Arduino-kompendium – MEKMEK01

En samling av teori om hårdvara och tekniker, samt referens för programmering för att använda Arduino i kursen MEKMEK01 på MTU.

Innehåll

1 Teori

Här förklaras lite grundläggande koncept som behövs för att förstå Arduino.

1.1 Vad är en pin

En pin är en fysisk kontakt på en mikrokontroller som kan användas för att koppla in och ut signaler. En pin kan vara en ingång eller en utgång (se 1.2). De pins som heter D0-D13 kan användas för digitala signaler, medan A0-A5 kan användas för analoga signaler (se 1.3).

1.2 Utgång eller ingång?

Du som programmerare väljer om en pin ska vara en ingång eller utgång genom `pinMode` (se 2.1). En ingång kan användas för att läsa av en signal, till exempel från en knapp eller en sensor. En utgång kan användas för att skicka en signal, till exempel till en lampa eller en motor.

1.3 Digital eller analog?

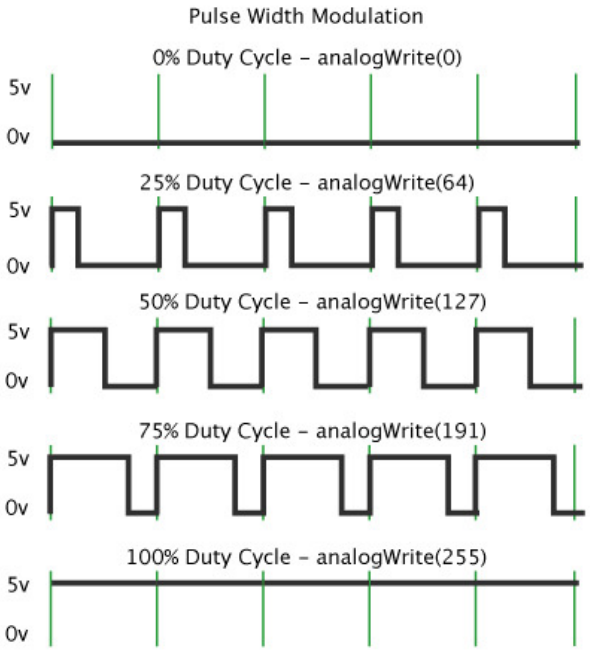
En digital signal kan bara ha två tillstånd: hög eller låg. En analog signal kan ha ett kontinuerligt värde mellan hög och låg. En analog signal kan till exempel användas för att styra en motor så att den snurrar med olika hastighet, eller en lampa så att den lyser med olika styrka.

1.4 PWM

Pulse Width Modulation (PWM) är en teknik som används för att skapa en analog signal med hjälp av digitala signaler. Detta är användbart för att styra till exempel motorer och lampor med en mikrokontroller.

PWM fungerar genom att skicka en serie av snabba höga och låga signaler, där förhållandet mellan hög och låg signal bestämmer hur stark den genomsnittliga signalen blir.

Se figur 1 för att se hur en PWM-signal ser ut vid olika procent.



Figur 1: Hur PWM-signaler ser ut vid olika procent.

1.5 Enpulsning

Väldigt ofta är vi intresserade av när en signal precis går från låg till hög (så kallad **positiv flank** eller **rising edge** på engelska), inte bara *att* signalen är hög. För att detektera en positiv flank kan enpulsning användas.

Ett sätt att genomföra enpulsning av en signal i Arduino är att spara det förra tillståndet av signalen, och jämföra det med det nuvarande tillståndet. Om det förra tillståndet var låg och det nuvarande är högt, har en positiv flank inträffat.

Exempelkod: Här enpulsas signalen på pin D2:

```
bool lastState = LOW;
void setup() {
  pinMode(2, INPUT);
}
void loop() {
  bool currentState = digitalRead(2);
  if (lastState == LOW && currentState == HIGH) {
    // En positiv flank har inträffat
  }
  lastState = currentState;
}
```

1.6 Datatyper

blavlajsadj

2 Programmering

2.1 `pinMode`

Används för att bestämma om en pin ska vara en utgång eller ingång (1.2)

`pinMode` har tre konfigurationer:

- `INPUT`
- `OUTPUT`
- `INPUT_PULLUP`

Exempelanvändning:

```
pinMode(4, OUTPUT);
```

Kommer att välja pin D4 som en utgång.

2.1.1 `INPUT_PULLUP`

En variant av input. Denna konfiguration gör så att pin:en internt kopplas till en så kallad pull-up resistor. Det betyder att när pinen inte är kopplad till GND kommer den att vara hög.

Detta är användbart när man kopplar in en knapp, eftersom då behövs endast två sladdar: en till GND och en till pinen.

Obs! När knappen är nedtryckt kommer alltså signalen vara låg, annars hög. Det är därför vanligt att man vill invertera den avlästa signalen.

Exempelanvändning: En knapp kopplad till pin D2:

```
void setup() {
  pinMode(2, INPUT_PULLUP);
}
void loop() {
  bool buttonState = !digitalRead(2); // inverterad logik
  if (buttonState) {
    // Knappen är nedtryckt
  }
}
```

2.2 `digitalWrite`

För att styra signaler på en pin som är bestämd som utgång. Kan användas för att tända eller släcka en lampa, eller styra en motor. Eftersom en digital signal bara kan vara hög eller låg, kan `digitalWrite` bara skicka signaler med dessa två tillstånd.

Exempelanvändning: För att skicka ut en hög signal på pin D4:

```
digitalWrite(4, HIGH);
```

Exempelanvändning: För att släcka den interna lampan på Arduino-enheten:

```
digitalWrite(LED_BUILTIN, LOW);
```

2.3 `digitalRead`

För att läsa av en pin som är bestämd som ingång. Kan användas för att läsa av en knapptryckning eller en digital sensor. Returnerar `HIGH` eller `LOW` beroende på om spänningen på pinnen är hög eller låg. Eftersom en digital signal bara kan vara hög eller låg, kan `digitalRead` bara läsa av dessa två tillstånd.

Det lämpar sig bra att lagra resultatet i en variabel med typen `bool`.

Exempelanvändning: För att läsa av en hög insignal på pin D11:

```
bool value = digitalRead(11);
```

2.4 `analogWrite`

För att skriva en analog signal till en pin som är bestämd som utgång. Den valda pinen måste dessutom vara en av de pins som stödjer PWM.

2.5 `delay`

Används för att vänta en viss tid innan programmet fortsätter. Kan användas för att skapa en paus i programmet, till exempel för att blinka en lampa med ett visst intervall.

Exempelanvändning: För att tända en lampa på pin D4 i 1000 millisekunder, sedan släcka den i 1000 millisekunder:

```
digitalWrite(4, HIGH);
delay(1000);
```

```
digitalWrite(4, LOW);
delay(1000);
```

En varning: många problem kan uppstå om `delay` används i större program, eftersom programmet inte kan göra något annat under tiden som `delay` körs. Det finns bättre sätt att vänta i programmet, till exempel `millis`.

2.6 Seriell kommunikation

2.6.1 `Serial.begin`

Måste alltid anropas i `void setup()` för att överhuvudtaget kunna använda seriell kommunikation. Argumentet är den baudrate som ska användas. En standard-baudrate är 9600.

Exempelanvändning: För att starta seriell kommunikation med baudrate 9600:

```
Serial.begin(9600);
```

2.6.2 `Serial.print`

Används för att skicka meddelanden genom seriell kommunikation. Vanligtvis vill man också att en radbrytning ska skickas efter meddelandet, vilket kan göras med `Serial.println`.

Exempelanvändning: För att skicka texten “Hello, world!” till seriell kommunikation:

```
Serial.print("Hello, world!");
```

2.6.3 `Serial.println`

Har exakt samma användning som `Serial.print` med skillnaden att den skickar en radbrytning efter meddelandet. Om man vill använda den seriella plottern är det enklast att använda `Serial.println`.

Exempelanvändning: För att skriva ut antalet millisekunder sedan Arduino-enheten startades varje loop:

```
void loop() {
  Serial.println(millis());
}
```

2.7 `millis`

Används för att mäta tid. Vid anrop returneras antal millisekunder sedan Arduino-enheten startades. En begränsning är att det inte går att mäta längre tid än vad datatypen `unsigned long` kan rymma, vilket är ungefär 50 dagar.

Exempelanvändning:

För att blinka en lampa varje sekund, utan att använda `delay`:

```
void loop() {
  unsigned long currentTime = millis(); // Hämta aktuell tid i millisekunder
  // Kontrollera om tillräcklig tid har passerat sedan senaste blinkningen
  if (currentTime - lastTime >= interval) {
    digitalWrite(ledPin, !digitalRead(ledPin)); // Byt tillstånd på lampan
    lastTime = currentTime; // Uppdatera senaste tiden
  }
}
```