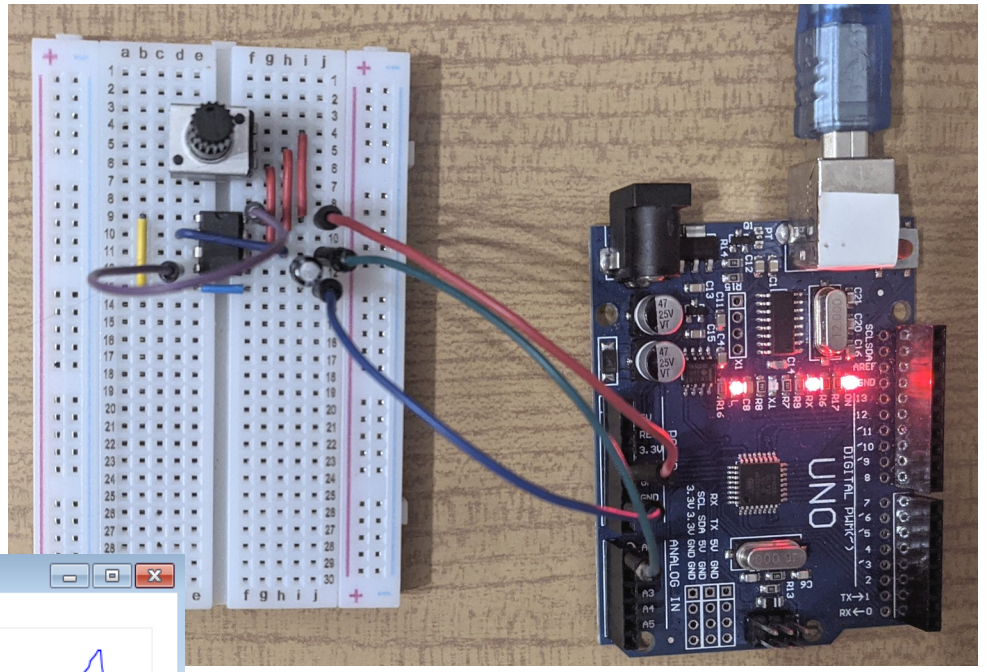


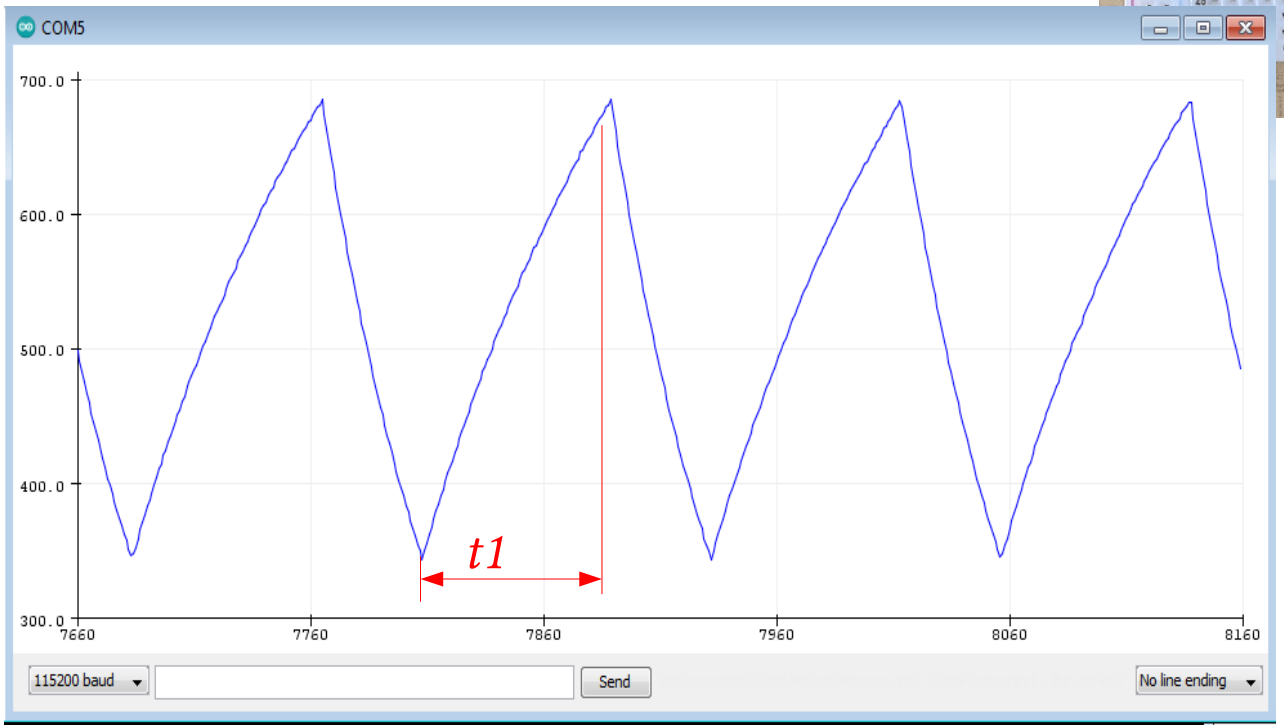
C1 = 4.7 µF, R1 + R1 = 10 kΩ esetén t1 ≈ 32.6 ms

(Forrás: https://www.electronics-tutorials.ws/waveforms/555_oscillator.html)

adc_int.ino futási eredmény

- Elfogadva *t1* ≈ 32.6 ms értékét, s elosztva az ábráról leolvasott adatpontok számával (kb. 80), a pontok közötti mérési idő kb. 407 µs, ilyen gyakorisággal küldünk adatot





Az ábrán az NE555 astabil multivibrátor **C1** kondenzátorán mérjük a feszültséget

A jelalakot az Arduino IDE **Serial plotter** ablakában látjuk

Milyen gyors az ADC?

- Az **ADC** konverzió 13 órajelciklus, kivéve a bekapcsolás (**ADEN**) utáni első konverziót, amely 25 órajelciklus
- A rendszer órajelét 16 MHz-nek véve az **ADC** órajele az előszámláló beállításától függően így alakul:

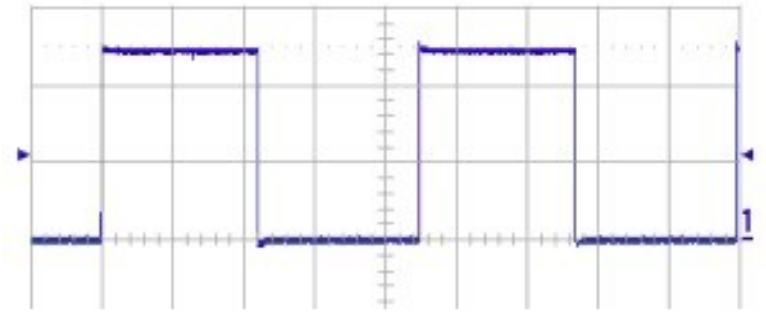
Prescaler	Frekvencia	Periódusidő	Konverziós idő
/2	8 MHz	0.125 µs	1.625 µs
/4	4 MHz	0.25 µs	3.25 µs
/8	2 MHz	0.5 µs	6.5 µs
/16	1 MHz	1 µs	13 µs
/32	500 kHz	2 µs	26 µs
/64	250 kHz	4 µs	52 µs
/128	125 kHz	8 µs	104 µs

- Adatlap szerint a 10 bites pontosság csak akkor biztosítható, ha az **ADC** órajele az **50 kHz – 200 kHz** tartományba esik (itt csak a /128 előosztás felel meg). Nagyobb frekvenciáknál nagyobb lesz a pontatlanság

Ellenőrző mérések

- A Meettechniek: **Arduino Analog measurements** honlapon az ADC konverziós idejét vizsgálták (egy I/O lábat kapcsolgatva konverzió előtt és után)

```
int marker = 12;    // marker output pin
int analogPin = 3; // analog input pin
int aval = 0;       // analog value

void setup() {
  pinMode(marker, OUTPUT);  // pin = output
}

void loop() {
  bitSet(PORTB, 4);               // marker high
  aval = analogRead(analogPin); // sample signal
  bitClear(PORTB, 4);             // marker low
  aval = analogRead(analogPin); // sample signal
}
```



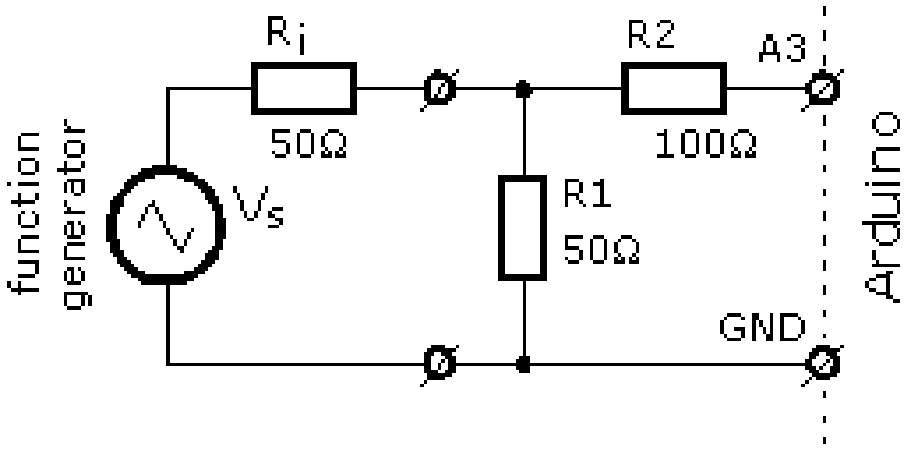
Az analogRead függvény végrehajtási ideje /128 előosztás esetén (skála 50 µs/div)

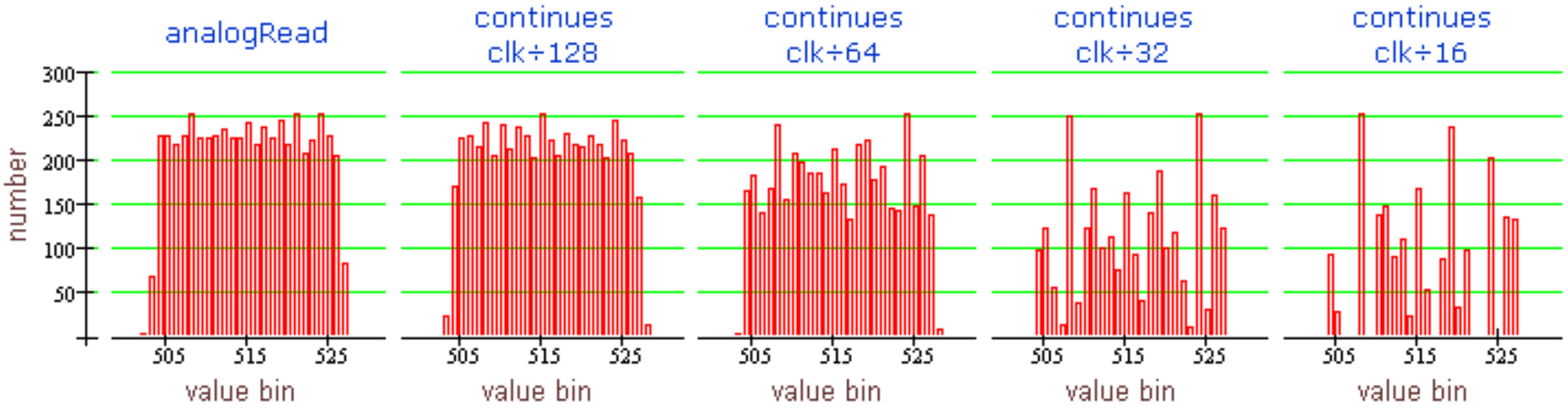
ADC konverzió végrehajtási ideje /16 előosztással (skála 5 µs/div)

- A második ábra folyamatos konverzió és /16 előosztó esetére vonatkozik

Ellenőrző mérések

- A Meettechniek: **Arduino Analog measurements** honlapon vizsgálták az ADC konverziós idejének a pontosságra gyakorolt hatását is
- Az alábbi ábrán a konverzióból kapott értékek eloszlása látható (egy háromszögjel generátor jelét mérték)



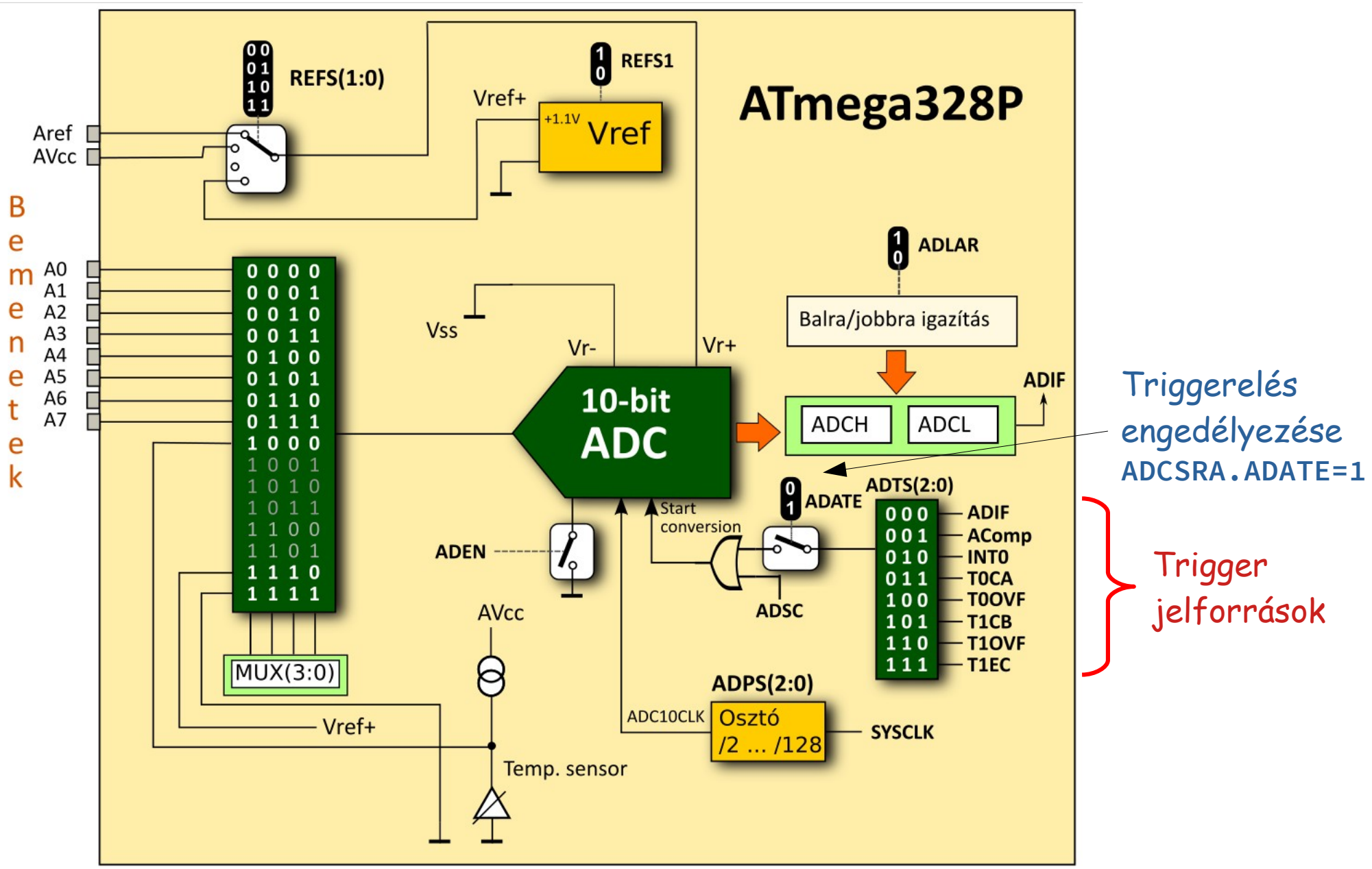


Ellenőrző mérések

- Nick Gammon az **ADC conversion on the Arduino** c. honlapján szintén vizsgálta a konverziós idő pontosságra gyakorolt hatását
- Az alábbi táblázatban 5 V, 3.3 V, 2.5 V és 0 V feszültség mérésének eredménye látható (mindegyik adat 1000 mérés átlaga)
- A tapasztalatok alapján az 1 MHz-es órajel (itt a /16-os osztó) a legnagyobb frekvencia, ahol még használható eredmény várható

Ube	Névl	/128	/64	/32	/16	/8	/4	/2
5 V	1023	1022	1022	1022	1022	842	673	1023
3.3 V	674	672	672	672	672	677	718	1023
2.5 V	511	508	508	508	509	509	512	1023
0 V	0	0	0	0	0	34	193	1022

Az ADC triggerelése a Timer1 időzítővel



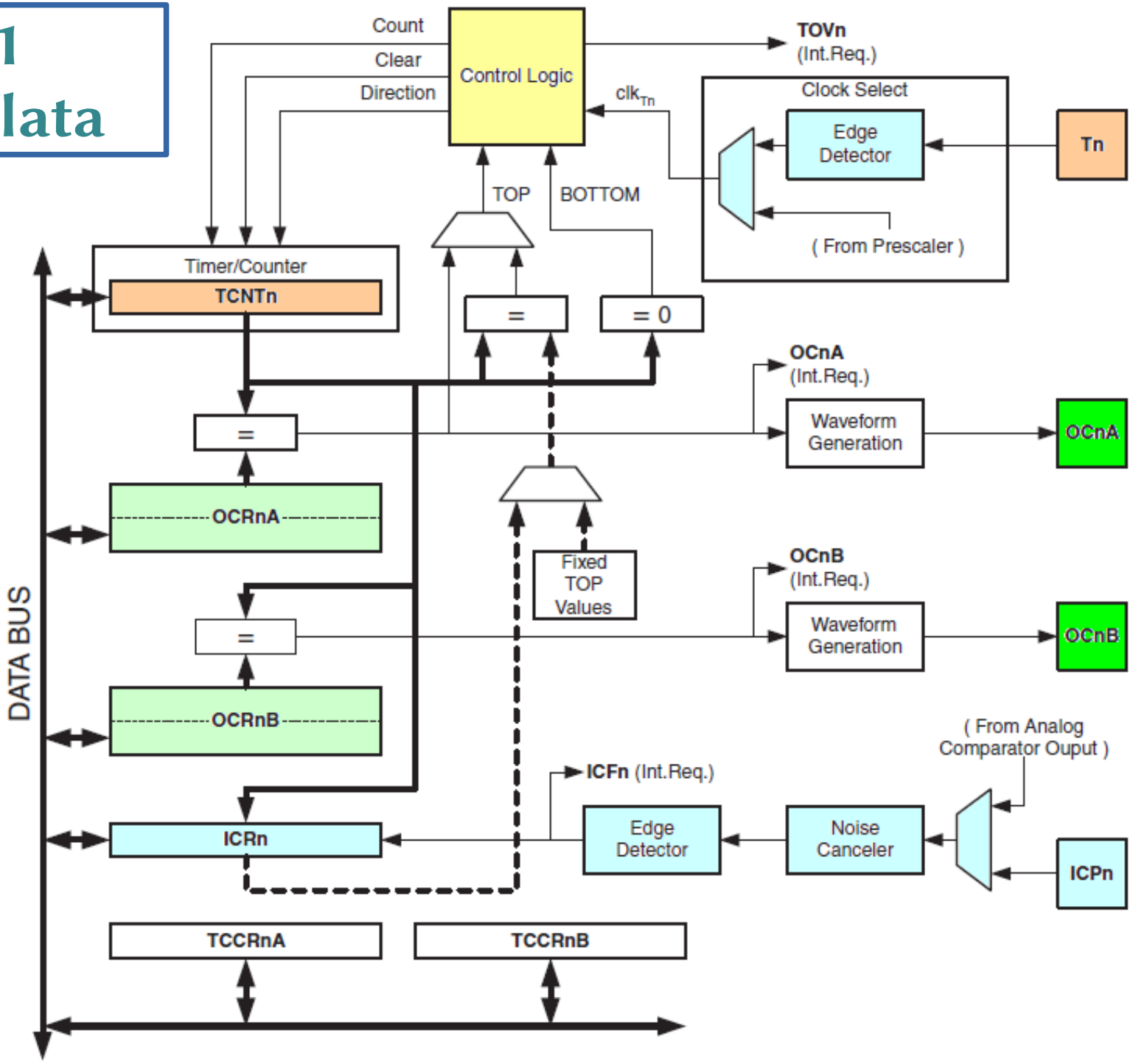
Arduino időzítők/számlálók

- Az ATmega328 mikrovezérlő három időzítője közül **Timer0** és **Timer1** használható az **ADC** triggerelésére
- 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, vagy annak leosztott jele (/8, /64, /246, /1024 leosztás választható), 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()

Timer1 blokkvázlata

Timer0 és Timer2 felépítése is hasonló, de van néhány eltérés:
- 8 bitesek
- nincs bemeneti jelfogás (ICRn)
- Timer2 esetén nincs független Tn bemenet



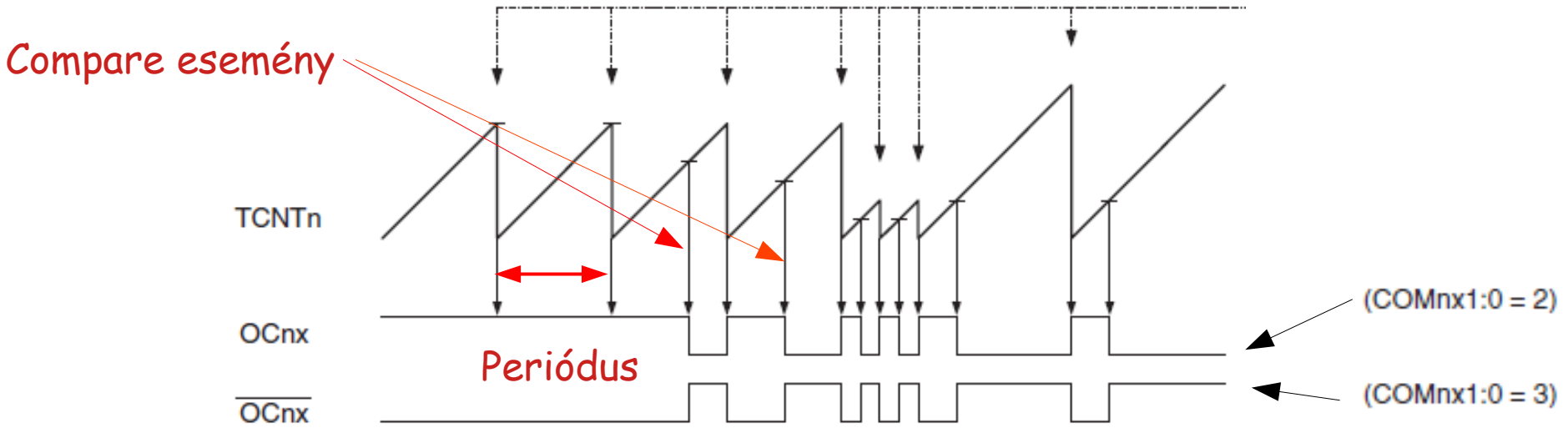
Timer1 regiszterek

- **TCCR1A** – Timer/Counter1 vezérlő regiszter *A*

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	

Table 16-1. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).



Timer1 regiszterek

- **TCCR1B** – Timer/Counter1 vezérlő regiszter ***B***

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	

- **ICNC1** – Input capture zajelnyomás engedélyezés (**0**: ki, **1**: be)
- **ICES1** – Input capture aktív él kiválasztás (**0**: lefutó, **1**: felfutó)

CS12	CS11	CS10	Description	
0	0	0	No clock source (Timer/Counter stopped).	
0	0	1	$clk_{I/O}$/1 (No prescaling)	16 MHz
0	1	0	$clk_{I/O}$/8 (From prescaler)	2 MHz
0	1	1	$clk_{I/O}$/64 (From prescaler)	250 kHz
1	0	0	$clk_{I/O}$/256 (From prescaler)	62.5 kHz
1	0	1	$clk_{I/O}$/1024 (From prescaler)	15 625 Hz
1	1	0	External clock source on T1 pin. Clock on falling edge.	
1	1	1	External clock source on T1 pin. Clock on rising edge.	

Timer1 regiszterek

- Hullámforma generátor üzemmód bitek: megszabják a számlálási sorrendet, a maximum értéket és a generált hullámformát

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Timer1 regiszterek

TCCR1C – Timer/Counter1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	–	–	–	–	–	–	TCCR1C
Read/Write	R/W	R/W	R	R	R	R	R	R	

- **FOC1A, FOC1B** – force output compare A, B (csak nem PWM mód esetén hatásos, s a bit 1-be állítása azonnali egyezési eseményt jelez a hullámforma generátornak

TCNT1H and TCNT1L – Timer/Counter1

Ez a számláló regiszter

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

ICR1H and ICR1L – Input Capture Register 1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

Timer1 regiszterek

OCR1AH and OCR1AL – Output Compare Register 1 A

Ez szabja majd meg a periódust

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

OCR1BH and OCR1BL – Output Compare Register 1 B

Ez szabja meg, mikokor induljon a konverzió (OCR1B ≤ OCR1A)

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	

TIMSK1 – Timer/Counter1 Interrupt Mask Register

Megszakítások engedélyezése

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	

ADC triggerelés Timer1 segítségével

- Az **adc_timer1.ino** programban az **ADC** konverziókat **Timer1** segítségével indítjuk (triggereljük)
- **Timer1** itt **CTC** módban (Clear on Terminal Count) működik [`WGM13:10 = 0b0100`], s az **OCR1A** regiszter szabja meg a számlálás felső határát
- A triggerelés az **TCNT1** = **OCR1B** egyezés eseményekor történik, ehhez az **ADCSRB** regiszterben `ADTS2:0 = 0b101` beállítás tartozik
- **Timer1** előosztója /8 lesz, így 2 MHz a bemenőjel, **OCR1A** értéke 39 lesz, azaz 40 ciklust számol (50 kHz, 20 µs periódus)
- Az **ADC** előosztójánál /16-szoros leosztást kell beállítanunk (1 MHz órajel), ami azt jelenti, hogy 10 bites pontosság nem érhető el (csak 8 bitet használunk)
- A fenti ütemezés csak akkor tartható, ha az adatokat eltároljuk egy tömbváltozóba, és csak az adatgyűjtés végén írjuk ki a soros portra

adc_timer1.ino 2/1

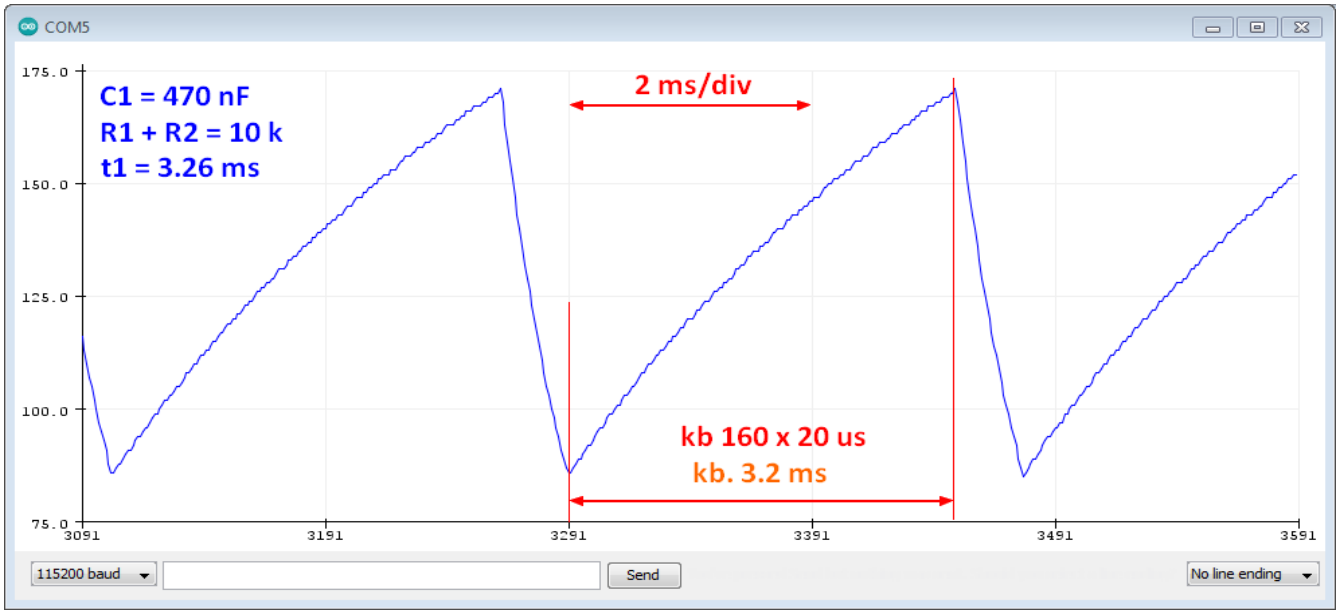
```
/* adc_timer1
 *
 * Az A0 bemenetre kapcsol jel alakjának vizsgálata ADC-vel, tároló módban.
 * Az ADC konverziók indítását Timer1 megszakításokkal végezzük.
 * Timer1 periódusát az OCR1A regiszter és az előosztó szabja meg.
 * Az előosztással 2 MHz-es órajelet küldünk a számláló bemenetére.
 * A programban szereplő 39 iígy 40 ciklus, azaz 20 us időzítést jelent.
 * Az ADC órajelét 1 MHz-re vesszük, így nem használható ki a 10 bites felbontás.
 *
 * A program eredetijének forrása: https://www.gammon.com.au/adc
 */

const int MAX_RESULTS = 512;
volatile byte results[MAX_RESULTS];
volatile int resultNumber = 0;

//--- ADC complete ISR
ISR (ADC_vect) {
  if (resultNumber >= MAX_RESULTS)
    ADCSRA = 0;  // turn off ADC
  else
    results [resultNumber++] = ADC >> 2;
}  // end of ADC_vect

EMPTY_INTERRUPT (TIMER1_COMPB_vect);
```

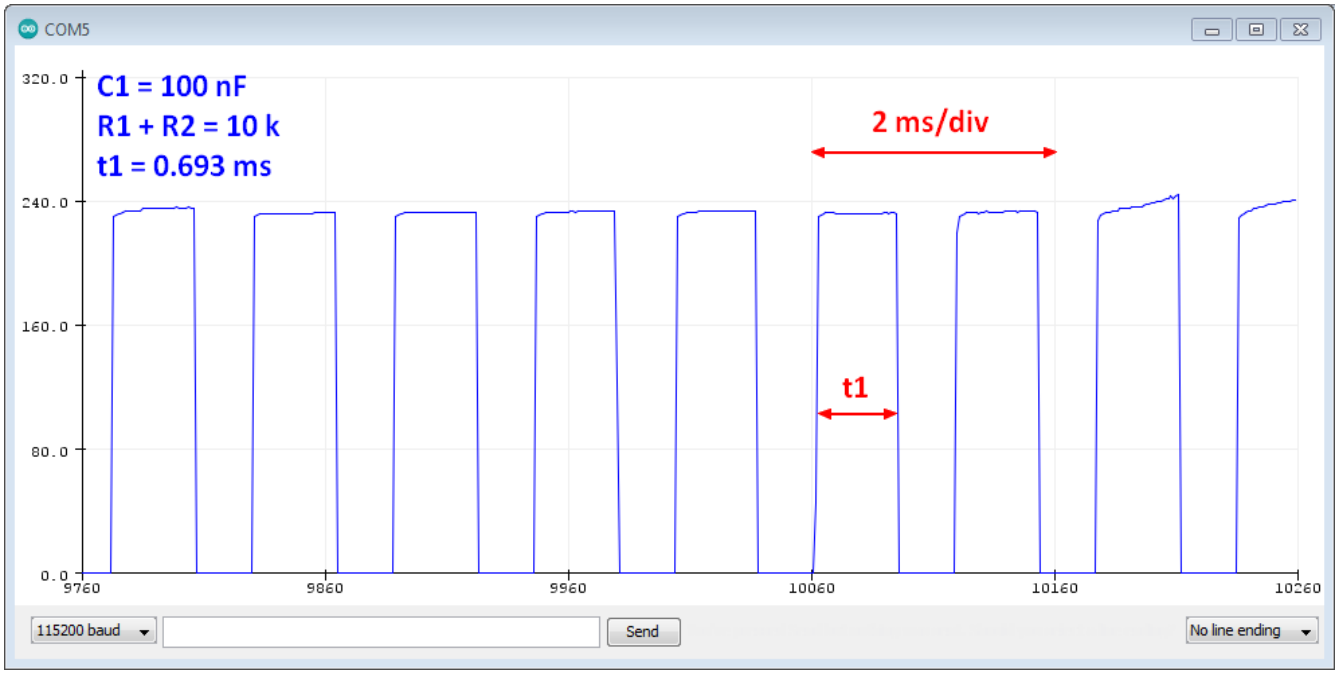
Folytatás a következő oldalon ...

adc_timer1.ino 2/2

```
void setup () {
  Serial.begin (115200);
  Serial.println ();
  // reset Timer 1
  TCCR1A = 0;
  TCCR1B = 0;
  TCNT1 = 0;
  TCCR1B = bit(CS11) | bit(WGM12);    // CTC, prescaler of 8
  TIMSK1 = bit(OCIE1B);               // Enable Timer1 OC1B interrupt
  OCR1A = 39;                         // 20 uS - sampling frequency 50 kHz
  OCR1B = 39;
//--- ADC configuration -------------
  ADCSRA =  bit(ADEN) | bit(ADIE) | bit(ADIF);   //ADC on, interrupt on completion
  ADCSRA |= bit(ADPS2);               // Prescaler of 16
  ADMUX = bit(REFS0) | (adcPin & 7);
  ADCSRB = bit(ADTS0) | bit(ADTS2);   // Timer/Counter1 Compare Match B
  ADCSRA |= bit(ADATE);               // Turn on automatic triggering

  // wait for buffer to fill
  while (resultNumber < MAX_RESULTS);
  for (int i = 0; i < MAX_RESULTS; i++)
    Serial.println (results [i]);
}  // end of setup

void loop () { }
```

adc_timer1.ino futási eredmények





Az Arduino nano kártya kivezetései