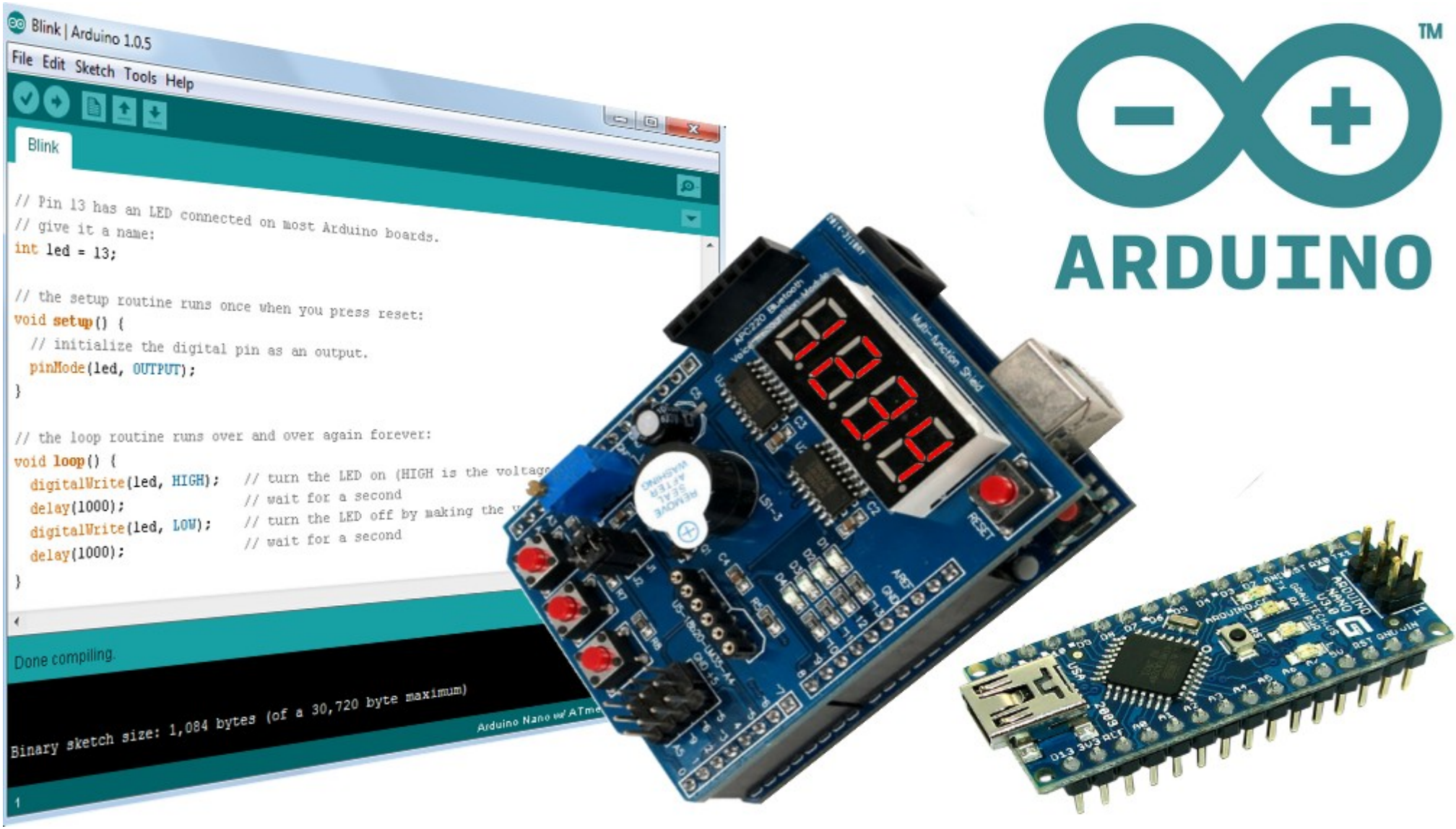
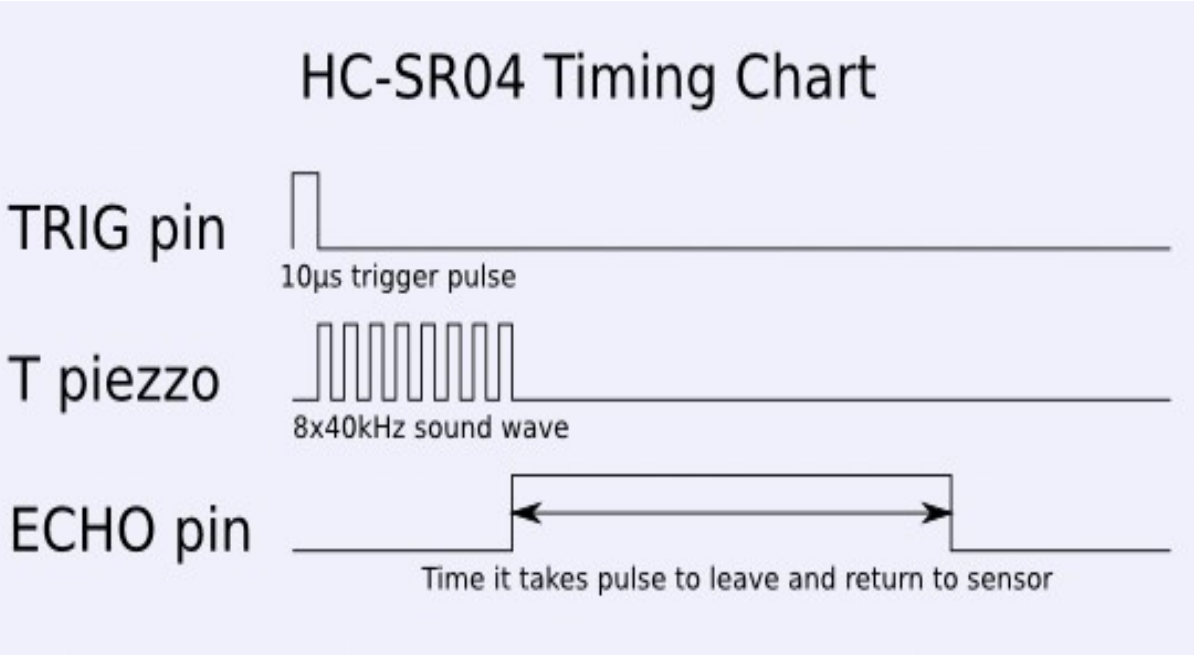
Arduino tanfolyam kezdőknek és haladóknak



7. Ultrahangos távolságmérés

Ultrahangos távolságmérés

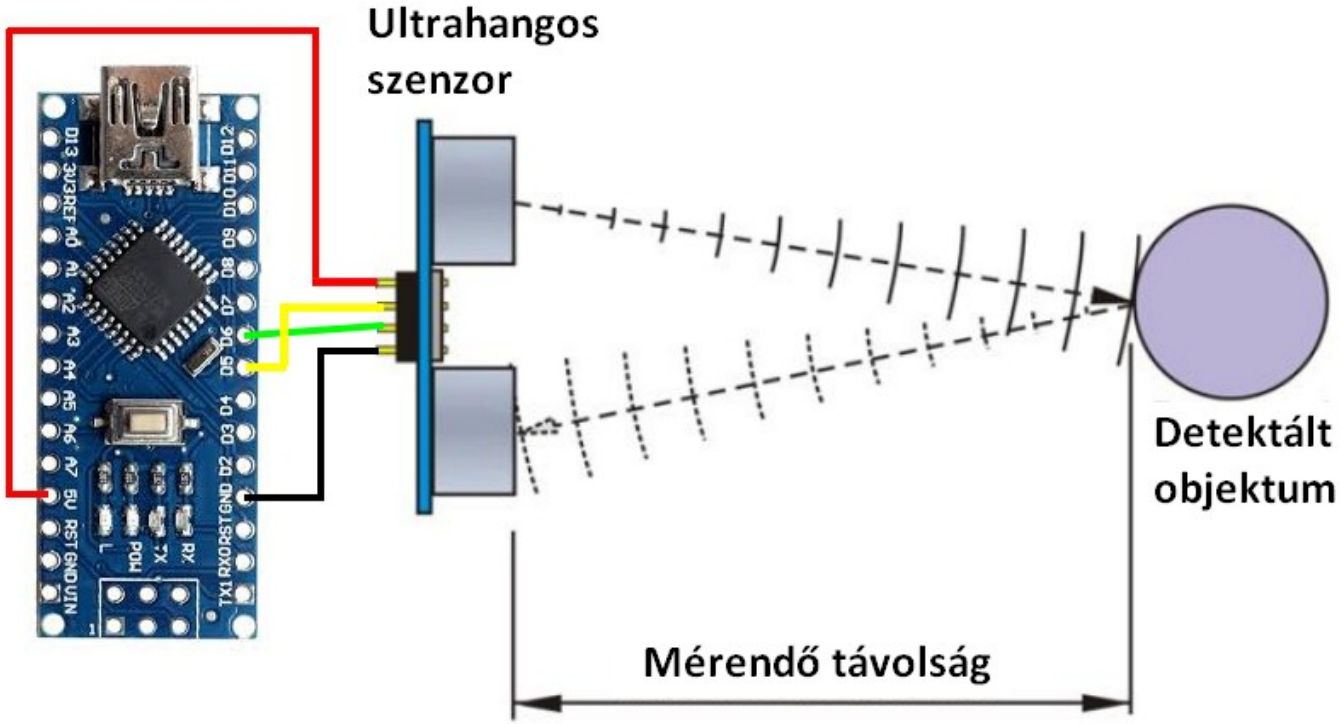
- A **HC-SR04** modul piezo jeladója az indító impulzus hatására egy 40 kHz-es jelcsomagot sugároz ki.
- A modul digitális kimenő impulzusának szélessége megegyezik a visszaverődött hang terjedési idejével.



Főbb paraméterek

- Tápfeszültség: 4.5 V – 5.5 V
- Mérési tartomány: 2 cm – 4 m (a gyakorlatban inkább 2 m)
- Érzékelési szögtartomány: ~ 16 °

Az ultrahangos távolságmérés elve

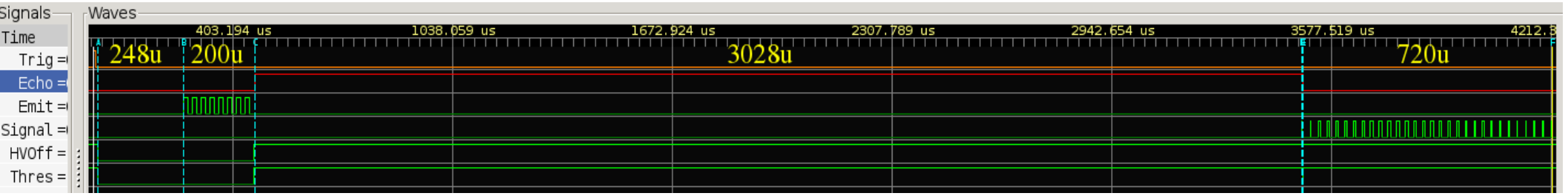


- Ha az ultrahang impulzus útjában egy tárgy található, akkor a hullámok egy része visszaverődik. Ez a visszhangot a vevő észlelheti
- Az adás és a vétel között eltelt t időből és a hang v terjedési sebességéből kiszámítható a szenzor és a tárgy között d távolság:

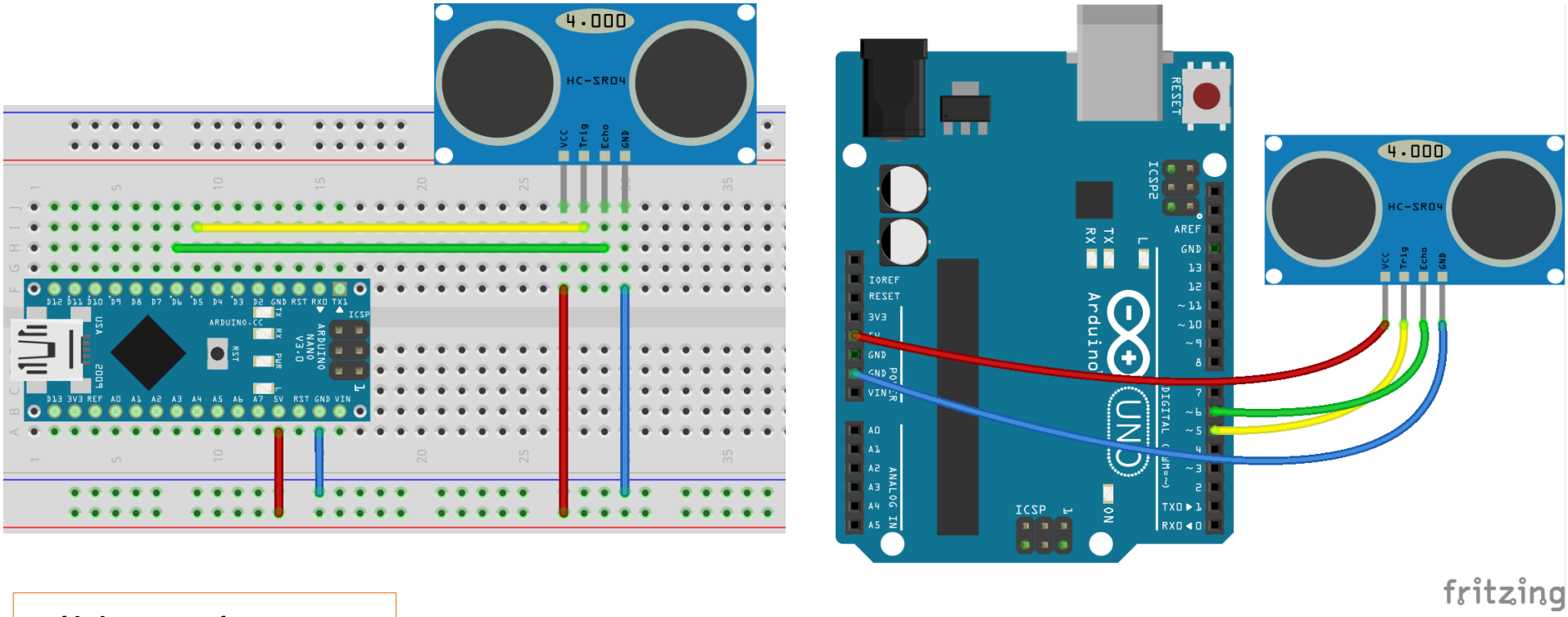
$$d = t \cdot v / 2$$ ahol v esetünkben kb. 340 m/s

A modul működése

- Az áramkör két piezoelektromos adót/vevőt tartalmaz, amelyek közül az adót egy MAX232 IC felhasználásával 20 V-tal hajtunk meg. Az IC a Q2 tranzisztoron keresztül adáskor kap tápfeszültséget, hogy a vevőt ne zavarja fölösleges zajjal
- A vevő jelét egy LM324 IC erősíti és szűri, amely 4 db műveleti erősítőt tartalmaz. A negyedik fokozat a Q1 tranzisztorral egy hiszterézises komparátort alkot
- Az eszközt működését egy kínai EM78P153S mikrovezérlő irányítja
- Bővebb leírás és javaslat a modul feljavítására: uglyduck.vajn.icu/ep/archive/2014/01/Making_a_better_HC_SR04_Echo_Locator.html
- Az alábbi ábrán egy tipikus visszhangjel látható:



Ultrahangos távolságmérő



Lábkiosztás
Trigger: D5
Echo: D6

unsigned long **pulseInt**(int *pin,* int *value*)

Meghatározza a beérkező impulzus szélességét (mikroszekundum egységekben).
pin – a vizsgált bemenet sorszáma
value – az vizsgálandó impulzus polaritása (HIGH vagy LOW)

Sonar.ino 2/1. oldal

- Az ultrahangos távolságmérővel másodpercenként mérünk
- Az eredményt a soros porton kiíratjuk

```
#define trigPin    5                // Trigger kimenet
#define echoPin    6                // Echo bemenet

int maximumRange = 200;             // Legnagyobb távolság cm-ben
int minimumRange = 1;               // Minimális távolság cm-ben
long duration;                      // Időtartam [us]
float distance;                     // Távolság [cm]

void setup() {
  Serial.begin(9600);               // Soros kapcsolat 9600 bit/s
  Serial.println("Sonar program");
  pinMode(echoPin, INPUT);          // Impulzus bemenet
  pinMode(trigPin, OUTPUT);         // Vezérlő kimenet
  digitalWrite(trigPin, LOW);       // Alaphelyzetben alacsony szint
  pinMode(LED_BUILTIN,OUTPUT);      // A beépített LED jelző funkciót lát el
  digitalWrite(LED_BUILTIN,HIGH);
}
```

Sonar.ino 2/2. oldal

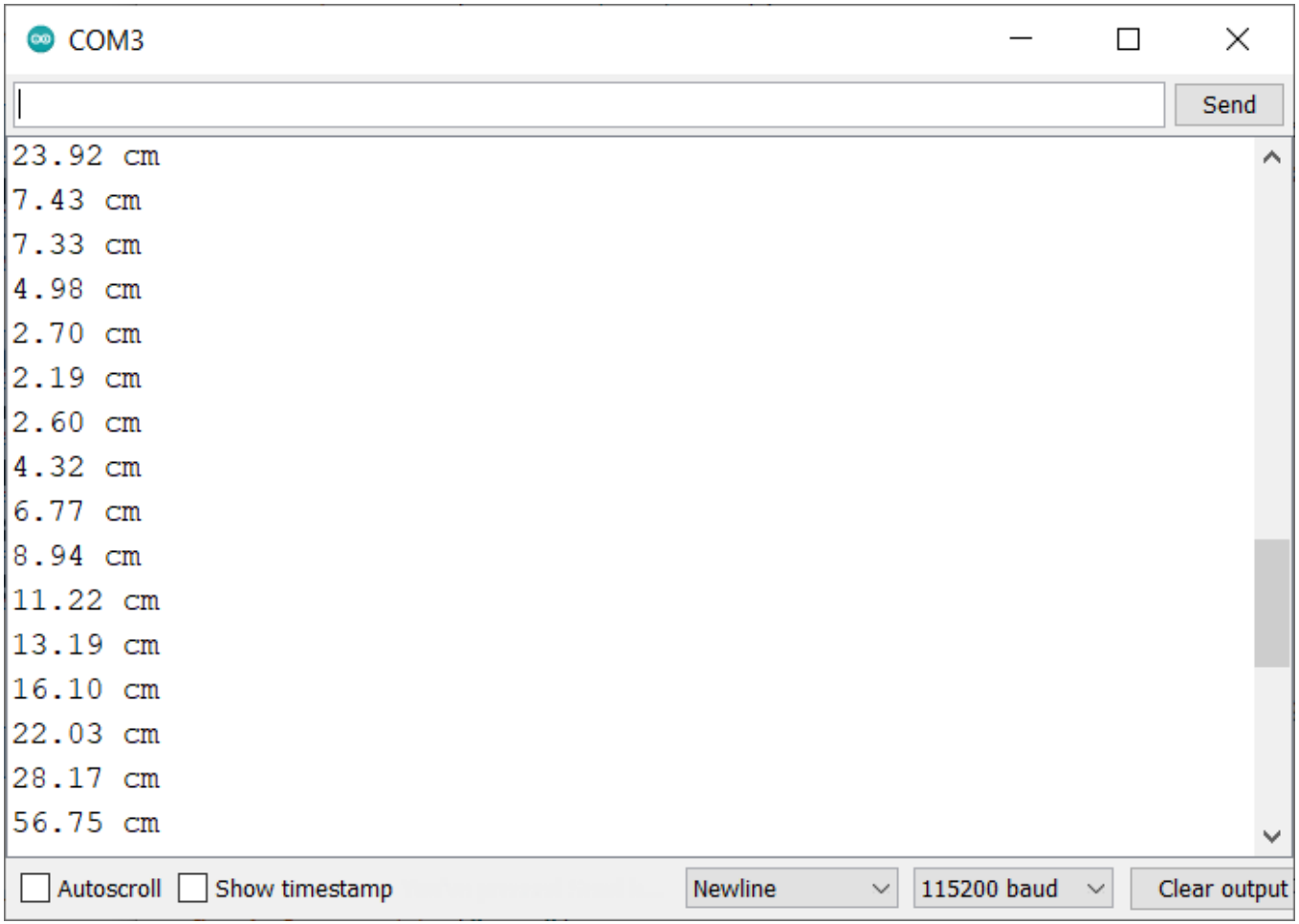
```
void loop() {
  delay(1000);                          //egy kis várakozás
  digitalWrite(LED_BUILTIN,HIGH);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH); //Impulzus szélességének meghatározása
  digitalWrite(LED_BUILTIN,LOW);
  distance = duration/58.82;            // Kiszámoljuk a távolságot
  if (distance >= maximumRange || distance < minimumRange) {
    distance = -1.0;
  }
  Serial.print(distance);               // A mért távolság kiíratása
  Serial.println(" cm");
}
```

$$d = \frac{t \cdot v}{2}$$

Ahol d a távolság, t az impulzus hossza, v a hang terjedési sebessége (~344 m/s). Mivel t értéke µs-ban adott, d-t pedig cm-ben mérjük, így d = t * 34 400/2 000 000, azaz **d = t/58.2**

Sonar.ino: futási eredmény

- A program futási eredménye az alábbi ábrán látható
- Az érzékelő előtti tárgyat előbb közelítettük, majd távolítottuk



Az Arduino NewPing programkönyvtár

- Tim Eckel **NewPing** programkönyvtára együttműködik a legtöbb ultrahangos távolságmérővel (HC-SR04, SRF05, SRF06 stb)
  - Nem vár egy másodpercig, ha nincs visszhangjel
  - Konzisztensen és megbízhatóan akár 30 mérés/s
  - Timer megszakítással is használható eseményvezérelt rendszerekben
  - Az I/O kezelése közvetlen regiszterelléréssel (gyorsan) történik
  - Lehetőség van a maximális távolság megadására
  - Beépített digitális szűrés az egyszerű hibajavításhoz
  - Távolságszámítás (*cm* vagy *inch*)
  - Nem használja a ***pulseIn()*** függvényt
  - Több szenzor együttes használata
  - Egyvezetékes használat (Echo és Trigger lábak összekötve) – SFR06 kivételével
- A **`NewPing`** osztály példányosítása:

```
NewPing sonar(trigger_pin, echo_pin [, max_cm_distance])
```

A NewPing objektumosztály metódusai

A **NewPing** objektumosztály az alábbi nyilvános hozzáférésű metódusokkal rendelkezik:

- **sonar.ping**([max_cm_distance]) – ping és a visszhang érkezési idejének mérése [µs]
- **sonar.ping_in**([max_cm_distance]) – ping és a távolság mérése hüvelykben (25.4 mm)
- **sonar.ping_cm**([max_cm_distance]) – ping and és a távolság mérése centiméterben
- **sonar.ping_median**(iterations [, max_cm_distance]) – több mérés (alapért. 5), a megadott határon túli értékek eldobása és eredményként a medián megadása
- **sonar.convert_in**(echoTime) – a visszhangidő konvertálása távolságra [in]
- **sonar.convert_cm**(echoTime) – a visszhangidő konvertálása távolságra [cm]
- **sonar.ping_timer(**function [, max_cm_distance]) – ping és a megadott függvény visszahívása a végén. Az eredmény kiolvasása és konvertálása előtt ellenőrizni kell, hogy mérés eredménye érvényes-e
- **sonar.check_timer**() – ellenőrzi, hogy a visszhang a megadott határon belül érkezett-e
- **NewPing::timer_us**(frequency, function) – függvény visszahívása adott időközönként
- **NewPing::timer_ms**(frequency, function) – függvény visszahívása adott időközönként
- **NewPing::timer_stop()** – leállítja az időzítéseket

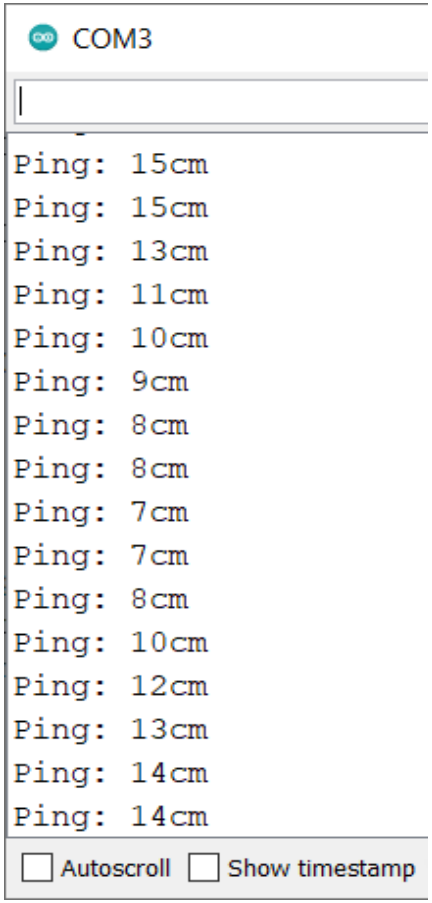
newping_serial.ino

- Egyszerű példa a **NewPing** programkönyvtár használatára
- A kapcsolás ugyanaz, mint az első példaprogramnál

```
#include <NewPing.h>

#define TRIGGER_PIN  5
#define ECHO_PIN     6
#define MAX_DISTANCE 200

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(115200);
}

void loop() {
  delay(100);
  Serial.print("Ping: ");
  Serial.print(sonar.ping_cm());
  Serial.println("cm");
}
```

200 cm lesz a legnagyobb elfogadott érték

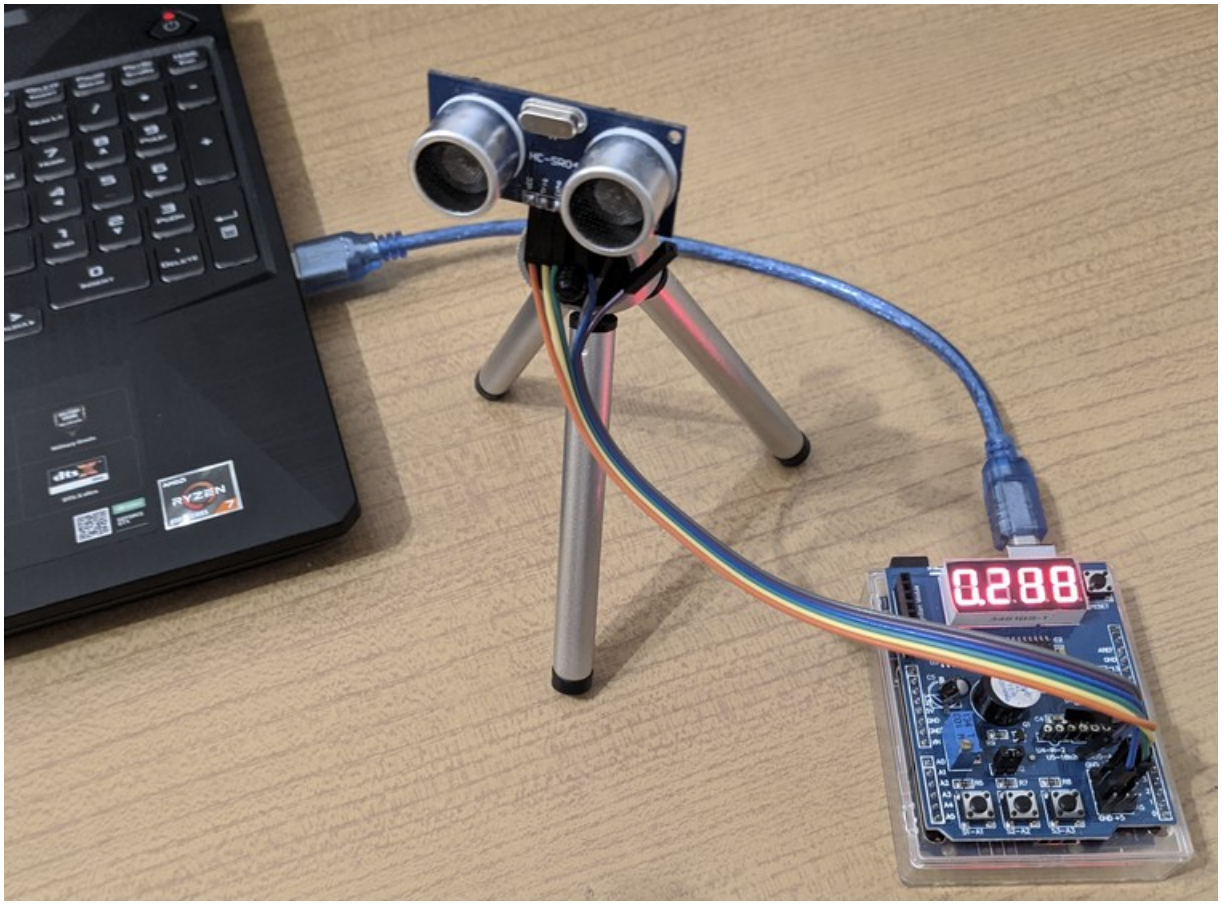
Centiméterekben adja vissza a távolságot



Távolságmérő hétszegmenses kijelzővel

- A Multifunkciós kártya felhasználásával önálló távmérő „készüléket” is kialakíthatunk: az ultrahangos érzékelőt a szabad tüskékre csatlakoztatjuk, s a hétszegmenses kijelzőn íratjuk ki az eredményt – az előző előadásokban bemutatott módon
- Bekötés:

HC-SR04	Arduino
VCC	5V
Trig	D5
Echo	D6
Gnd	GND



newping_7seg.ino 2/1.

A **Multifunkciós kártya** hétszegmenses kijelzője kezelésének részleteit az **arduino20_04** előadásban ismertettük (**thermometer_display** mintaprogram)

```
#include <NewPing.h>

/* HC-SR04 paraméterek */
#define TRIGGER_PIN  5
#define ECHO_PIN     6
#define MAX_DISTANCE 200

/* Multifunkciós kártya kijelző vezérlés */
/* 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};
volatile byte segment_data[] = {0, 0, 0, 0}; // adattároló buffer
byte idx = 0;                                // Kijelzés számlálója
long nexttime;                               // Időbélyeg mérés indításhoz

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // Példányosítás
```

newping_7seg.ino 2/2.

```
void setup() {
  pinMode(LATCH_DIO, OUTPUT);
  pinMode(CLK_DIO, OUTPUT);
  pinMode(DATA_DIO, OUTPUT);
  nexttime = millis();
}

void loop() {
  if ( millis() > nexttime) {
    int distance = sonar.ping_cm();
    segment_data[0] = 0xFF;
    segment_data[1] = SEGMENT_MAP[(distance/100) % 10];
    segment_data[2] = SEGMENT_MAP[(distance /10) % 10];
    segment_data[3] = SEGMENT_MAP[distance % 10];
    nexttime += 250;  // 250 ms-onként indítunk új mérést
  }
  refresh_display();  // Mérési szünetekben pörgetjük a kijelzést
}

void refresh_display() {  // A soron következő számjegy megjelenítése
  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);
}
```

A **Multifunkciós kártya** hétszegmenses kijelzője kezelésének részleteit az **arduino20_04** előadásban ismertettük (**thermometer_display** mintaprogram)

A hangsebesség hőmérsékletfüggése

- A levegőben terjedő hang terjedési sebességének közelítő értékét az alábbi képlet adja meg, ahol ϑ a hőmérséklet ℃ fokokban:

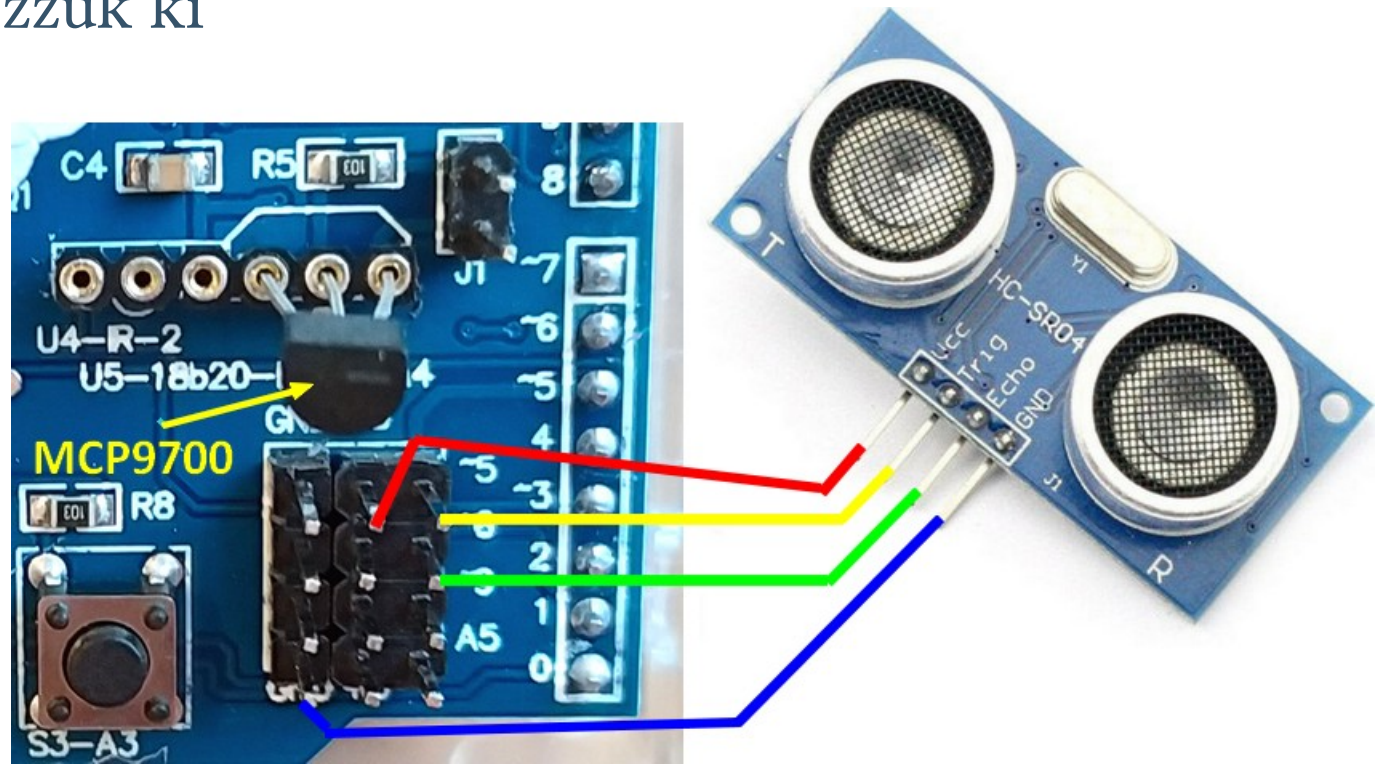
$$v = (331.5 + (0{,}6 \cdot \vartheta))[m/s]$$ (forrás: Wikipédia)

- A hang terjedési sebességét a hőmérséklet jelentős mértékben, a légnedvesség kevésbé, a légnyomás pedig nem befolyásolja
- Néhány hőmérsékletre kiszámoltuk a hang terjedési sebességét és annak reciprokát

T [°C]	v [m/s]	v [cm/µs]	1/v [µs/cm]	1/2v [µs/cm]
0	331,5	0,03315	30,2	60,4
5	334,5	0,03345	29,9	59,8
10	337,5	0,03375	29,6	59,2
15	340,5	0,03405	29,4	58,8
20	343,5	0,03435	29,1	58,2
25	346,5	0,03465	28,9	57,8
30	349,5	0,03495	28,6	57,2

Távolságmérés hőmérsékletre korrigálva

- Mérjük a hőmérsékletet egy analóg hőmérővel (pl. **MCP9700**) az Arduino **A4** analóg bemenetén és a mért értékkel korrigáljuk az ultrahangos szenzorral mért távolságot!
- A **J1** átkötést ne felejtsük el levenni!
- A feladat egy kicsit bonyolult, ezért több lépésben dolgozzuk ki



homero_int: Eseményvezérelt hőmérés

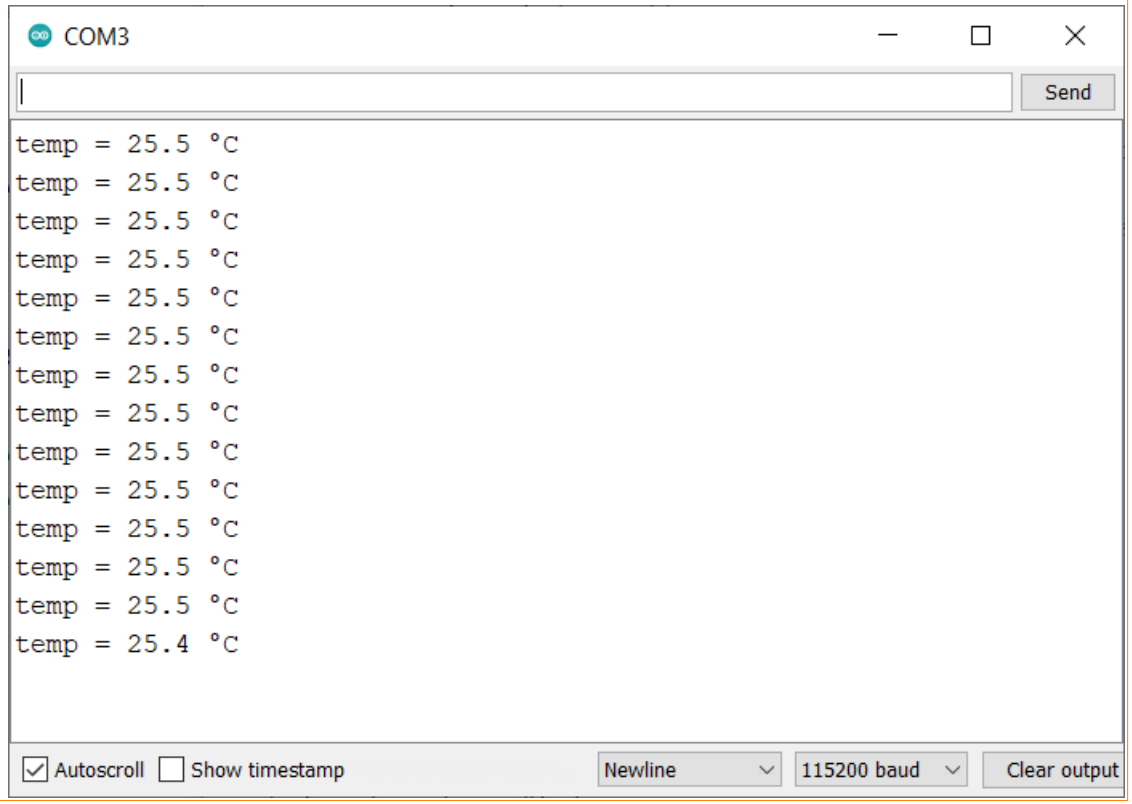
- Az ADC az **A4** bemenetre kapcsolt MCP9700 hőmérő jelét méri az 1.1 V-os belső referenciához viszonyítva. Folyamatos módban mérünk, s az ADC megszakításkor összegezzük, majd 1100 mérésenként eltároljuk az eredményt.

```
volatile long adc_sum = 0; // ADC összegző változó
volatile long adc_val = 0; // az eredmény 1100 mérés összege
uint16_t adc_cnt = 1100;   // ADC számláló

void setup() {
  Serial.begin(115200);
  //--- ADC indítás megszakításos módban ---
  DIDR0 = 0x3F;             // digital inputs disabled
  ADMUX = _BV(REFS0) | _BV(REFS1) | _BV(MUX2); // A4 csatorna, 1.1 V ref
  ADCSRA = _BV(ADEN) |      // ADC enable
           _BV(ADATE) |     // auto trigger enable
           _BV(ADIE) |      // interrupt enable
           _BV(ADPS2);      // prescaler = 16  (azaz 2^4)
  ADCSRB = 0x00;            // AD channels MUX off, free running mode
  ADCSRA |= _BV(ADSC);      // Konverzió indítása
}
```

7	6	5	4	3	2	1	0	
REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
1	1	0	0	0	1	0	0	

homero_int: Eseményvezérelt hőmérés

```
void loop() {
  delay(1000);
  long mv = adc_val / 1024;
  float tempC = (mv - 500) / 10.0;
  Serial.print("temp = ");
  Serial.print(tempC, 1);
  Serial.println(" °C");
}

// ADC megszakítás kiszolgálása
ISR(ADC_vect) {
  adc_sum += ADCW;
  if (--adc_cnt == 0) {
    adc_val = adc_sum;
    adc_sum = 0;
    adc_cnt = 1100;
  }
}
```



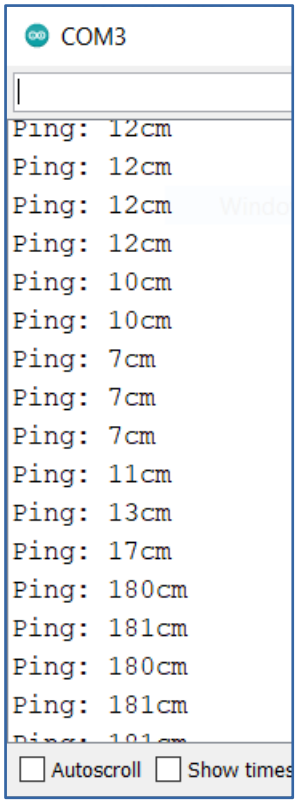
event_timer: Eseményvezérelt távolságmérés

- A távolságmérést csak elindítjuk, s egy visszahívási függvényben kérdezzük le a mérés állapotát, illetve eredményét
- A visszahívási függvényt Timer2 megszakításból hívja meg a program

```
#include <NewPing.h>

/* HC-SR04 paraméterek */
#define TRIGGER_PIN  5
#define ECHO_PIN     6
#define MAX_DISTANCE 200

unsigned int pingSpeed = 500;     // Ilyen gyakorisággal mérünk (500 ms)
unsigned long pingTimer;          // Időbélyeg a következő méréshez
volatile long sonar_val = 0;      // A kiolvasott eredmény

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(115200);
  pingTimer = millis() + 50;      // Ekkor kezdjük a mérést
}
```

event_timer: Eseményvezérelt távolságmérés

```
void loop() {
  if (millis() >= pingTimer) {   // Ha elérkezett az idő
    pingTimer += pingSpeed;      // Következő időpont előjegyzése
    sonar.ping_timer(echoCheck); // Mérés indítása
  }
  if (sonar_val) {
    Serial.print("Ping: ");
    Serial.print(sonar_val / US_ROUNDTRIP_CM);
    Serial.println("cm");
    sonar_val = 0;
  }
}

// Timer2 megszakításból hívjuk
void echoCheck() {
  if (sonar.check_timer()) {      // Volt-e már válasz?
    sonar_val = sonar.ping_result;// Eredmény kiolvasása
  }
}
```



sonar_tempcorrected.ino

- Kombináljuk össze az előző két programot, adjuk hozzá a hétszegmenses kijelzést, és a távolságot korrigáljuk a hőmérsékletre!

```
#include <NewPing.h>
/* HC-SR04 paraméterek */
#define TRIGGER_PIN  5
#define ECHO_PIN     6
#define MAX_DISTANCE 200
/* A kijelzőt vezérlő lábak definiálása */
#define LATCH_DIO 4
#define CLK_DIO   7
#define DATA_DIO  8
const byte SEGMENT_MAP[] = {0xC0,0xF9,0xA4,0xB0,0x99,0x92,0x82,0xF8,0X80,0X90};
const byte SEGMENT_SELECT[] = {0xF1, 0xF2, 0xF4, 0xF8};
volatile byte segment_data[] = {0, 0, 0, 0}; // adattároló buffer
byte idx = 0;                    // Kijelzés számlálója

volatile long adc_sum = 0;       // analog value
volatile long adc_val = 0;       // ADC 1100 mérés összege
uint16_t adc_cnt = 1100;         // ADC számláló

unsigned int pingSpeed = 500;    // Ilyen gyakorisággal mérünk (500 ms)
unsigned long pingTimer;         // Időbélyeg a következő méréshez
volatile long sonar_val = 0;     // A kiolvasott eredmény

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);
```

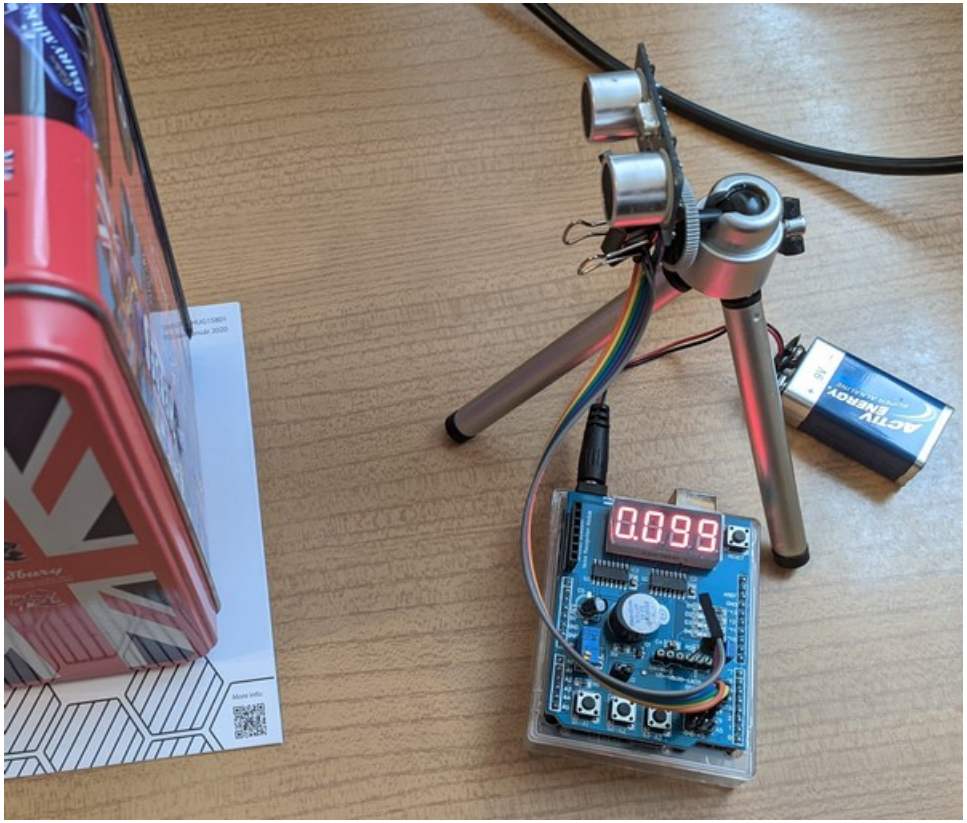
sonar_tempcorrected.ino

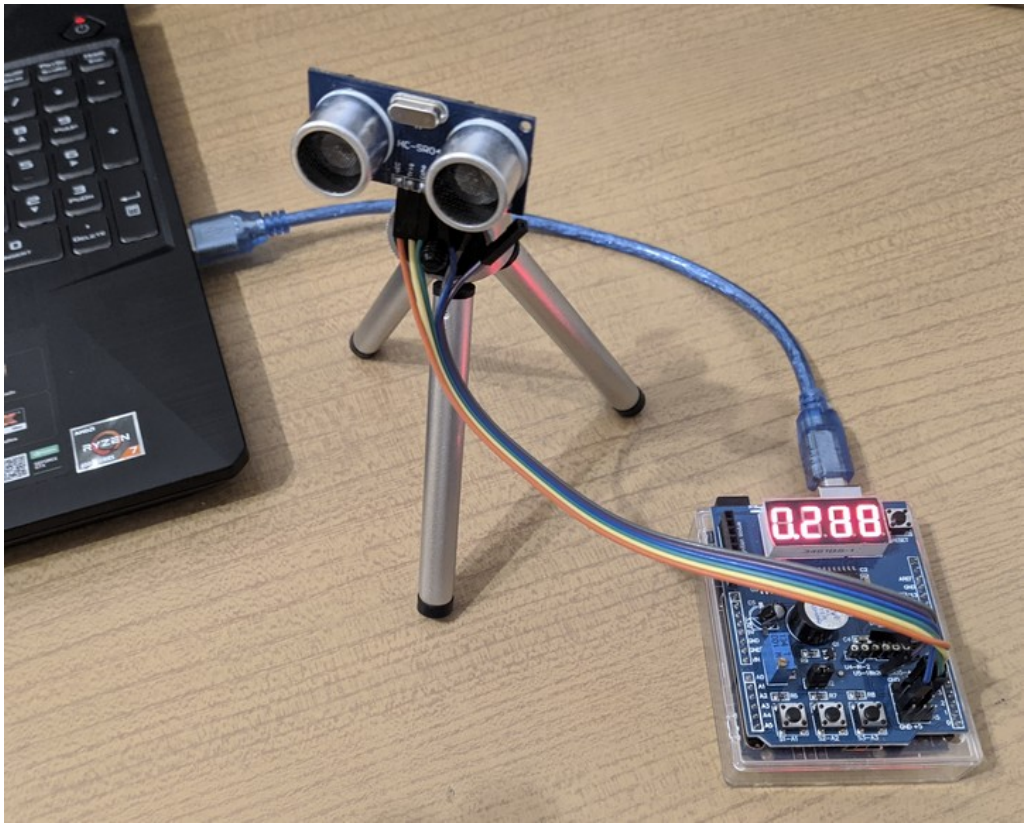
```
void setup() {
  /* Vezérlő lábak konfigurálása */
  pinMode(LATCH_DIO, OUTPUT);
  pinMode(CLK_DIO, OUTPUT);
  pinMode(DATA_DIO, OUTPUT);
  pingTimer = millis() + 50;      // Ekkor kezdjük a mérést
  //--- ADC indítás megszakításos módban ---
  DIDR0 = 0x3F;                  // digital inputs disabled
  ADMUX = _BV(REFS0) | _BV(REFS1) | _BV(MUX2); // A4 csatorna, 1.1 V ref
  ADCSRA = _BV(ADEN) |           // ADC enable
           _BV(ADATE) |          // auto trigger enable
           _BV(ADIE) |           // interrupt enable
           _BV(ADPS2);           // prescaler = 16  (azaz 2^4)
  ADCSRB = 0x00;                 // AD channels MUX off, free running mode
  ADCSRA |= _BV(ADSC);           // Konverzió indítása
  sei();                         // megszakítás globális engedélyezése
}

//--- 'ADC kész' megszakítás kiszolgálása ---
ISR(ADC_vect) {
  adc_sum += ADCW;               // 1100 mérést összegzünk
  if (--adc_cnt == 0) {
    adc_val = adc_sum;
    adc_sum = 0;
    adc_cnt = 1100;
  }
}
```

sonar_tempcorrected.ino

```
void loop() {
  if (millis() >= pingTimer) {   // Ha elérkezett az idő
    pingTimer += pingSpeed;      // Következő időpont előjegyzése
    sonar.ping_timer(echoCheck); // Mérés indítása
  }
  if (sonar_val) {
    long mv = adc_val / 1024;
    float tempC = (mv - 500) / 10.0;
    int distance = sonar_val * (331.5 + 0.6 * tempC) / 2000;
    sonar_val = 0;               // Új adatot várunk bele
    segment_data[0] = SEGMENT_MAP[(distance / 1000) % 10] & 0x7F; // Tizedespont
    segment_data[1] = SEGMENT_MAP[(distance / 100) % 10];
    segment_data[2] = SEGMENT_MAP[(distance  / 10) % 10];
    segment_data[3] = SEGMENT_MAP[distance  % 10];
  }
  refresh_display();
}
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);
}
void echoCheck() {                    // Timer2 megszakításból hívjuk
  if (sonar.check_timer()) {          // Így ellenőrizzük, hogy volt-e már válasz
    sonar_val = sonar.ping_result;    // Eredmény kiolvasása
  }
}
```

A távolságmérő működés közben...





Az Arduino nano kártya kivezetései