CPROUC/2021-2022

Ad van den Bergh

CPROUC

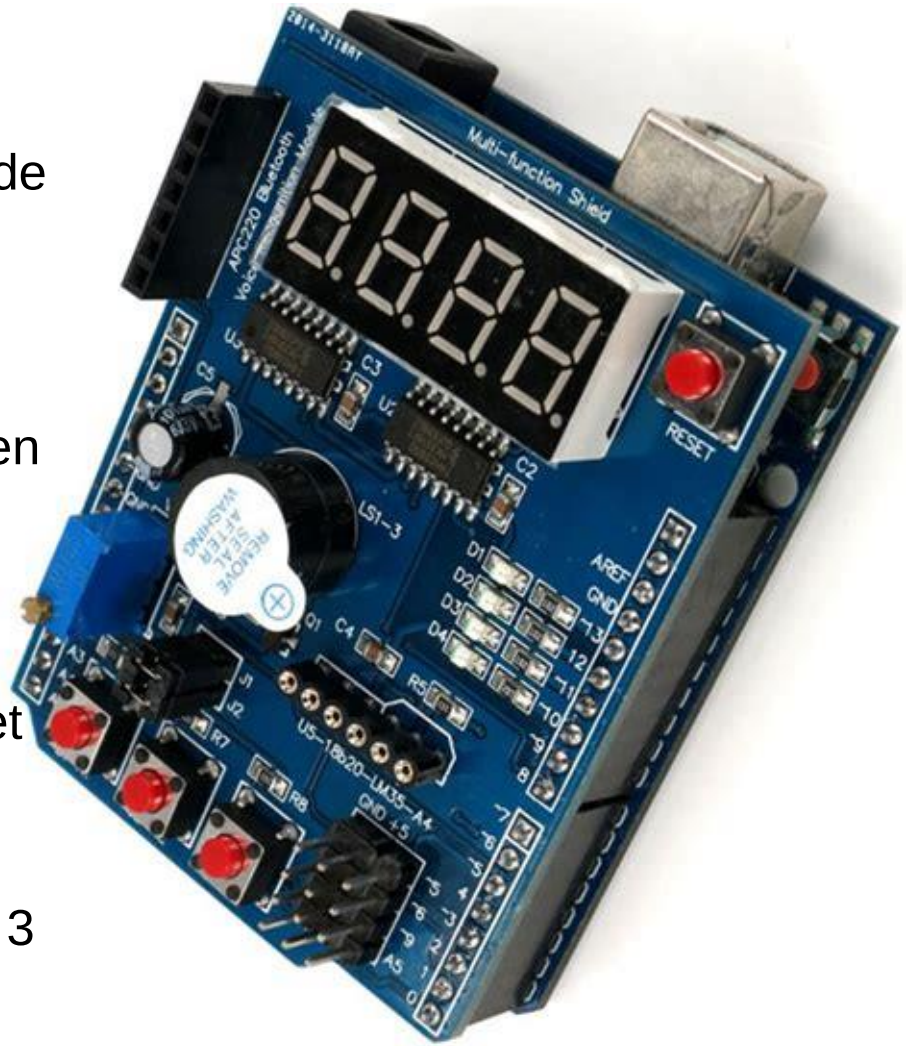
Werken met interrupts en libraries

Week 16 (Les 29 en 30)

DE HAAGSE HOGESCHOOL

Werken met interrupts

- Vorige week zijn we begonnen met de aansturing van de 4 digit 7 segmentsdisplays.
- De besturing ging toen “pollend”, dus iedere keer in de hoofdloop de 4 digits aansturen.
- Nadeel: zodra je iets anders wil doen (al is het maar een delay(1); krijgt de laatste digit een hogere helderheid, omdat die aangestuurd blijft.
- Dit wordt alleen maar erger als het hoofdprogramma langer duurt (probeer maar delay(30) waarna alleen het vierde digit nog zichtbaar is).
- Het mooiste zou zijn om regelmatig (=periodiek) ieder digit op volgorde aan te sturen (Digit 1 > Digit 2 > Digit 3 > Digit 4 > Digit 1 > Digit 2….)
- En als de frequentie groter is dan 30 Hz zien we geen flikkering.
- Oplossing: interrupts



DE HAAGSE HOGESCHOOL

Interrupts

- Interrupts zijn handig omdat ze helpen om timing problemen op te lossen;
  - Je hoeft dan niet constant te kijken of er iets in de omgeving verandert.
  - De processor stopt (tijdelijk) met waar hij mee bezig is en gaat een “interrupt service routine” uitvoeren.
- Er zijn drie soorten interrupts:
  1. Hardware interrupts (worden vanuit hardware geactiveerd)
  2. Software interrupts (worden vanuit software geactiveerd)
  3. Interne (synchrone) interrupts (veroorzaakt door een verandering of verstoring in de uitvoering van een programma, zoals ongeldig adres)
- Wij maken onderscheid in 2 soorten Hardware interrupts:
  - I/O interrupts (een ingang verandert van waarde)
  - Timer interrupts (een ingestelde tijd is verlopen)

I/O interrupts

- De Arduino UNO heeft slechts 2 interrupts beschikbaar om de processor te interumperen als een ingangswaarde van een I/O pen veranderd:
  - Interrupt 0: verbonden met pin 2
  - Interrupt 1: verbonden met pin 3
- Arduino heeft hier aparte instructies voor:
  - `attachInterrupt(interrupt, function, mode);`
  - Met:
  - `Interrupt:` 0 of 1 (pin 2 of pin 3) maar beter om `digitalPinToInterrupt(pin)` te gebruiken.
  - `function:` de naam van de aan te roepen functie als interrupt is geconstateerd. (Let op: geen input- of return-parameters, dus `void function();`)
  - `Mode`: trigger voorwaarde:
    - **`LOW`** : actief als ingang logisch 0 is.
    - **`CHANGE`** : actief als ingang van logische waarde verandert
    - **`RISING`** : actief als ingang verandert van logische 0 naar 1 (positieve flank)
    - **`FALLING`** : actief als ingang verandert van logische 1 naar 0 (negatieve flank)

I/O interrupts waarschuwingen

- De uitvoering van de functie die uitgevoerd wordt wanneer een interrupt geactiveerd is, noemt men een Interrupt Service Routine (ISR).
- Deze ISR moet zo kort mogelijk gehouden worden, omdat anders andere interrupts worden tegengehouden en daardoor “gemist” worden.
- Gebruik in de ISR nooit een **`delay()`** omdat deze intern ook gebruik maakt van interrupts en timers en daardoor de ISR blokkeert.
- De compiler ziet niet of een variabele in verschillende delen wordt gebruikt, waardoor een (globale) variabele door een ISR opeens van waarde kan veranderen.
  - Maak variabelen binnen een ISR altijd `volatile` (bijv. **`volatile int`** `i`) zodat het geheugen en de waarde bewaard blijft en gereserveerd voor die ISR.
- Gebruik maximaal 6-8 interrupts in een sketch. (complexiteit een foutkans groot)

Rotary encoder met interrupts setup()

```
1  #define encoder0PinA 2     // voor interrupt 0
2  #define encoder0PinB 3     // voor interrupt 1 nu niet gebruikt
3  #define encoder0Btn 4
4  volatile int encoder0Pos = 0; // volatile in deze sketch niet belangrijk
5
6  void setup() {
7    Serial.begin(9600);
8    pinMode(encoder0PinA, INPUT);
9    pinMode(encoder0PinB, INPUT);
10   pinMode(encoder0Btn, INPUT_PULLUP); // aan SW hangt geen pull-up
11   attachInterrupt(digitalPinToInterrupt(encoder0PinA), doEncoder, CHANGE);
12 }
13
```

UNO alleen 2 of 3

ISR

Rotary encoder ISR doEncoder

```
33 void doEncoder()
34 {
35   if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB))
36   {
37     encoder0Pos++;
38   }
39   else
40   {
41     encoder0Pos--;
42   }
43   valRotary = encoder0Pos / 2.5;
44 }
```


Doordat ISR wordt aangeroepen
bij een CHANGE van A

Rotary encoder ISR loop()

```
14 int valRotary, lastValRotary;
15 void loop() {
16   int btn = digitalRead(encoder0Btn);
17   Serial.print("btn= ");
18   Serial.print(btn);
19   Serial.print(" ");
20   Serial.print(valRotary);
21   if (valRotary > lastValRotary)
22   {
23     Serial.print("  CW");
24   }
25   if (valRotary)  {
26     Serial.print("  CCW");
27   }
28   lastValRotary = valRotary;
29   Serial.println(" ");
30   delay(250);
31 }
```

Demo

Interrupts

- Interrupts zijn handig omdat ze helpen om timing problemen op te lossen;
  - Je hoeft dan niet constant te kijken of er iets in de omgeving verandert.
  - De processor stopt (tijdelijk) met waar hij mee bezig is en gaat een “interrupt handler” uitvoeren.
- Er zijn drie soorten interrupts:
  1. Hardware interrupts (worden vanuit hardware geactiveerd)
  2. Software interrupts (worden vanuit software geactiveerd)
  3. Interne (synchrone) interrupts (veroorzaakt door een verandering of verstoring in de uitvoering van een programma, zoals ongeldig adres)
- Wij maken onderscheid in 2 soorten hardware interrupts:
  - I/O interrupts (een ingang verandert van waarde)
  - Timer interrupts (een ingestelde tijd is verlopen)

Registers in ATmega 382 voor timers

- De ATmega382 heeft 3 Timer/Counters:
  - Timer/Counter 0 : 8 bits (TC0)
  - Timer/Counter 1 : 16 bits (TC1)
  - Timer/Counter 2 : 8 bits (TC2)
- TC0 wordt intern gebruikt voor de delay() en millis() functie en gebruiken we daarom niet in onze sketches
- De timer/counters kunnen in veel verschillende modes worden gezet (onder andere om timers, counters maar ook PWM te maken)
- Die modes kunnen worden gezet door de juiste bitjes te setten of resetten in REGISTERS.
- Voor Timer/Counter1 zijn dat de registers TCCR1A en TCCR1B (Timer/Counter Control Register 1)
- We gaan niet alle instellingen bekijken maar zoeken naar de juiste mode om periodiek een interrupt te genereren zodat we de digits van het display één voor één kunnen aansturen.

Registers in ATmega 382 voor timers

COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	TCCR1A
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Input capture			Timer Mode		Timer1 clock selector			

CS12	CS11	CS10	Timer1 clock selector
0	0	0	No clock source (T/C stopped)
0	0	1	clk (16 MHz)
0	1	0	clk / 8 (2 MHz)
0	1	1	clk / 64 (250 kHz)
1	0	0	clk / 256 (62,5 kHz)
1	0	1	clk / 1024 (15,625 kHz)
1	1	0	External clock T1 falling edge
1	1	1	External clock T1 rising edge

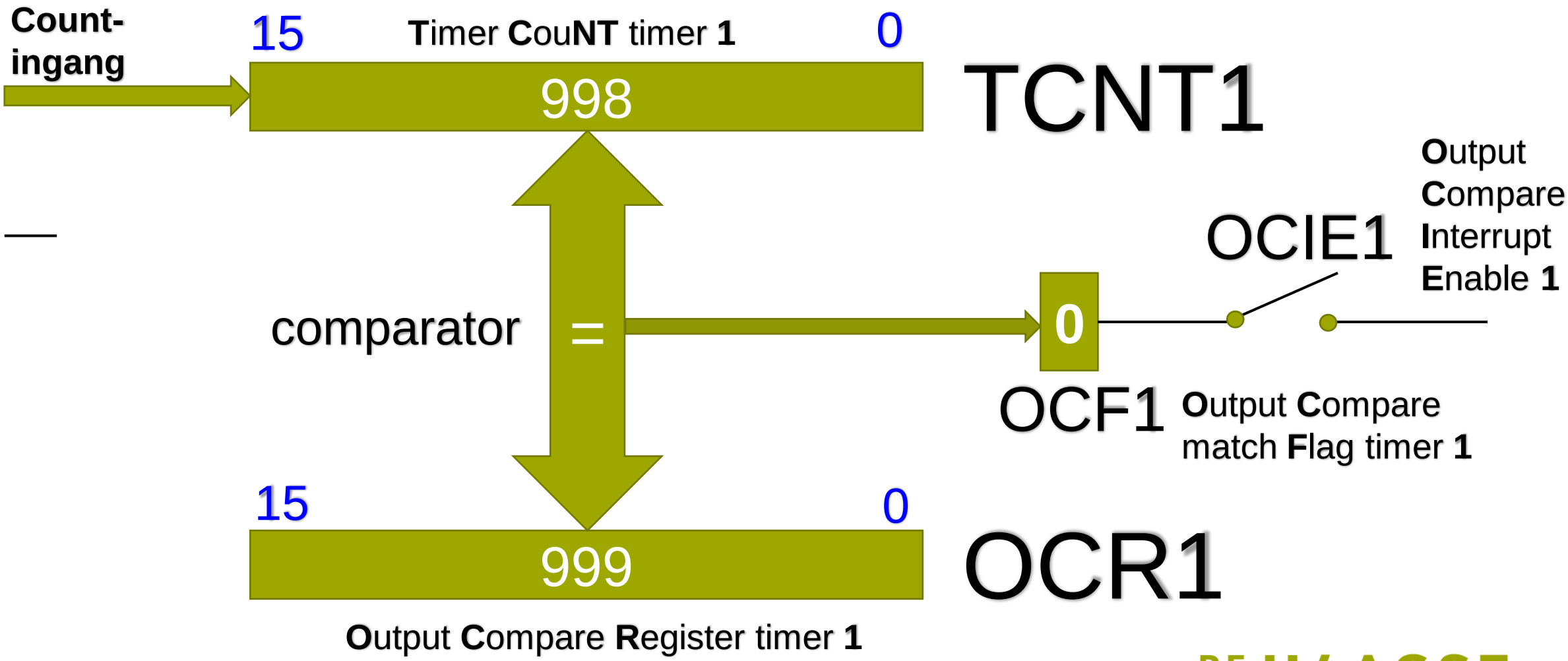
CTC-mode

TCCR1B TCCR1A

WGM13	WGM12	WGM11	WGM10	Mode	Top
0	1	0	0	CTC	OCR1A

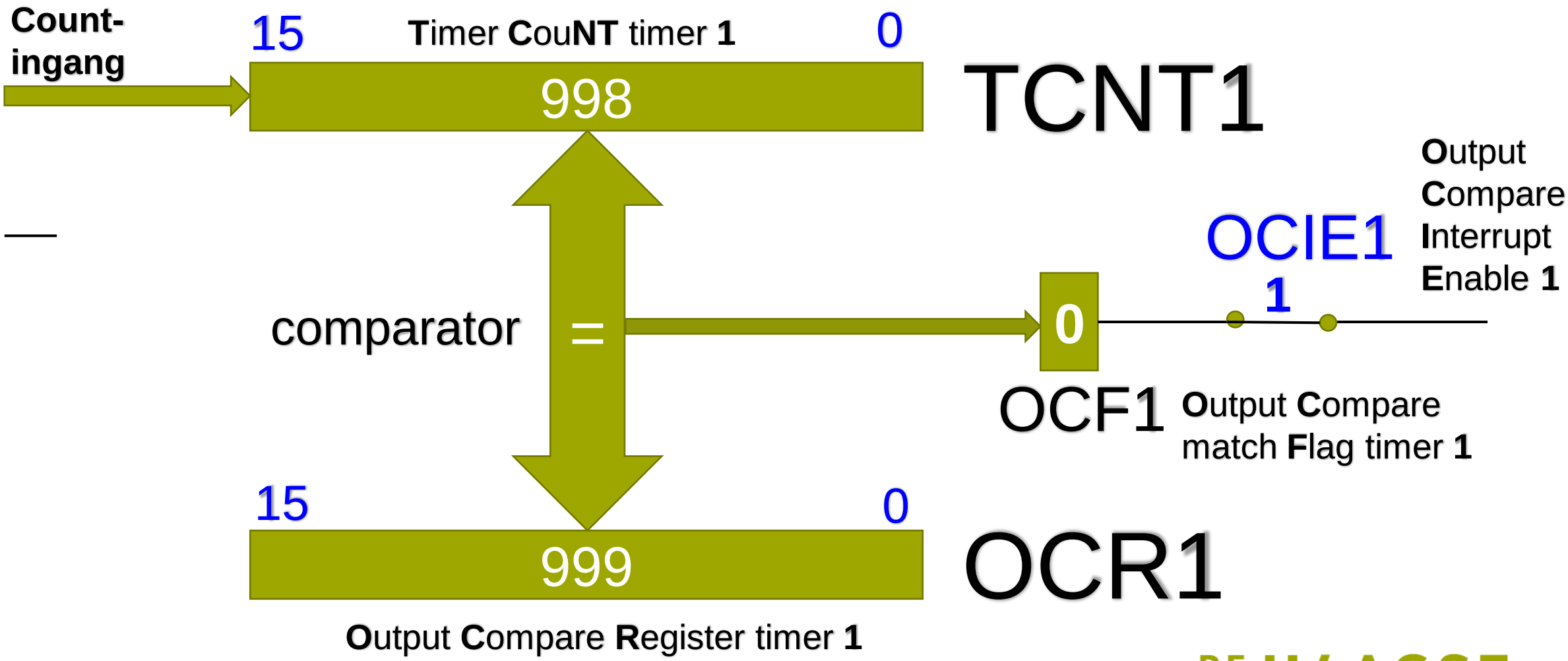
DE HAAGSE HOGESCHOOL

Timer interrupts ATmega382 timer 1



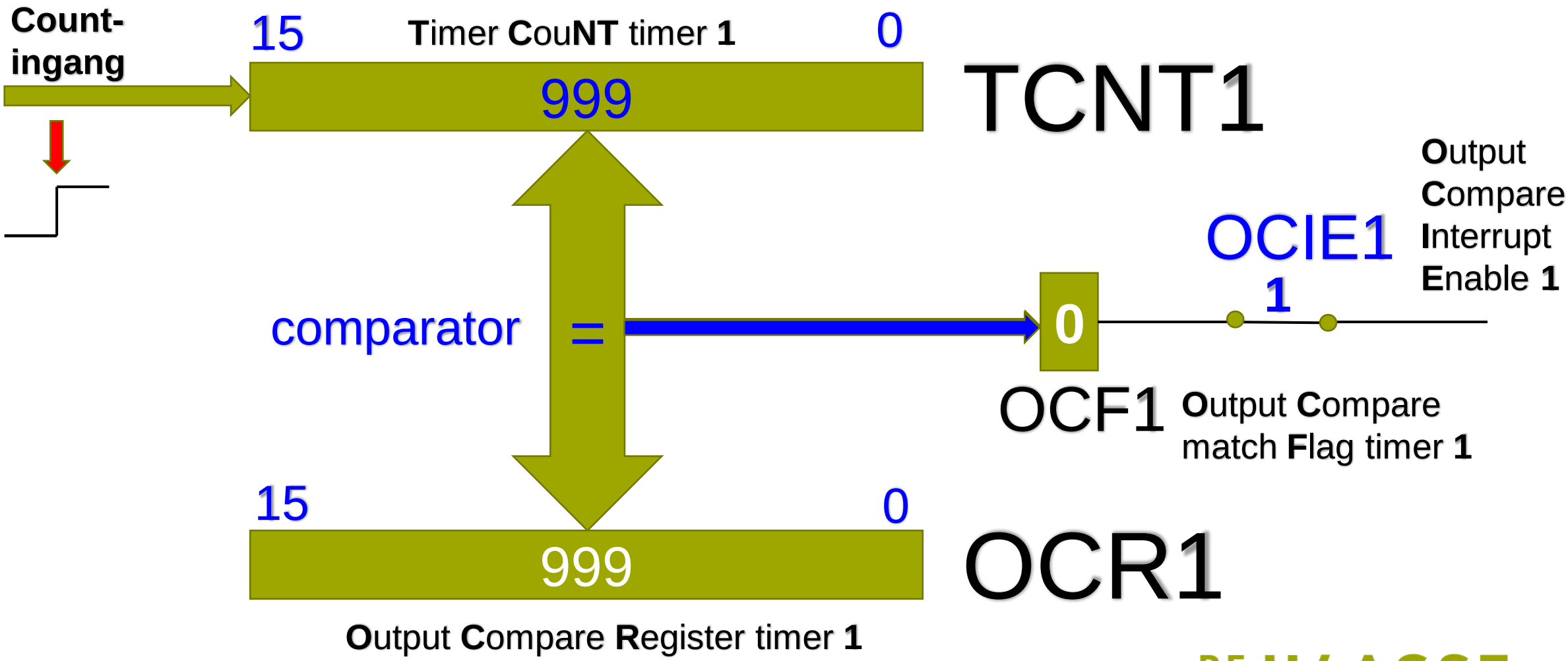
DE HAAGSE HOGESCHOOL

Timer interrupts ATmega382 timer 1



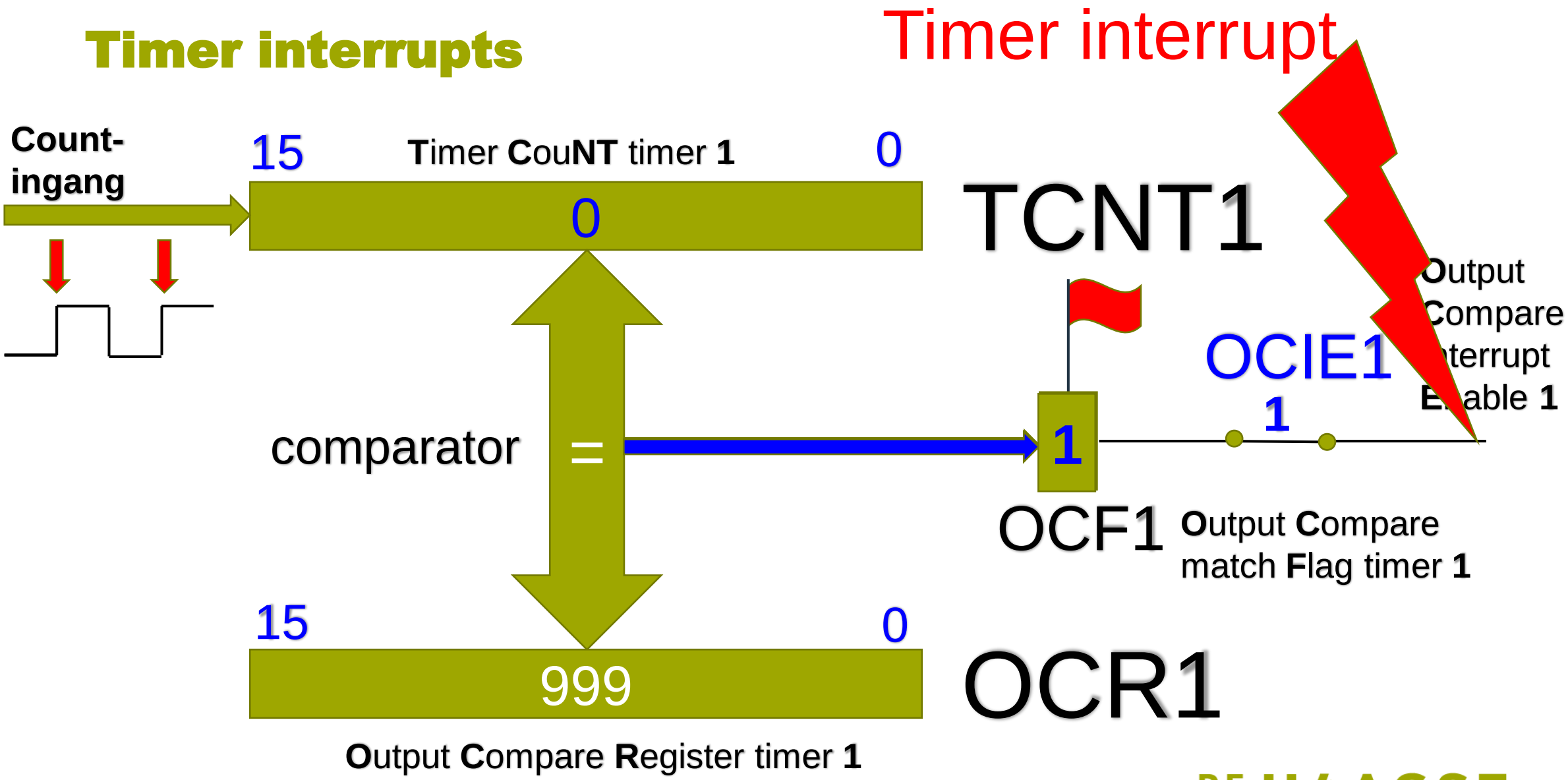
DE HAAGSE HOGESCHOOL

Timer interrupts



DE HAAGSE HOGESCHOOL

Timer interrupts



DE HAAGSE HOGESCHOOL

Display MFS aansturing met interrupts setup()

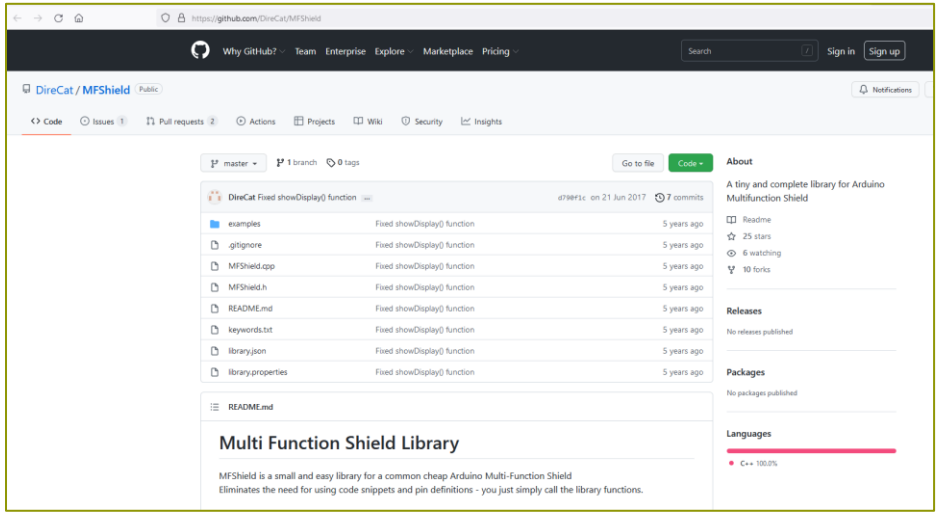
```
80 void setup()
81 {
82   pinMode( DATA_IO, OUTPUT);
83   pinMode( CLK_IO, OUTPUT);
84   pinMode( LATCH_IO, OUTPUT);
85   pinMode(LED, OUTPUT);
86   // setup Timer 1 (clock speed is 16 MHz
     TCCR1A = 0;        // normal operation
     TCCR1B = bit(WGM12) | bit(CS10) |bit(CS12);  //CTC, Compare TimerCounter en prescaler clock 1024
     OCR1A = 999;       // compare A register value (1000 * clock speed / 1024) (15,6 / 4 =  4 Hz)
90 //  OCR1A = 49;       // compare A register value (50 * clock speed / 1024)(312,5/4 = 78 Hz)
     TIMSK1 = bit (OCIE1A);  // interrupt on Compare A Match
92 }
```

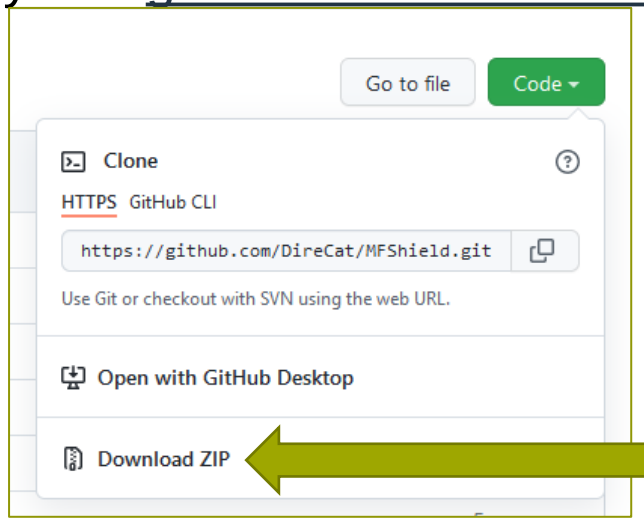
Demo

DE HAAGSE HOGESCHOOL

Libraries

- Aansturen 7 segmentsdisplay nu prachtig met interrupts gerealiseerd, maar nog niet alle andere “functions” van het MultiFunctionShield gebruikt.
- Daarom gaan we nu een library gebruiken, zodat we een heleboel functies eenvoudig kunnen gebruiken.
- Doel library: veel gebruikte functies hergebruiken door een library op te nemen in een sketch dmv **`#include <libraryName.h>`**.
- In deze presentatie gaan we alleen de library voor het MFShield installeren. Voor andere componenten is de werkwijze hetzelfde.
- Download de MultifunctionShield Library van github.com/DireCat/MFShield.git

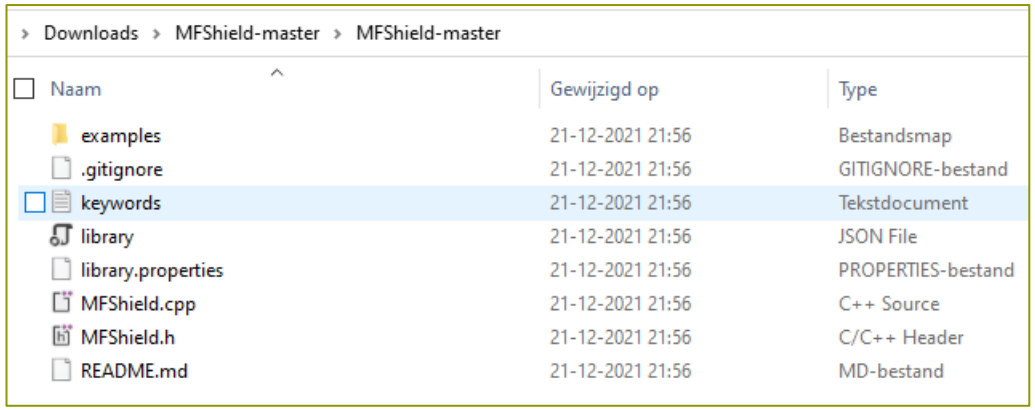


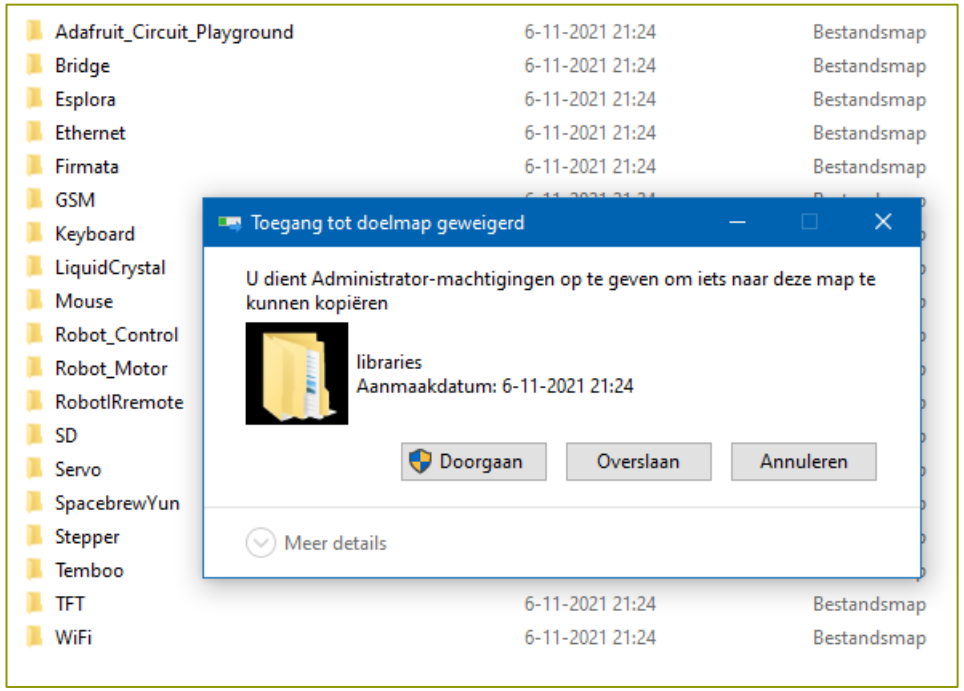


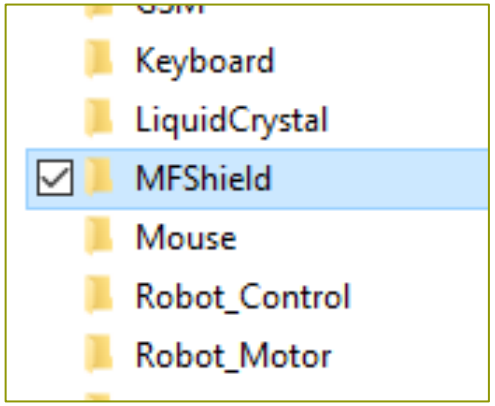
DE HAAGSE HOGESCHOOL

MFShield Library

- Ga naar de map waar het .zip bestand is opgeslagen (meestal Download).
- Unzip het bestand daar en ga dieper totdat je bestanden ziet staan.
- Ga dan 1 map naar boven en verander de mapnaam naar MFShield (dus –master eraf halen).
- Knip die hele map en ga naar Program Files (x86) > Arduino > libraries.
- Plak daar de map en klik op Doorgaan als administrator.
- Dan moet de map er als volgt uitzien….







Installatie Geslaagd

DE HAAGSE HOGESCHOOL

MFShield Library gebruik

- Altijd bovenin sketch: `#include <MFShield.h>`
- Én een object aanmaken van de klasse MFShield: `MFShield mfs;`
- Én in de loop altijd opnemen: `mfs.loop()`.
- Nu zijn de volgende handige functies beschikbaar:
  - `onKeyPress ( FunctieNaam (uint8_t toetsNummer));`
    - Start een functie als een toets S1, S2 of S3 is ingedrukt. *(toetsNummer 0..2)*
  - `display (int waarde)`
    - Laat een getal op het 7 segmentsdisplay zien (werkt intern met interrupt).
  - `int readTrimmerValue();`
    - Leest de analoge waarde van de meerslagenpotmeter (0 – 1023).
  - `beep (int tijd);`
    - Laat de buzzer geluid maken gedurende `tijd` in ms.
  - `setLed (uint8_t ledNummer, uint8_t aanUit);` *(ledNummer 0..3)*
  - `uint8_t getLed(uint8_t ledNummer);`

  - WAARSCHUWING:
  - Gebruik ook hier NOOIT `delay()`,omdat anders het display niet meer wordt geupdate en de toetsen niet meer worden gescanned (door interrupts)

Voorbeeld MFSdemLib

MFSdemoLib

```
#include <MFShield.h>

MFShield mfs;

void setup ()
{
   Serial.begin (9600);
   /* Handle buttons */
   mfs.onKeyPress ([](uint8_t button) {
       mfs.beep(10);     // make sound
       mfs.setLed (button, !mfs.getLed(button)); // toggle onboard leds
       Serial.println ("Button pressed: " + String(button));
   });
}

void loop ()
{
   /* Display seconds */
   mfs.display (millis() / 1000);

   mfs.loop();
   /* ^ It's important to insert this loop function in the main loop
    *   else numeric display wont work
    */
}
```

Demo

DE HAAGSE HOGESCHOOL

Voorbeeld ShieldDemo setup()

ShieldDemo §

```
#include <MFShield.h>

/* Create an object which is needed by library to control the shield */
  MFShield mfs;

/* This function is called every time any button is pressed */
/* The only argument means the number of the button pressed */
void showKey (uint8_t key)
{
  // print out the button number
  Serial.println("Button pressed: " + String (key, DEC));

  // turn on and off a led which has the number equal to the button's number
  mfs.setLed (key, !mfs.getLed (key));

  // make a short beep for 5 ms (because it's too loud)!
  mfs.beep (5);
}

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

  // Assign the custom function 'showKey' (see above) to the button press event
  // Note: this function must have one argument (8-bit variable) which defines
  // the button number is being pressed
  mfs.onKeyPress (showKey);
}
```

Voorbeeld ShieldDemo loop()

```
30 void loop()
31 {
32   // Always insert mfs.loop() in the main loop, without this function
33   // the MFShield library wont work. It's a neccessary routine needed
34   // to update the display, poll the buttons and run the internal timer.
35   mfs.loop();
36
37   static uint32_t t = millis();
38   // Run this cycle once every 200 msec
39   if (millis() - t >= 200)
40   {
41     t = millis();
42     // Shows the potentiometer (trimmer) value on it's numeric display
43     uint16_t trimmer = mfs.getTrimmerValue();
44     mfs.display(trimmer);
45   }
46 }
```

Demo

let’s change