

Handleiding Freesbank

Deze handleiding is voor het gebruik en aansluiten van de freesbank bij Sens2sea.

Aansluiten

Het systeem bestaat uit de volgende hardware:

- Stepper motoren (2 voor X, 1 voor Y, 1 voor Z).
- Handfrees (werkt op netstroom).
- Encoders.
- Arduino mega microcontroller.
- Freesbank.
- CNCshield.
- 4x DRV8824 stepper motor drivers.

Let op: Om deze freesbank aan te zetten is er ook een labvoeding nodig. Hiernaast kan het systeem ook werken met 3 motoren.

Sluit het systeem aan zoals aangegeven op het elektrische schema. Omdat er een CNCShield gebruikt wordt kan de hoge voltage naar de CNCShield en zijn er geen losse condensatoren nodig. Verder moet de handfrees op het netstroom en handmatig aangezet worden.

Software.

De software (te vinden onder code) kan worden geupload op de arduino mega. Hierna kunnen de commando's die ondersteund worden uitgevoerd worden.

Ondersteunde commando's

Systeem commandos

\$?: Krijg de huidige informatie. (state, huidige locatie, eindpunt beweging, beginpunt beweging).

G-Code

Snelle rechte beweging (G0) Dit commando laat de machine snel naar de gekozen locatie gaan in een rechte lijn. Hierbij is de tool niet actief.

Parameters:

X: X positie in mm

Y: Y positie in mm

Z: Z positie in mm

Snelle rechte beweging (G1) Dit commando laat de machine snel naar de gekozen locatie gaan in een rechte lijn. Hierbij is de tool wel actief.

Parameters:

X: X positie in mm

Y: Y positie in mm
Z: Z positie in mm
F: Feedrate in mm/min (niet geïmplementeerd.)

Clockwise cirkel beweging (G2) Dit commando laat de machine bewegen met de klok mee. Hierbij is de tool actief.

Parameters: X: X positie in mm
Y: Y positie in mm
Z: Z positie in mm
I: X middelpunt van cirkel in mm J: Y middelpunt van cirkel in mm R: Radius in mm

Anticlockwise cirkel beweging (G3) Dit commando laat de machine bewegen tegen de klok in. Hierbij is de tool actief.

Parameters: X: X positie in mm
Y: Y positie in mm
Z: Z positie in mm
I: X middelpunt van cirkel in mm J: Y middelpunt van cirkel in mm R: Radius in mm

Plane selectie (G17 G18 G19) Dit commando laat de gebruiker de plane kiezen op welke plane de machine cirkels maakt.

G17: XY G18: YZ G19: ZX

Mode selectie (G90 G91) Dit commando laat de gebruiker kiezen of de machine werkt in Incrementeel of absolute bewegingen.

G90: Absolute G91: Incrementeel

M-code

Stop stop direct (M1) dit commando zet de state van de machine op error. Hierdoor gaat de machine niet verder. (restart nodig)

Dev guide.

Hieronder zijn verschillende onderdelen die handig zijn voor developers van het systeem.

- Global variables.
- Datatypes.
- Interrupt timers.
- Assen veranderen / toevoegen
- Bitwise poorten aansturen.

Globals.

Er zijn verschillende macro's en globals om de code leesbaarder te maken. Macro's zijn volledig hoofdletter zoals **IDLE** en globals zijn als variables **systemState**.

systemState Deze variable houd de statemachine bij. Hier hangen devolgende waardes aan.

- IDLE (0) -> De machine wacht op instructies
- RUNNING (1) -> De machine is bezig met commando's uit te voeren.
- ERROR (2) -> Error in het systeem. Kan worden op gelost door het systeem
- INTERNAL_ERROR_RESTART_REQUIRED (3) -> Een error die het systeem zelf niet kan verhelpen.
Hier voor is ook een restart nodig

stepperState Stepperstate houd de status van de stepper motoren bij. Deze hebben 1 plek in een buffer waarin een segment gezet word.

- STEPPER_FULL (0) -> De stepperbuffer heeft inhoud. Deze wordt uitgevoerd.
- STEPPER_EMPTY (1) -> De stepperbuffer is leeg. Indien deze niet wordt aangevuld zal het systeem naar IDLE gaan.
- STEPPER_ERROR (2) -> Er is een fout opgetreden. Word gefixt door main loop en anders wordt de systemstate INTERNAL_ERROR_RESTART_REQUIRED.

Macro's

Er zijn verschillende macro's die gebruikt worden. De macros die gebruikt worden in de GLOBALS worden hier niet behandeld.

Simple macro's:

- **BAUD**: Baud rate van de serial monitor. Alleen **9600** is mogelijk/getest.
- **MAX_LINE_SIZE**: De maximale lengte van een lijn met seriële input dat gelezen kan worden uit de buffer.
- **PI**: Arduino's PI implementatie voor het getal van PI.
- **OK_MESSAGE**: Het bericht dat terug gestuurd wordt na elke G-Code opdracht. Standaard **OK**.
- **ENCODERS_AXIS**: Aantal axis met encoders. Standaard op **3**.
- **STEPPER_ACCURACY**: de nauwkeurigheid van de stappen motoren.
- **STEPPER_ISR_MS**: Aantal milliseconde tussen de interrupts.
- **STEPPER_ISR_US**: Aantal microseconde tussen de stap interrupt en de stap reset interrupt.
- **X_AXIS**: Nummer van de X as in vectoren.
- **Y_AXIS**: Nummer van de Y as in vectoren.
- **Z_AXIS**: Nummer van de Z as in vectoren.

Pinnen:

De pinnen worden met de registers aangestuurd en hieronder zijn alle register nummers te vinden voor de actuatoren en sensoren.

Stapper pinnen:

- **X_STEP_PIN**: De stap pin van de X as.
- **X_DIR_PIN**: De directie pin van de X as.
- **Y_STEP_PIN**: De stap pin van de Y as.
- **Y_DIR_PIN**: De directie pin van de Y as.
- **Z_STEP_PIN**: De stap pin van de Z as.
- **Z_DIR_PIN**: De directie pin van de Z as.

Stapper registers:

- **STEPPER_PORT**: Register om de pin hoog of laag te maken.
- **STEPPER_DDR**: Register om de pin in of output te maken.

Encoder pinnen:

- **ENCODER_PIN_XA**: De A pin van de X encoder.
- **ENCODER_PIN_XB**: De B pin van de X encoder.
- **ENCODER_PIN_YA**: De A pin van de Y encoder.
- **ENCODER_PIN_YB**: De B pin van de Y encoder.
- **ENCODER_PIN_ZA**: De A pin van de Z encoder.
- **ENCODER_PIN_ZB**: De B pin van de Z encoder.

Encoder registers:

- **ENCODER_DDR_XY**: Register om de pin in of output te maken.
- **ENCODER_PORT_XY**: Register om de pin hoog of laag te maken. (Hoog bij input is pullup)
- **ENCODER_DDR_Z**: Register om de pin in of output te maken.
- **ENCODER_PORT_Z**: Register om de pin hoog of laag te maken. (Hoog bij input is pullup)
- **ENCODER_PIN_XY**: Register om de output te lezen van de pin.
- **ENCODER_PIN_Z**: Register om de output te lezen van de pin.

Datatypes

Omdat de arduino zijn grote van een integer kleiner is dan die van andere computers zijn de standaard 16, 32 en 64 bit integers uit [`<iostream>`](#) niet correct. Hierom zijn er custom macro's.

iostream	grote (bit)	expanded	custom	grote (bit)	expanded
-	-	-	uchar	8	unsigned char
uint8_t	8	unsigned char	uint8	8	unsigned char
int8_t	8	char	int8	8	char
uint16_t	16	unsigned short	uint16	16	unsigned int
int16_t	16	short	int16	16	int
uint32_t	32	unsigned int	uint32	32	unsigned long int
int32_t	32	int	int32	32	long int
uint64_t	64	unsigned long long	-	-	-
int64_t	64	long long	-	-	-

Timers

In dit programma zitten verschillende interrupts. Hieronder zijn Timers & hardware interrupts. Timer interrupts hebben de hoogste prioriteit in een arduino. Hierna komen de externe hardware pin interrupts.

Hardware interrupts

De implementatie is te vinden in de code, maar eerst worden alle pinnen op input (DDR) en pullup (PORT) gezet. Hierna wordt de modus op **any change (falling en rising edge)** gezet. Als laatste worden ze geactiveerd.

Internal timer compare interrupts

Een arduino heeft meerdere timer compare interrupts die te gebruiken zijn. Een uno heeft Timer0 (8 bits), Timer1(16 bits) en Timer2 (8 bits). De Timer0 wordt ook gebruikt voor pwm outputs, dus veranderen van de configuration zal de pwm channels veranderen. Hiernaast wordt timer0 gebruikt voor de delay, millis en micros. Hier de configuraties van veranderen zal dit veranderen. De implementatie is te vinden in de functie **setHardwareCompareTimer** in **stepper.cpp**. Voor meer informatie zijn hier twee bronnen.

- <https://deepbluemediated.com/arduino-timers/>
- <https://deepbluemediated.com/arduino-timer-calculator-code-generator/>

Assen toevoegen.

Om assen toe te voegen en aan te passen zijn twee plekken die veranderd moeten worden in devolgende functie in stepper.cpp:

```
/**  
 * Main ISR, runs at intervalls defined in STEPPER_ISR_MS in macros.h.  
 * blocks a new interrupt if it fires before the previous interrupt is handled.  
 * This ISR does all the movements for the steppers and activate the reset ISR.  
 */  
ISR(TIMER1_COMPA_vect) {  
    /* variable to prevent reentering the loop until it is finished */  
    static bool isBusy = false;  
  
    if(isBusy) return; /* return if previous cycle is not done */  
  
    isBusy = true; /* set busy */  
    float currentPosition[ENCODERS_AXIS]{};  
    getCurrentMMFromEncoders(currentPosition); /* get current encoder values */  
  
    /* check if the timer has reached and check if the axis are at the right place.  
 */  
    /* @note maybe add a -4 of time and slowly move to the target while counting  
    down and checking? */  
    if(stepperData.timerValue >= stepperData.modifier) {  
        uint8 donecheck = 0;  
  
        ****  
        ****  
        * @note add here axis that are used by the stepper motor. (and have an  
        encoder) @attention (Place 1)  
        ****  
        ****  
        ****/  
}
```

```

//donecheck += checkIfAxisNotDone(Z_AXIS, currentPosition[Z_AXIS]);

/*
*****
* END for adding more steppers.

*****
*/
if(donecheck == 0) {
    println(currentPosition);
    setSegmentDone();
    isBusy = false;
    return;
}
}

/*
*
* @note add here axis to be used by the stepper motor. @attention (Place 2)
* @note that they need to be added by @attention (Place 1) if encoder is used.
This to check if they are done.

*****
*/
stepAxisFromStepVar(X_AXIS, stepperData.timerValue, stepperData.modifier,
&STEPPER_STEP_PORT, X_STEP_PIN, &STEPPER_DIR_PORT, X_DIR_PIN);
stepAxisFromStepVar(Y_AXIS, stepperData.timerValue, stepperData.modifier,
&STEPPER_STEP_PORT, Y_STEP_PIN, &STEPPER_DIR_PORT, Y_DIR_PIN);
stepAxisFromStepVar(Z_AXIS, stepperData.timerValue, stepperData.modifier,
&STEPPER_STEP_PORT, Z_STEP_PIN, &STEPPER_DIR_PORT, Z_DIR_PIN);
//stepAxisFromPos(Z_AXIS, stepperData.timerValue, stepperData.modifier,
currentPosition[Z_AXIS], &STEPPER_STEP_PORT, Z_STEP_PIN);

/*
*****
* END for adding more steppers.

*****
*/
stepperData.timerValue >= stepperData.modifier ? stepperData.timerValue =
stepperData.modifier : stepperData.timerValue++;

/** set reset isr */
TCNT1 = 0;           /* Set timer to 0 */
TIMSK1 |= (1 << OCIE1B); /* enable Reset ISR */
isBusy = false;
}

```

De eerste plek is onder de `@attention (Place 1)`. Hier is de variabele donecheck die opgehoogd moet worden als de as nog niet klaar is als de tijd verstrekken is. Hiervoor is de functie `checkIfAxisNotDone` die 1 teruggeeft als de as nog niet klaar is. Deze check is alleen nodig als er een encoder op de as gebruikt wordt. Indien er geen encoder op de as gebruikt wordt, moet de functie weggehaald worden.

De tweede plek is onder de `@attention (Place 2)`. Hier moeten de assen toegevoegd worden met de functies om een stap te zetten. Hiervoor zijn `stepAxisFromStepVar` en `stepAxisFromPos`. `stepAxisFromPos` heeft een waarde nodig van een sensor en heeft geen implementatie voor de cirkel beweging. Hiernaast telt `stepAxisFromStepVar` de stappen die gezet worden. Deze functie heeft wel de implementatie voor een cirkel beweging.

Poorten aansturen.

Hierbij een kleine tutorial met pinnen aansturen. Dit is onder te verdelen in in- en output. Iedere pin heeft zijn eigen register en nummer op dat register. Voor deze uitleg gaan we uit van pin 5 van een Arduino Uno, maar dit is toe te passen op alle andere pinnen van de Arduino Uno en ook andere microcontroller zoals de Arduino Mega. Pin 5 zit op register D en is nummer 5. Hierbij horen PORTD, PIND, DDRD.

Er zijn verschillende operators die gebruikt worden. Deze zijn: `& (bitwise and)`, `| (bitwise or)`, `^ (bitwise XOR)` en `~ (bitwise not)`.

Input

Om de input te lezen van een pin moet de pin op het DDR register laag gezet worden.

`DDRD &= ~(1 << PD5)`. Hierin is DDRD het register, doen we een `and` operatie op de 5e pin van het register. Hiervoor shiften we de `1` naar de juiste positie (5). Dit werkt als volgt als we pin 5 willen aanpassen als het register als volgt staat `01101010`:

```
00100000 // (shift `1` 5 naar links(`1 << PD5`) op een leeg register).
Hierna flippen we deze met de `~` operator.
11011111

// Hierna gebruiken we een `and` operatie op het register.
01101010 (&) (DDRD &= (11011111))
11011111
-----
01001010
```

Hierna willen we de pullup aanzetten op de pin. hiervoor gebruiken we een `or` operatie.

`PORTD |= (1 << PD5)`. Hierin is PORTD het register. Als deze hoog is staat de pullup aan, anders is deze uit. We gaan uit van het register in de volgende staat: `01000000`.

```
00100000 // (shift `1` 5 naar links(`1 << PD5`) op een leeg register).

// Hierna gebruiken we een `or` operatie op het register.
01000000 (&) (PORTD |= (00100000))
```

```
00100000
```

```
-----
```

```
01100000
```

Om de pinnen uit te lezen wordt het register PIND gebruikt. We kunnen checken of een pin hoog is door `if(PIND & (1 << PD5))`. We gaan uit dat het register zich bevindt in deze staat: **10101110**.

```
00100000 // (shift `1` 5 naar links(`1 << PD5`) op een leeg register).
```

```
10101110 (&) (PIND & (00100000))
```

```
00100000
```

```
-----
```

```
00100000 -> hoog
```

```
indien het register 10000110 is:
```

```
10000110 (&) (PIND & (00100000))
```

```
00100000
```

```
-----
```

```
00000000 -> laag
```

Output

Het zetten van een pin kan met dezelfde operaties als hiervoor.

- Pin als output: `DDRD |= (1 << PD5)`
- Pin hoog: `PORTD |= (1 << PD5)`
- Pin laag: `PORTD &= ~(1 << PD5)`
- Pin toggle: `PORTD ^= (1 < PD5)`

Toggle werkt alleen op Arduinos, niet op esp32's. Het toggelen gaat als volgt met een XOR: register is: **01001011**.

```
01001011 (^)
```

```
00100000
```

```
-----
```

```
01101011
```

```
01101011 (^)
```

```
00100000
```

```
-----
```

```
01001011
```