Dette dokument er et forsøg på at beskrive hvordan man kan undgå at bruge funktionen delay().

Inside delay(); Brug Millis, Kodeeksempler, Hvordan virker millis(), Roll-Over efter 49 dage, Micros(), Kilder,

Undgå Funktionen delay()

Funktionen delay() er ofte praktisk at bruge i et program i Arduino-verdenen.

Man skriver fx blot ” delay(100); ” for at få Microcontrolleren til at vente 100 mS inden programmet fortsætter.

Men nogle gange er det et problem, at der i den tid, delay-perioden varer, ikke kan udføres andre funktioner, fx at tjekke om en knap er trykket.

Man kan heller ikke få fx en lysdiode – blinke - sekvens til at vare i en given periode.

Funktionen delay() **stopper nemlig alt anden programafvikling** i delay-perioden.

Et kig ind i den bagvedliggende kode for delay():

Delay() er en funktionen der er defineret i den bagvedliggende kode (i et bibliotek) til Arduinos IDE. Compileren oversætter kildeteksten, og sørger for at addere de nødvendige kodestumper – dvs. subrutiner – eller funktioner – til den endelige hex-kode, der udgør programmet og uploades i Unoens Atmega328P

```
// Inside delay :-)

void delay(unsigned long ms)       // ms er en variable på 32 bit
{
  unsigned long start = millis(); // Read current millisec after start

  while (millis() - start <= ms) {
 // Wait ms millisec. Just wait!!

} ;
```

Parameteren til delay(), som medtages ved kald til delay-funktionen som variablen ms, er den ønskede delaytid i millisekunder.

Brug funktionen millis() til delay for at omgå at processoren "hænger".

Som det ses ovenfor, bruger Delay() en anden funktion, millis() til at bestemme tiden. Men den funktion kan vi jo selv bruge.

Funktionen Millis() returnerer et tal, der angiver det antal millisekunder, der er gået siden programmet i Microcontrolleren blev startet. Dvs. at når værdien, der returneres, fx er blevet øget med 100, er der gået 100 mS siden bootning af controlleren.

Her et eksempel på kode, der udnytter dette:

Strategi:

Når en tidsperiode skal starte gemmes først den aktuelle tid (timestamp) i en variabel.

```
unsigned long ms = 900;  // ønsket varighed

unsigned long Start = millis();  // gem timestamp for start

while (millis() - start < ms) ;  //
{
   // do this until time gone
}
```

Her udføres "noget" indtil der er gået et antal ms.

Men programmet hænger jo igen i While-løkken! Og hvis nogen fx trykker på en knap, bliver knappen jo ikke læst i den periode.

I næste eksempel loopes der, og der testes i hver loop, om der er gået en bestemt tid:

Der udmåles en tid, - men programmet " hænger ikke ".

```
// denne kode looper normal, men der udføres noget hver 500 mS.

unsigned long time_now;

// I setup():

time_now = millis();     // gem "nu-tiden"

void loop() {

    if ( (millis()-time_now) > 500) {    // udfør kun, hvis der er gået 500 ms.
```

```
        time_now = millis();    // gem ny "nu-tid" - genstart tidsudmål.

       // Do something every after 500ms

     }

   // Do something else every loop

}
```

// Kilde: https://www.best-microcontroller-projects.com/arduino-millis.html#Arduino_millis_to_Seconds

Her kommer flere eksempler:

```
// Variable defineres til unsigned long

unsigned long interval = 5000;  //

unsigned long timestamp; // def var og type

void setup(){
   timestamp = millis();
}

void loop() {

  if(millis() - timestamp > interval) {

    timestamp = millis();    // save the last time

    // do stuff here each interval
  }

// do other stuf
}
```

Eksempel: Blink lysdiode:

```
unsigned long time_now = 0;   // Def variable "Now"

int toggle = 1;

/*********************************************
 * setup() function
 *********************************************/

void setup()  {

  pinMode(10, OUTPUT);        //Red LED
  pinMode(11, OUTPUT);        //Green LED
```

```
  pinMode (12, INPUT_PULLUP); //Switch med intern pull up.

  digitalWrite(10, HIGH);     //Initial state of red LED
}  // Endsetup

/*******************************************
 * loop() function
 *******************************************/

void loop()
{
  if(millis()-time_now > 1000)      //Has one second passed?
  {
    toggle = !toggle;           //If so, toggle = not toggle ( inverter ) !!
    digitalWrite(10, toggle);  //toggle LED
    time_now = millis();            //and reset time-now.
  }

  digitalWrite(11, !digitalRead(12)); // LED dependent of switch.
} // Endloop
```

// Kilde: https://www.instructables.com/id/Beginning-Arduino-delay-without-delay/

Her følger nogle kodeeksempler:

```
unsigned long interval=1000;     // the time we need to wait
unsigned long previousMillis = 0;           // millis() returns an unsigned
long.

bool ledState = false;           // state variable for the LED
unsigned long currentMillis;

void setup() {
 pinMode(13, OUTPUT);
 digitalWrite(13, ledState);
} // Endsetup


void loop() {
  currentMillis = millis(); // grab current time

           // check if "interval" time has passed (1000 milliseconds)

 if ((unsigned long)(currentMillis - previousMillis) >= interval) {

   ledState = !ledState; // "toggles" the state

   digitalWrite(13, ledState); // sets the LED based on ledState

   previousMillis = millis();   // save the "current" time

 }
} // Endloop
```

Seriel Print ” Hello” hvert sekund:

```
/*
millis() won’t prevent us from running code while “waiting”.

Let’s say we want to print “Hello” over serial once each second while doing
other stuff in the meantime.

*/

int period = 1000;
unsigned long time_now = 0; // ved bootning er millis() jo = 0

void setup() {
    Serial.begin(9600);
} // Endsetup

void loop() {
    if((millis() - time_now) > period){
        time_now = millis();
        Serial.println("Hello");
    }

    //Run other code
}  // Endloop
```

Her blinkes 2 lysdioder med hver deres frekvens:

```
// each "event" (LED) gets their own tracking variable
unsigned long previousMillisLED12=0;
unsigned long previousMillisLED13=0;
unsigned long currentMillis=0;

// different intervals for each LED
int intervalLED12 = 500;
int intervalLED13 = 1000;

// each LED gets a state varaible
boolean LED13state = false;     // the LED will turn ON in the first iteration
of loop()
boolean LED12state = false;     // need to seed the light to be OFF

void setup() {
   pinMode(13, OUTPUT);
   pinMode(12, OUTPUT);
}  // Endsetup
```

```
void loop() {
   // get current time stamp
   // only need one for both if-statements

   currentMillis = millis();

   // time to toggle LED on Pin 12?

   if ((unsigned long)(currentMillis - previousMillisLED12) >= intervalLED12)
{
      LED12state = !LED12state;
      digitalWrite(12, LED12state);

      // save current time to pin 12's previousMillis

      previousMillisLED12 = currentMillis;
   }

// time to toggle LED on Pin 13?

  if ((unsigned long)(currentMillis - previousMillisLED13) >= intervalLED13) {
      LED13state = !LED13state;
      digitalWrite(13, LED13state);

      // save current time to pin 12's previousMillis

      previousMillisLED13 = currentMillis;
  }
} // Endloop
```

// Kilde: https://www.baldengineer.com/millis-tutorial.html

Test ovenstående:

Single shot timer

```
int led = 13;
unsigned long timer;             // the timer
boolean timedOut = false;        // set to true when timer fired
unsigned long INTERVAL = 5000;   // the timeout interval

void setup() {
  pinMode(led, OUTPUT);          // initialize LED output
  timedOut = false;              // allow timer to fire
  timer = millis();              // start timer
}  // Endsetup

void loop() {

  // this will toggle the led ONCE only after 5sec (timeOut)

  if ((!timedOut) && ((millis() - timer) > INTERVAL)) {
    // timed out
    timedOut = true;             // don't do this again

                                 // you can reset the single shot timer by
```

```
                                    // setting timedOut = false;
                                    // timer = millis();
                                    // toggle led
    if (digitalRead(led) == 1) {
      digitalWrite(led, LOW);    // turn the LED off

    } else {

      digitalWrite(led, HIGH);   // turn the LED on
    }
  }
} // Endloop
```

Kilde: https://www.forward.com.au/pfod/ArduinoProgramming/TimingDelaysInArduino.html

Opgave: lav do kode om så der er lys de første 5 sek efter bootning.

Opgave: køb lys 5 sek efter tryk på knap

Mere til historien: RoolOver problem efter 49 dage – og - Hvordan virker Millis()

Roll-Over efter 49 dage

Bemærk, at variablen timer0_millis er en unsigned long. Altså 32 bit, og kun positive tal.

Dvs. den tæller antal millisekunder fra programstart indtil roll-over. Det sker efter (2^{32} - 1). millisekunder. Dvs. 2^{32} / (1000 * 60 * 60 * 24) som svarer til 49,71 dage, altså knap 50 dage.

Herefter starter værdien af talt antal millisekunder igen forfra !!

Derfor kan der opstå fejl ved roll over. Det er vist i følgende kode!

```
/*
Here we will get a buggy behavior after approximately
50 days when millis() will go from returning a very
high number (close to (2^32)-1) to zero.
This is known as overflow or rollover.
*/

int period = 1000;
unsigned long time_now = 0;

void setup() {
    Serial.begin(9600);
}

void loop() {
    if(millis() > time_now + period){  // FEJL !!! Duer ikke
        time_now = millis();
        Serial.println("Hello");
    }
```

```
    //Run other code
}
```

Men med følgende kode fungerer det:

```
if (currentMillis - startMillis >= period)
```

Eksempel:

```
/**************
* Her er vist en metode til at omgå den situation, der ville opstå efter
* 49 dage, hvor millis() vil give overflow.
*
* Ved at bruge en unsigned long til udregningen af millis() - time_now,
* vil resultatet altid være positivt, netop fordi en unsigned long ikke kan
* indeholde et negativt tal !
*
* "to be sure that this works across multiple platforms and architectures
* you can explicitly cast the result of the calculation to the correct
* data type like this:"
*/

int period = 1000;  // Delay Time
unsigned long TimeOfDelayStart = 0;

  //---------------------------------

void setup() {
    Serial.begin(9600);
}
  //---------------------------------

void loop() {
    if((unsigned long)(millis() - TimeOfDelayStart) > period){  //
        TimeOfDelayStart = millis();
        Serial.println("Hello");
    }

    //Run other code
}
```

Kilde: <https://www.norwegiancreations.com/2018/10/arduino-tutorial-avoiding-the-overflow-issue-when-using-millis-and-micros/>

Hvis man udfører beregningen: Millis() – Startværdi vil det jo give et negativt resultat når millis()-tælleren giver overflow.

Men en "Unsigned long" kan bare ikke indeholde et negativt tal.

Forklaring:

Example 1:

prev = 240, cur = 255
cur – prev = 11111111 – 11110000

Computers do subtraction by adding a 2's complement number.
11110000 in 2's complement = 00010000

A way of finding the two's complement of a number is to take its ones' complement and add one:

The **ones' complement** of a binary number is defined as the value obtained by inverting all the bits in the binary representation of the number (swapping 0s for 1s and vice versa).

cur – prev = 11111111 + 00010000

```
  1 1 1 1 1 1 1 1
+ 0 0 0 1 0 0 0 0
-----------------
  1 0 0 0 0 1 1 1 1
```

Since it's an 8-bit number, the leftmost 1 is omitted.

= 00001111 = 15, as expected.

Example 2:

prev = 240, cur = 0 (overflowed)
cur – prev = 00000000 – 11110000

-11110000 in 2's complement = 00010000

cur – prev = 00000000 + 00010000

```
  0 0 0 0 0 0 0 0
+ 0 0 0 1 0 0 0 0
-----------------
  0 0 0 1 0 0 0 0
```

= 00010000 = 16

unsigned 32-bit subtraction also works with the same principle.

Kilde: https://www.baldengineer.com/arduino-how-do-you-reset-millis.html

Test og bevis:

```
/*Test og bevis:

This will print 2 to the serial monitor.
The Serial.print() function does not modify
the answer in any way in this case.

*/

    unsigned long a = 1;
    unsigned long b = 4294967295; //unsigned long maximum value = 0xFFFFFFFF


void setup() {
    Serial.begin(9600);

    Serial.println(a-b);
}

void loop() {
```

```
}
```

Hvordan virker Millis()

Arduino Unos microcontroller ATmega328P har 3 indbyggede timere. Timer0, Timer1 og Timer2.

Timer0 bruges til at generere et millisekund interrupt der updater en millisekund-tæller. Det er denne tæller, der læses med funktionen millis().

Det betyder jo så også, at nogle af de andre funktioner, der bruger Timer0 ikke vil fungere.

Krystallets 16 MHz ledes til tælleren Timer0 via en prescaler (en frekvensdeler), der deler frekvensen med 64.

Det giver 250 KHz til tælleren. Dvs. at Timer0 tæller 1 frem hver 4. uS.

Tælleren er på 8 bit, dvs. den kan tælle fra 0 til 255.

Tælleren genererer et interrupt ved overflow, eller ” roll over ”, dvs. går fra 255 til 0. Dette udløser et interrupt med en frekvens på 250 kHz / 256 = 976,5625 Hz.

Svarende til hver 1 / 976,5625 Hz = 0,001024 sekund. Dvs. hver 1024 uS eller 1,024 millisekund.

Dvs. hver 1,024 mS køres et timer0-interruptprogram, der opdaterer tællere, der følgelig indeholder antal millisekunder efter programmet blev startet.

Men fordi interrupt-frekvensen ikke er på nøjagtig 1000 Hz, må interrupt-rutinen en gang imellem korrigere millisekund-tælleren.

Det kan ses i interruptkoden.

```
SIGNAL(TIMER0_OVF_vect) {
  timer0_millis += 1;
  timer0_fract += 3;
  if (timer0_fract >= 125) {
    timer0_fract -= 125;
    timer0_millis += 1;
  }
  timer0_overflow_count++;
}
```

The overflow handler increments the value of timer0_millis every 1.024ms and then adds another increment to timer0_millis (catches up, if you will) every time timer0_fract is greater than 125.

Hence, timer0_millis accounts for the missing 0.024ms from every “timer overflow” at intervals of 125/3 = 41.67ms. Which means timer0_millis accumulates an error of 0.024ms each time it executes (every overflow), until the error approaches 1ms. At which time, timer0_millis jumps by 2 and corrects itself.

https://ucexperiment.wordpress.com/2012/03/16/examination-of-the-arduino-millis-function/

Så funktionen millis() returnerer ” bare ” værdien af variablen timer0_millis.

Funktionen millis() ser således ud:

```
unsigned long millis() {
  return timer0_millis;
}
```

Funktionen Micros()

Ud over millis() er der i Arduino-verdenen også en funktion kaldet micros().

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes.

```
void loop(){
  Serial.print("Time: ");
  time = micros();

}
```

I micros()-funktionen udregnes antal microsekunder siden start af programmet simpelt af ligningen:

(millis() * 1000) + (TCNT0 * 4)

```
unsigned long micros() {
    unsigned long m;
    uint8_t oldSREG = SREG, t;
    cli();
    m = timer0_overflow_count;
```

```
    t = TCNT0;
    if ((TIFR0 & _BV(TOV0)) && (t < 255))
        m++;
    SREG = oldSREG;
    return ((m << 8) + t) * (64 / clockCyclesPerMicrosecond());
}

/*
The code above allows for the overflow (it checks the TOV0 bit)
so it can cope with the overflow while interrupts are off but only once,
  - there is no provision for handling two overflows.
  */
```

Flere Kilder

RollOver: https://arduino.stackexchange.com/questions/12587/how-can-i-handle-the-millis-rollover

Kilde: https://stackoverflow.com/questions/37375602/arduino-millis-in-stm32

https://www.best-microcontroller-projects.com/arduino-millis.html

https://ucexperiment.wordpress.com/2012/03/16/examination-of-the-arduino-millis-function/