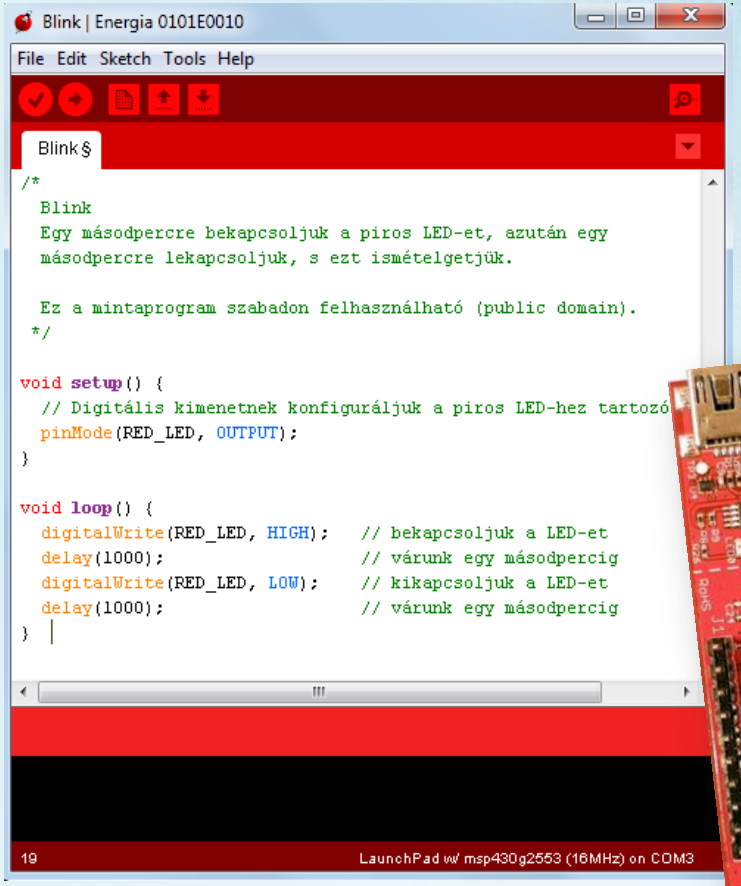
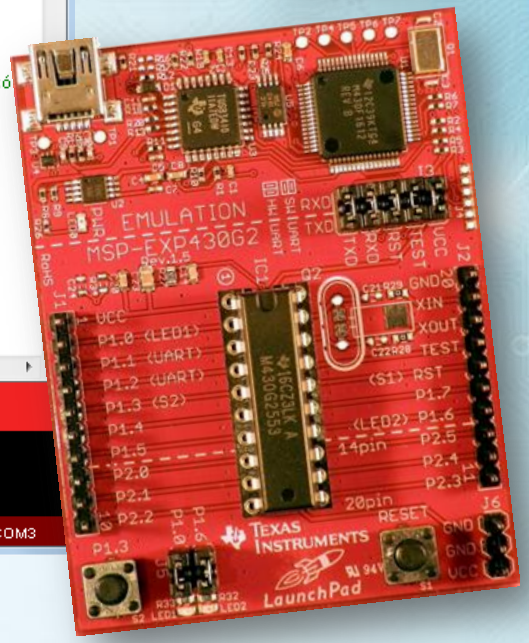


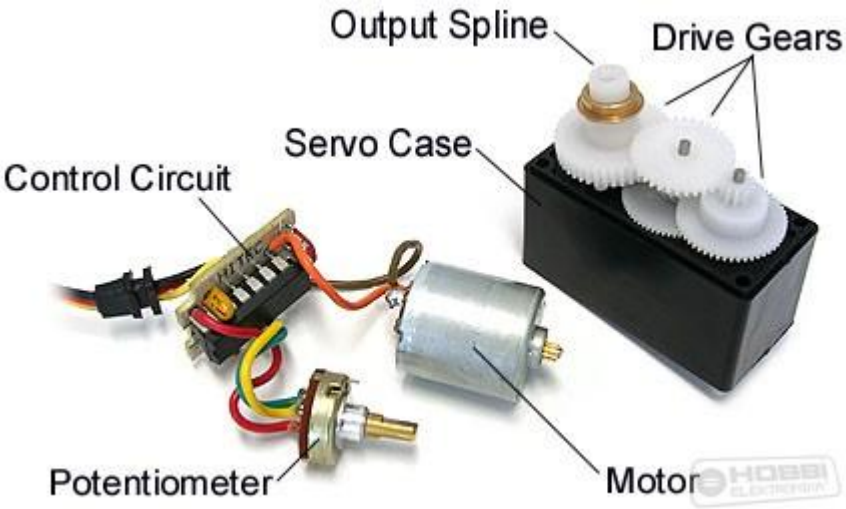


Energīa

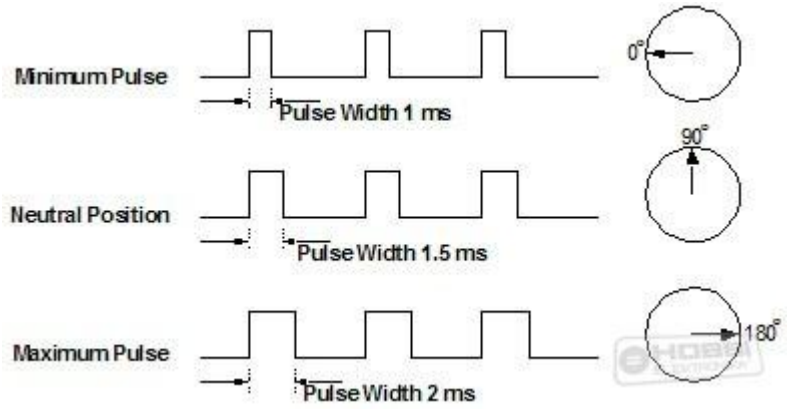
MSP430 programozás Energia környezetben

Szervó motorok vezérlése

Szervo motorok



Felépítés



Jelalak

Servo programkönyvtár

A „gyári” Servo programkönyvtár max. 8 db szervót kezel, s ezekhez felhasználja a Timer0 időzítőt, tehát közben az más célra (PWM vagy hangkeltés) nem használható.

A legfontosabb metódusok (részletes leírás egyelőre csak az Arduino Reference szekciójában található):

Servo.attach(pin[, minvalue, maxvalue]**)** – hozzárendeli a Servo objektumot a megadott kivezetéshez. A mikroszekundumokban mért határértékek (minimális és maximális impulzusszélesség) megadása nem kötelező. Az alapértelmezett értékek: 544 és 2400 µs.

Servo.write(adat) – a szervó beállítása (ha az adat < 200, akkor fokokban értendő, s a 0 – 180 tartományban vehet fel értéket, 200-nál nagyobb számokkal pedig mikroszekundumban adhatjuk meg az impulzusszélességet)

Servo.writeMicroseconds(adat) – a szervó beállítása (az adat itt mikroszekundumokban értendő, az inicializálásnál megadott minvalue.maxvalue tartományban)

Lab14

- **Servo_Sweep** – Egyszerű mintaprogram, amely a szervó motorral automatikusan oda-vissza végigpásztázza a 0-180 fokos tartományt
- **Servo_Knob** – „Gyári” mintaprogram, amelyben egy potméterrel vezéreljük a szervót
- **Servo_Set** – Mintaprogram, amelyben a soros portról vezéreljük a szervót
- **Walking_robot** – Egyszerű, két szervóval megvalósított lépegető robot működtető programja

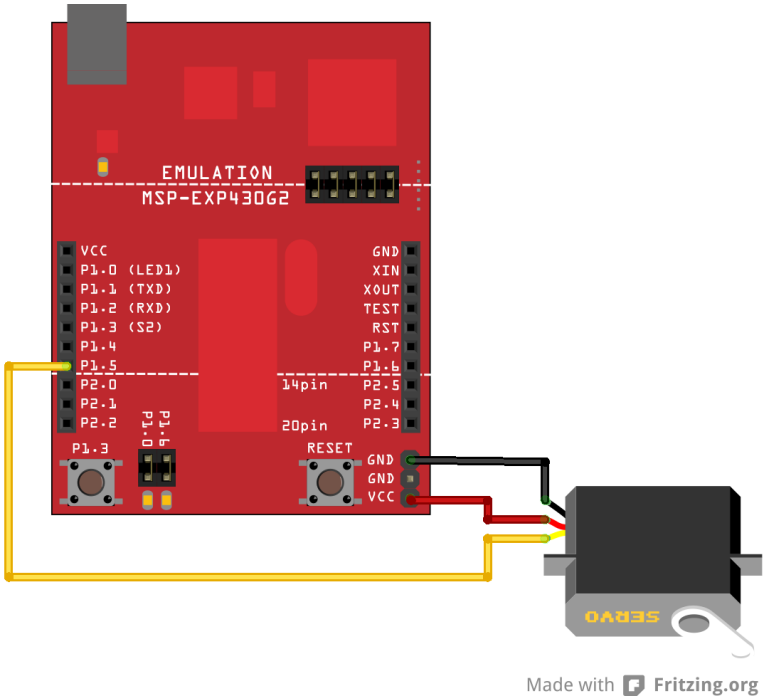
Servo_Sweep

A beépített mintaprogram (**File/Examples/Servo/Sweep**) picit módosított változata, amely a szervó motorral automatikusan oda-vissza végigpásztázza a 0-180 fokos tartományt.

A szervó bemenetei:

Fekete – GND
Piros – VCC
Sárga v.
Fehér – vezérlőjel

Hozzávalók
1 db szervó motor
3 db F+M vezeték
1 db Launchpad kártya



Kapcsolási vázlat

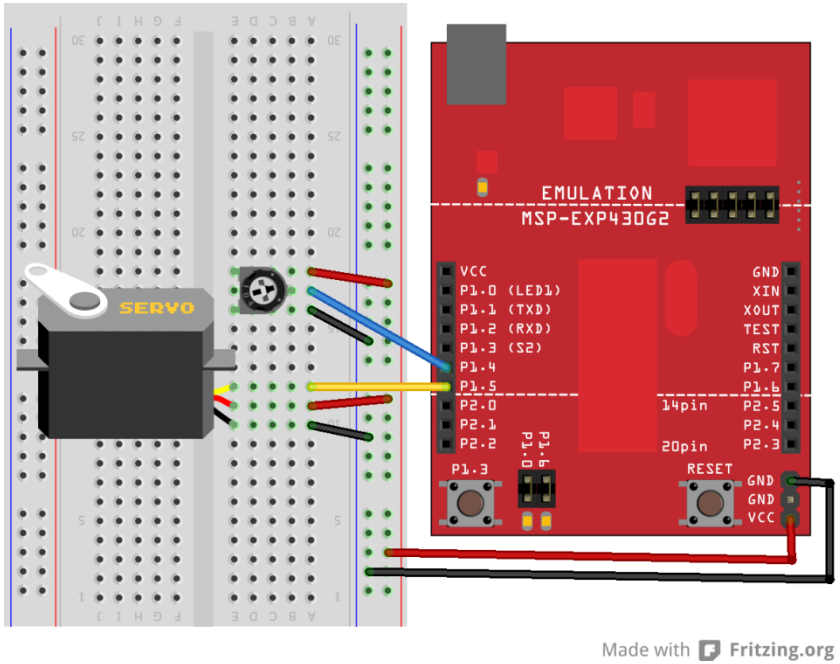
Lab14: Servo_Sweep.ino

```
#include <Servo.h>
const int servoPin = 7;              //Szervó a P1.5 lábra
Servo servo;                         //Objektum példányosítása
int angle = 0;                       //Szervó pozíciója fokokban

void setup() {
 servo.attach(servoPin,640,2400);    //Inicializálás határértékek megadásával
}

void loop() {
  // Pásztázás 0-tól 180 fokig
  for(angle = 0; angle < 180; angle++) {
    servo.write(angle);
    delay(100);
  }
  // Pásztázás vissza 180-tól 0 fokig
  for(angle = 180; angle > 0; angle--) {
    servo.write(angle);
    delay(100);
  }
}
```

Servo_Knob

A beépített mintaprogram (File/Examples/Servo/Knob) némileg módosított változata, amelyben az **AN4** bemenetre (P1.4 láb) kötött potméterrel vezéreljük a szervót.

Hozzávalók
1 db 10k potméter
1 db szervó motor
6 db F+M vezeték
1 db dugaszolós próbapanel
1 db Launchpad kártya



Kapcsolási vázlat

Lab14: Servo_Knob.ino

```
#include <Servo.h>
Servo myservo;                          //Objektum példányosítása
int potpin = 4;                         // A potmétert az A4 (P1.4) lábra kötjük
int val;                                // Változó az analóg feszültség beolvasásához

void setup() {
  myservo.attach(7,640,2400);           //Inicializálás határértékek megadásával
}                                       //Szervó a P1.5 lábra kötve (pin 7)

void loop() {
  val = analogRead(potpin);             //a potméter állásának beolvasása
  val = map(val, 0, 1023, 0, 179);      //a kapott szám átskálázása (0-180)
  myservo.write(val);                   //a szervó beállítása
  delay(100);                           // várunk, hogy a szervó beálljon
}
```

Az analóg referenciát nem definiáltuk, így alapértelmezetten VCC lesz.

A 10 bites ADC-vel 0 – 1023 közötti számot kapunk, amit át kell skáláznunk a 0 - 180 közötti tartományra. Ehhez az **y = map(x, minA, maxA, minB, maxB)** függvényt használjuk, ahol **x** az átszámítandó érték, **minA** és **maxA** a bejövő érték alsó és felső határa, **minB** és **maxB** pedig az átszámított eredmény alsó és felső határa. Az átskálázás az alábbi képlettel írható fel:

y = minB + (x - minA) * (maxB - minB)/(maxA - minA)

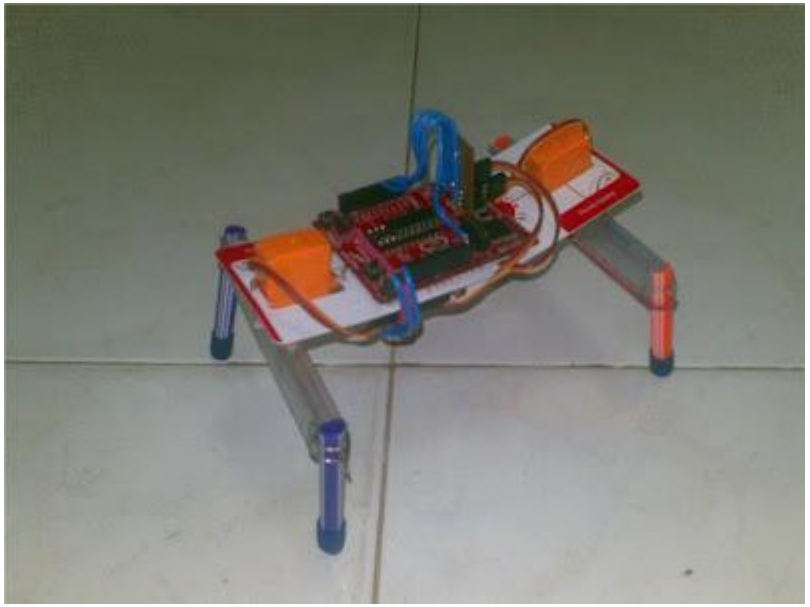
Lab14: Servo_Set.ino

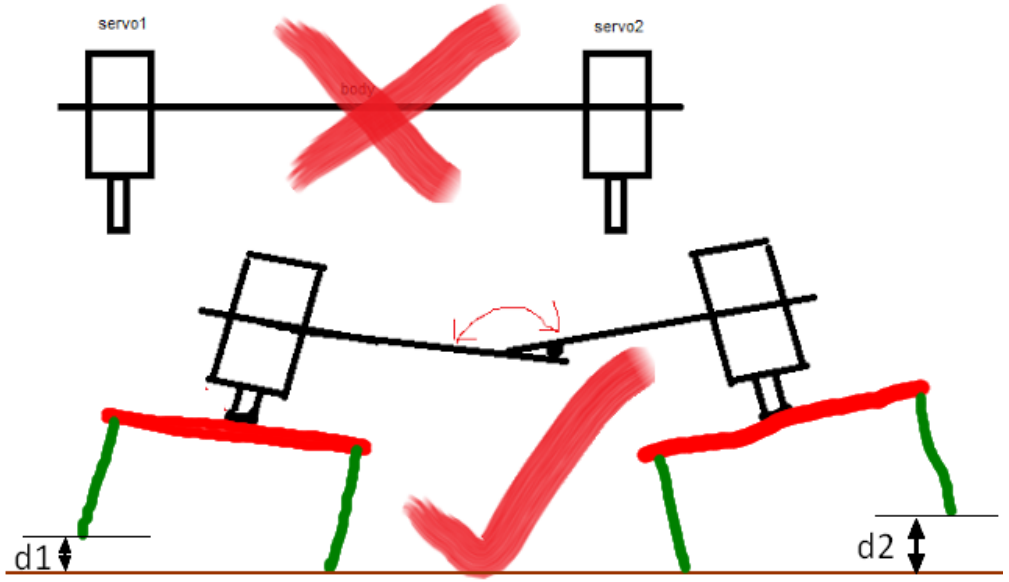
Szervó beállítása, soros porton keresztül megadott számmal.
A kapcsolás és a hozzávalók ugyanaz, mint a Servo_Sweep projektnél.

```
#include <Servo.h>
Servo servo;                          //Objektum példányosítása
const int servoPin = 7;               //Szervó a P1.5 lábra kötve (pin 7)

void setup() {
  Serial.begin(9600);                 //A soros port inicializálása
  servo.attach(servoPin,640,2400);    //Inicializálás határértékek megadásával
}

void loop() {
  if(Serial.available() > 0) {
    int angle = Serial.parseInt();    //Pozíció megadása soros porton
    angle = constrain(angle,0,179); //Korlátozzuk 0-179 közé!
    servo.write(angle);               //Szervó beállítása a megadott állásba
    Serial.print("Servo position: ");
    Serial.println(angle);
    delay(100);
  }
}
```

Lépegető robot

Már két szervó is elég lehet ahhoz, hogy egy egyszerű lépegető robotot építsünk.
Az ötlet és a zseniálisan egyszerű megvalósítás Vinod Stanur (Kerala, India) nevéhez fűződik.
Link: http://blog.vinu.co.in/2012/06/two-servo-walking-robot-using-ti.html





Walking robot (részletek)

Forrás: http://blog.vinu.co.in/2012/06/two-servo-walking-robot-using-ti.html

```
main:
    mov.b &CALIBC1_1MHZ, &BCSCTL1
    mov.b &CALDCO_1MHZ, &DCOCTL
    mov.b #0, &P1OUT
    mov.b #255,&P1DIR
    call #timer_init
    call #servo_init
    eint

infinite_loop:
    mov.w #5,r14
ntimes1:
    call #walk_forward
    dec r14
    jnz ntimes1
    mov.w #5,r14
    call #stop_idle
    call #delay
ntimes2:
    call #walk_reverse
    dec r14
    jnz ntimes2
    call #stop_idle
    call #delay
    jmp infinite_loop
```

CPU frekvencia 1 MHz

GPIO, timer, szervó inicializálás
Interrupt engedélyezés

5 lépés előre

5 lépés hátra

Folytatás a következő oldalon…

Walking_robot (részletek)

```
Stop_idle:
    stop_idle:
    mov.w #1500, &(SERVO1_ANGLE)
    call #delay
    mov.w #1500, &(SERVO2_ANGLE)
    call #delay
    ret

walk_forward:
    mov.w #1000, &(SERVO1_ANGLE)
    call #delay
    mov.w #1000, &(SERVO2_ANGLE)
    call #delay
    mov.w #2000, &(SERVO1_ANGLE)
    call #delay
    mov.w #2000, &(SERVO2_ANGLE)
    call #delay
    ret
```

```
walk_reverse:
    mov.w #1000, &(SERVO2_ANGLE)
    call #delay
    mov.w #1000, &(SERVO1_ANGLE)
    call #delay
    mov.w #2000, &(SERVO2_ANGLE)
    call #delay
    mov.w #2000, &(SERVO1_ANGLE)
    call #delay
    ret
```

Lab14: Walking_robot.ino

Az előző oldalakon bemutatott assembly program átirata Energia (Wiring) nyelvre

```
#include <Servo.h>
Servo servoA, servoB;          //Objektum példányosítás
void setup() {
  servoA.attach(6,640,2400); //Inicializálás határértékek megadásával
  servoB.attach(7,640,2400); //Inicializálás határértékek megadásával
}

void loop() {
  walk_forward(5);             //Öt lépés előre
  stop_idle();                 //Lábak alaphelyzetbe
  delay(2000);
  walk_reverse(5);             //Öt lépés hátra
  stop_idle();                 //Lábak alaphelyzetbe
  delay(2000);
}

void walk_forward(int n) {
  for(int i=0; i<n; i++) {
    servoA.writeMicroseconds(1000);    delay(200);
    servoB.writeMicroseconds(1000);    delay(200);
    servoA.writeMicroseconds(2000);    delay(200);
    servoB.writeMicroseconds(2000);    delay(200);
  }
}
```

Folytatás a következő oldalon…

Lab14: Walking_robot.ino

… folytatás az előző oldalról

```
void walk_reverse(int n) {
  for(int i=0; i<n; i++) {
    servoB.writeMicroseconds(1000);    delay(200);
    servoA.writeMicroseconds(1000);    delay(200);
    servoB.writeMicroseconds(2000);    delay(200);
    servoA.writeMicroseconds(2000);    delay(200);
  }
}

void stop_idle(void) {
  servoB.writeMicroseconds(1500);    delay(200);
  servoA.writeMicroseconds(1500);    delay(200);
}
```