Textul si imaginile din acest document sunt licentiate

Attribution-NonCommercial-NoDerivs
CC BY-NC-ND

Codul sursa din acest document este licentiat

Public-Domain

Esti liber sa distribui acest document prin orice mijloace consideri (email, publicare pe website / blog, printare, sau orice alt mijloc), atat timp cat nu aduci nici un fel de modificari acestuia. Codul sursa din acest document poate fi utilizat in orice fel de scop, de natura comerciala sau nu, fara nici un fel de limitari.

Arduino – software PWM

Ce este PWM ?

Termenul de PWM provine din limba engleza de la Pulse Width Modulation ceea ce inseamna Modulatia Pulsurilor in Lungime si este o tehnica de a simula un semnal analogic folosindu-te de un semnal digital.

Semnalul digital poate lua doar 2 stari: ON sau OFF ; „1“ sau „0“ ; 5V sau 0V. Un semnal PWM te ajuta sa obtii o multime de valori cuprinse intre 5 si 0V. De exemplu poti obtine 2.5V ceea ce inseamna ca poti ajusta luminozitatea unui LED sau viteza de rotatie a unui motor.

Semnalele PWM sunt utile in urmatoarele situatii: atunci cand vrei sa creezi jocuri de lumini daca ai la dispozitie o multitudine de LED-uri sau matrici, atunci cand vrei sa variezi viteza de rotatie a unui motor sau cand vrei sa controlezi unghiul unui servomotor. Am enumerat situatiile des intalnite dar pot exista si altele: daca doresti sa controlezi automat temperatura unei camere si sa folosesti un element de incalzire al carui raspuns sa fie proportional cu semnalul PWM, ori doresti sa produci sunete de frecvente diferite folosindu-te de un difuzor sau vrei sa ajustezi automat tensiunea unei surse de alimentare.

Semnalele PWM sunt foarte utile in diverse situatii iar placa Arduino nu duce lipsa de aceasta functie. O placa Arduino UNO poate genera pana la 6 semnale PWM prin pinii 3, 5, 6, 9, 10 si 11 deoarece microcontroller-ul este echipat cu un modul hardware special care se ocupa exclusiv de aceste semnale.

Pentru a afla mai multe detalii despre metoda PWM acceseaza link-urile de mai jos:

http://arduino.cc/en/Tutorial/PWM#.UxJAW_l_uSo

http://arduino.cc/en/Reference/analogWrite#.UxJFbvl_uSo

http://web.cecs.pdx.edu/~gerry/class/EAS199A/topics/pdf/PWM_output_Arduino.pdf

Totusi placa Arduino poate genera pana la 6 semnale si sunt cazuri cand doresti sa generezi un numar mai mare. Spre exemplu un hexapod are un numar foarte mare de servomotoare care il pun in miscare, iar un robot biped ajunge pana la 32 de servomotoare care concomitent lucreaza si il ajuta sa isi mentina echilibrul.

In situatiile de mai sus cei 6 pini PWM sunt insuficienti asa ca vei avea nevoie de mai multi. Aici iti vine in ajutor o tehnica de a genera semnale PWM nu hardware (folosindu-te de cei 6 pini) ci prin software. Asta inseamna ca, teoretic, orice pin digital de intrare/iesire de pe placa Arduino poate genera semnale PWM si asta ti-ar creste semnificativ numarul de pini.

De ce teoretic si nu practic ? Se pare ca metoda software PWM este mai greu de inteles si putin mai complexa. Daca prin metoda hardware PWM (cei 6 pini enumerati mai sus) te foloseai de cateva functii si instructiuni, la metoda software PWM este necesar sa scrii linie cu linie codul din spatele functiilor.

Cum generez un semnal PWM cu placa Arduino ?

Exemplu: Cum pot sa comand un motor brushless cu controller ESC (Electronic Speed Controller):

```
#include <Servo.h>

Servo esc;
int throttle = 0;
void setup()
{
  Serial.begin(9600);
  esc.attach(6);
  delay(15);
  esc.write(30);
  delay(2000);
}

void loop()
{
  for (throttle = 0; throttle <=179; throttle++ ) {
    esc.write(throttle);
    Serial.println(throttle);
    delay(400);
  }
for (throttle = 179; throttle >=0; throttle-- ) {
    esc.write(throttle);
    Serial.println(throttle);
    delay(400);
  }
}
```

Exemplul de mai sus te poate ajuta sa controlezi viteza de rotatie a unui motor brushless folosindu-te de un controller ESC. Controller-ul este responsabil cu intrepretarea semnalului PWM provenit de la placa Arduino si alimentarea motorului brushless.

Te vei folosi de libraria Servo pentru a realiza acest lucru.

Prima linie de cod, directiva preprocesor, ii spune compilatorului ce librarie vei folosi in program, respectiv libraria Servo.h

A doua linie: se declara obiectul esc adica motorul pe care placa Arduino il va comanda.

A treia linie: se declara variabila throttle si se initializeaza cu 0. Variabila te va ajuta sa variezi viteza de rotatie a motorului (vei vedea mai jos cum).

In rutina setup () se initializeaza Monitorul Serial, se ataseaza motorul esc la pinul 6 (unul din cei 6 pini PWM), se executa o intarziere de 15 milisecunde, se transmite catre motor un semnal PWM cu valoarea 30 si se asteapta 2 secunde.

De ce acest semnal PWM ? Raspunsul este ca ESC-urile (controller-ul electronic de viteza) necesita o armare inainte de a porni motorul. Armarea ii permite controllerului sa isi calibreze plaja de reactie. Aceasta plaja poate varia de la controller la controller (nu exista un standard) dar poate fi descoperita usor. Plaja de reactie inseamna minimul si maximul pe care il poate atinge motorul.

In rutina loop() se executa o bucla for() care incrementeaza variabila throttle de la 0 la 179 in pasi de cate o unitate.

Linia **esc.write(throttle)** transmite motorului viteza de rotatie, astfel ca la 0 motorul nu se misca deloc iar la 179 motorul se roteste cu viteza maxima. Practic bucla for() creste treptat viteza de rotatie a motorului.

Ultimele 2 linii din bucla transmite valoarea vitezei catre Monitorul Serial si introduce o intarziere de 400 de milisecunde in asa fel incat panta de acceleratie a motorului sa nu fie abrupta.

A doua bucla for() scade treptat viteza motorului in aceeasi maniera ca si prima.

Pentru a afla mai multe detalii despre motoarele brushless si tehnica de comanda a acestora acceseaza link-urile de mai jos:

http://techvalleyprojects.blogspot.ro/2012/06/arduino-control-escmotor-tutorial.html

http://dronesandrovs.wordpress.com/2012/11/24/how-to-control-a-brushless-motor-esc-with-arduino/

Codul sursa de mai sus este perfect functional si poate fi utilizat cu succes dar exista un dezavantaj si anume ca poti genera pana la 6 semnale PWM, ceea ce nu te ajuta foarte mult daca doresti sa comanzi un numar si mai mare de: motoare de curent continuu, servomotoare sau LED-uri.

Cum generez un semnal PWM software ?

Tehnica de generare a semnalului PWM software este putin mai complexa decat exemplul anterior. La inceput placa porneste cu iesirea PWM in „0“ logic. Urmeaza ca placa Arduino sa activeze un timer, acesta sa contorizeze iar cand se „umple“ sau cand a ajuns la capatul contorizarii, timer-ul sa genereze o intrerupere.

Ce este o intrerupere ?

Orice microcontroller executa un program, o rutina, genereaza semnale, citeste date, afiseaza informatii pe un display, comanda o turbina, s.a.m.d.

In timpul functionarii programului pot aparea situatii neprevazute. Iata un exemplu: sa presupunem ca placa Arduino executa un program care este in mare parte o rutina oarecare. Dar din cand in cand placa Arduino trebuie sa raspunda prompt la o intamplare, la ceva care apare din exterior sau din interior si sa faca lucrul asta fara sa deranjeze intr-un fel executia programului.

Raspunsul pentru acest gen de situatii este sa folosesti intreruperile microcontrollerului. Pentru fiecare intrerupere nou aparuta sau situatie, microcontroller-ul sare rapid din executia programului apoi executa programul special de intrerupere (intr-un timp foarte rapid) si apoi se intoarce de unde a plecat si isi continua treaba de la care a plecat.

In stilul asta placa Arduino executa programul principal dar in acelasi timp poate executa si anumite coduri specifice intreruperii.

Iata un exemplu de generare a unui semnal PWM folosind un timer si o intrerupere:

```
#include <avr/interrupt.h>
#include <avr/io.h>

// Pinul pe care dorim sa generam semnal PWM
#define PWM_PIN 7


#define PWM_TICK_SIZE 50 // uS

#define PWM_PERIOD   20000 // uS (20ms)

#define PWM_TICK_PERIOD (int) (PWM_PERIOD / PWM_TICK_SIZE)

volatile unsigned int timer2_counter = 0;
volatile unsigned int timer2_trigger_low = 150;

ISR(TIMER2_OVF_vect) {
  timer2_counter++;
  if(timer2_counter == timer2_trigger_low) {
    digitalWrite(PWM_PIN, LOW);
  } else if(timer2_counter >= PWM_TICK_PERIOD) {
    timer2_counter = 0;
    digitalWrite(PWM_PIN, HIGH);
  }

  TCNT2 = 0;
}


void setup() {
  pinMode(PWM_PIN, OUTPUT);
  TIMSK2 = 1<<TOIE2;  // Timer 2 overflow interrupt enable
  TCNT2 = 0;
}
```

```
void loop() {
  timer2_trigger_low = 100;
  delay(2000);
  timer2_trigger_low = 200;
  delay(2000);
  timer2_trigger_low = 300;
  delay(2000);
  timer2_trigger_low = 400;
  delay(2000);
  timer2_trigger_low = 300;
  delay(5000);
  timer2_trigger_low = 200;
  delay(2000);
}
```

Cum functioneaza codul ?

Primele doua linii, directivele preprocesor, includ librariile responsabile cu intreruperile microcontrollerului Atmega si registrii acestuia. Mai jos vei vedea cum se acceseaza in mod direct anumiti registrii din microcontroller si cum se activeaza timer-ul + intreruperea.

A treia linie: defineste pinul pe care placa Arduino va genera semnalul PWM. A fost ales un pin pe care nu se poate genera semnal PWM hardware.

Prin urmatoarele 3 linii se definesc cativa parametrii: primul parametru PWM_TICK_SIZE iti spune ca timer-ul va contoriza pentru 50 de microsecunde, al doilea parametru PWM_PERIOD iti spune care este perioada totala a semnalului PWM iar al treilea parametru PWM_TICK_PERIOD se obtine din raportul celorlalti 2 parametrii si iti arata cand se va reseta semnalul PWM din „1“ logic in „0“ logic.

Urmatoarele 2 linii initializeaza 2 variabile dintre care una va mentine valoarea contorului (timer-ul) iar cealalta mentine valoarea la care semnalul PWM sa isi schimbe nivelul logic.

Functia ISR este apelata ori de cate ori timer-ul genereaza o intrerupere adica ori de cate ori timer-ul a terminat de contorizat. La fiecare intrerupere gestionata de catre microcontroller se incrementeaza timer2_counter.

Pentru inceput tensiunea pe pinul 7 PWM este de 0V. Apoi se testeaza daca variabila incrementata este egala cu parametrul timer2_trigger_low. Daca este egala atunci pinul 7 este in continuare mentinut cu 0V.

Se continua incrementarea timp in care tensiunea pinului PWM este de 0V. Tensiunea se schimba in 5V atunci cand variabila incrementata este mai mare sau egala decat parametrul PWM_TICK_PERIOD. Tensiunea ramane la 5V, contorul de timp este resetat si din nou incrementat.

Din nou se incrementeaza contorul la fiecare intrerupere, timp in care tensiunea pe pinul PWM ramane la 5V. Tensiunea se schimba in 0V atunci cand se produce egalitatea intre variabila incrementata si parametrul timer2_trigger_low.

In acest fel s-a generat un semnal PWM folosindu-te de un timer si o intrerupere.

In rutina setup() se initializeaza pinul PWM, intreruperea timerului 2 prin scrierea bitului TOIE2 cu valoarea logica „1“ din registrul TIMSK2 (am mentionat ca vei afla cum se acceseaza in mod

direct un registru) si se reseteaza valoarea contorului pin scrierea registrului TCNT2 = 0;

Pentru a demonstra ca semnalul PWM isi poate schimba factorul de umplere (lungimea nivelului „1“), in bucla loop() se modifica treptat la intervale de 2 secunde valoarea parametrului timer2_trigger_low. Asta inseamna ca nivelele semnalului PWM se vor declansa la momente diferite (100, 200, 300, 400, 300, 200).

Daca vrei sa studiezi in detaliu sistemul de intreruperi al microcontrollerului Atmega328 de pe placa Arduino urmeaza link-urile de mai jos:

http://courses.cs.washington.edu/courses/csep567/10wi/lectures/Lecture7.pdf

http://www.avr-tutorials.com/interrupts/about-avr-8-bit-microcontrollers-interrupts

http://www.avrfreaks.net/index.php?name=PNphpBB2&file=viewtopic&t=89843&start=all&postdays=0&postorder=asc

http://www.avr-tutorials.com/interrupts/avr-external-interrupt-c-programming

http://playground.arduino.cc/Code/Interrupts#.UxJiTvl_uSo

Tot ce iti ramane de facut este sa descoperi cum poti genera mai multe semnale PWM folosindu-te de aceeasi tehnica software de generare a semnalelor. In principiu trebuie sa ai in vedere declararea in plus a unor parametrii, cate un set pentru fiecare pin PWM si rescrierea rutineri de intrerupere in asa fel incat sa fie capabila sa gestioneze mai multi pini.

Acest lucru iti da un avantaj pentru ca poti creea miscari complexe cu servomotoare, poti controla motoare la viteze diferite (poti creea o tractiune diferentala in care, pe scurt, rotile unei masini trebuie sa se roteasca la viteze diferite atunci cand masina vireaza intr-o curba) sau poti creea un joc de lumini cu fiecare LED controlat independent.