



Biegemaschinen-Steuerung

ESP32 Firmware-Dokumentation

© Copyright by Frank Matthiensen (FM-MicroTech) 2026, All Rights Reserved

<http://www.fm-microtech.de>

Firmware v1.2.0

Biegemaschinen-Steuerung (ESP32)

Firmware für eine umgebaute Biegemaschine. Befehle werden per serieller Schnittstelle vom PC empfangen, ausgeführt und quittiert. Die Steuerung läuft auf einem **ESP32-DevKitC (38 Pin)** mit PlatformIO (Arduino-Framework).

Inhalt

- [Überblick](#)
- [Projektstruktur](#)
- [Hardware](#)
- [Pinbelegung](#)
- [Wichtige ESP32-Hinweise](#)
- [Seriellles Protokoll](#)
- [Befehle](#)
- [Ablauf der Befehle](#)
- [Maschinenlogik](#)
- [Konfiguration \(`config.h` \)](#)
- [Build und Upload](#)
- [Fehlersuche](#)

Überblick

Eigenschaft	Wert
Mikrocontroller	ESP32-DevKitC (38 Pin)
Baudrate	115200
Framework	PlatformIO + Arduino
Firmware	v1.2.0
Konfigurationsdatei	<code>include/config.h</code>

Die Steuerung verwaltet:

- Schrittmotor (Vorschub) mit Endschaltern
- Magnetventile (Heben, Senken/Biegen)
- Halteklammer, Auswerfer, Hydraulikpumpe

- Winkel- und Höhen-Encoder per Software-Quadratur (Polling, 4x-Auflösung)
- Jog-Encoder (Polling) fuer manuelles Verschieben
- Watchdog, Not-Aus, Bewegungs-Timeout, Hydraulik-Auto-Aus

Projektstruktur

```

bmfinal/
├── include/
│   ├── config.h           # Pins, Kalibrierung, Timing
│   ├── config_commands.h  # #HELP, #DEBUG, ...
│   ├── debug.h           # Debug-Ausgabe (DEBUG_LOG Makros)
│   ├── help.h            # #HELP Ausgabe
│   ├── help_content.h     # Auto-generiert aus README.md
│   ├── machine_controller.h # Maschinenlogik
│   └── serial_protocol.h  # Serielle Kommunikation
├── src/
│   ├── main.cpp           # Hauptprogramm
│   ├── config_commands.cpp
│   ├── debug.cpp
│   ├── help.cpp
│   ├── machine_controller.cpp
│   └── serial_protocol.cpp
├── platformio.ini
└── README.md

```

Hardware

Ausgänge (Relais / Ventile / Schrittmotor)

Signal	Funktion
ventil_heben	Magnetventil Backe heben
ventil_senken	Magnetventil Backe senken / Biegen
halteklammer	Halteklammer ein/aus
auswurf	Auswerfer ein/aus
hydraulik	Hydraulikpumpe ein/aus
schrittmotor_takt	Schrittmotor-Takt (Impulse beim Fahren)
schrittmotor_richtung	Schrittmotor-Richtung (HIGH = vorwaerts, LOW = rueckwaerts / steht)

Alle Ausgänge sind standardmäßig **HIGH = aktiv** (`active=true` → AN). Beim Start werden alle Ausgänge ausgeschaltet.

In `config.h` können Ausgänge invertiert werden (z. B. für Low-aktive Relais):

```
constexpr bool INVERT_ALL_OUTPUTS = true;    // alle Ausgänge invertieren

// oder einzeln (nur wenn INVERT_ALL_OUTPUTS = false):
constexpr bool INVERT_VENTIL_HEBEN = true;
constexpr bool INVERT_HYDRAULIK = true;
```

Eingänge

Signal	Funktion
Endschalter vorne	Verhindert weitere Vorwärtsbewegung
Endschalter hinten	Verhindert weitere Rückwärtsbewegung
Winkel-Encoder	Aktueller Biege-Winkel
Höhen-Encoder	Aktuelle Backe-Höhe
Jog-Encoder	Manuelle Schrittmotor-Bewegung im Leerlauf

Pinbelegung

Übersicht

GPIO	Richtung	Signal	Anmerkung
1	TX	UART0 TX	PC-Kommunikation, nicht belegen
3	RX	UART0 RX	PC-Kommunikation, nicht belegen
4	Eingang	Jog-Encoder A	Interner Pullup
5	Eingang	Jog-Encoder B	Interner Pullup
13	Ausgang	hydraulik	Nicht GPIO12 (Boot-Pin)
14	Ausgang	auswurf	
16	Ausgang	schrittmotor_takt	Schritt-Takt
17	Ausgang	schrittmotor_richtung	HIGH = vorwaerts, LOW = rueckwaerts / steht
18	Eingang	Winkel-Encoder A	
19	Eingang	Winkel-Encoder B	
21	Eingang	Höhen-Encoder A	
22	Eingang	Höhen-Encoder B	
25	Ausgang	ventil_heben	
26	Ausgang	ventil_senken	
27	Ausgang	halteklammer	
32	Eingang	Endschalter vorne	Interner Pullup
33	Eingang	Endschalter hinten	Interner Pullup
23	—	frei	Für spätere Erweiterungen

Verdrahtung Endschalter

Standard in `config.h`:

```
ENDSTOP_ACTIVE_LOW = true
```

Das bedeutet:

- Endschalter **nicht ausgelöst** → GPIO liest **HIGH** (interner Pullup)
- Endschalter **ausgelöst** → GPIO liest **LOW** (Schalter gegen Masse)

Typische Schaltung: **NO-Endschalter** zwischen GPIO und GND.

Wichtige ESP32-Hinweise

Diese Pins nicht verwenden

GPIO	Grund
6 – 11	Intern mit Flash verbunden
0	Boot-Strapping (Download-Modus)
2	Boot-Strapping, oft onboard LED
12	Boot-Strapping (Flash-Spannung)
15	Boot-Strapping

Input-only Pins (34 – 39)

GPIO 34–39 sind **nur Eingänge** und haben **keinen internen Pullup/Pulldown**.

Deshalb ist der Jog-Encoder auf **GPIO 4/5** gelegt, nicht auf 34/35.

Serielle Schnittstelle

- **UART0**: GPIO 1 (TX) → RX des USB-UART-Adapters, GPIO 3 (RX) ← TX des Adapters
- Gemeinsame Masse (GND) nicht vergessen
- Baudrate PC und ESP32: **115200**

Serielles Protokoll

Befehlsformat

```
<Buchstabe><Zahl>\n
```

Beispiele:

V10
V2.5
F0
W0
b25
K1
Z0

Regeln:

Regel	Details
Buchstabe	Gross-/Kleinschreibung egal
Zahl	Ganzzahl; bei <code>V</code> / <code>R</code> auch Dezimal (<code>V2.5</code>)
Zeilenende	<code>\n</code> ; <code>\r\n</code> von Windows: <code>\r</code> wird ignoriert
Timeout	Standard 200 ms (per <code>#TIMEOUT200</code> aenderbar)
Gleichzeitig	Nur ein Befehl zur Zeit; waehrend Ausfuehrung: <code>ER BUSY</code>

Antworten

Antwort	Bedeutung
<code>BM_READY v1.2.0\n</code>	Steuerung bereit
<code>OK\n</code>	Befehl angenommen
<code>ER BUSY\n</code>	Maschine beschaeftigt
<code>ER SYNTAX\n</code>	Ungueltiges Format
<code>ER VALUE\n</code>	Wert ausserhalb Bereich
<code><befehl>_OK\n</code>	Erfolgreich ausgefuehrt
<code><befehl>_OK PARTIAL POS=7.000\n</code>	Endschalter, Ziel nicht erreicht
<code><befehl>_TIMEOUT\n</code>	Bewegung zu lange (Ventil/Auto)
<code>STOP_OK\n</code>	Not-Aus ausgefuehrt (<code>#STOP</code>)

Config-Befehle (Zeilen mit #)

Befehl	Antwort	Bedeutung
#HELP\n	README (ASCII)	Vollstaendige Hilfe
#STOP\n	STOP_OK\n	Not-Aus (jederzeit)
#BUSY\n	BUSY / IDLE	Maschinenstatus
#STATUS\n	#STATUS BUSY=...	Ausfuehrlicher Status
#POS\n	#POS=12.345\n	Schrittmotor-Position mm
#ANGLE\n	#ANGLE=45.00\n	Winkel Grad
#HEIGHT\n	#HEIGHT=10.00\n	Hoehe mm
#ANGLE0\n	#ANGLE0 OK\n	Winkel-Nullpunkt setzen
#CAL\n	Kalibrierung	Winkel/Hoehe/D/Timeout
#WINKEL5\n	persistent	Impulse/Grad
#HOEHE10\n	persistent	Impulse/mm
#TIMEOUT200\n	persistent	Befehls-Timeout ms (Serielleingabe)
#HYDRAULIK20000\n	persistent	Hydraulik Leerlauf-Aus ms (0 = aus)
#BENDSTOP1.5\n	persistent	Vorlauf-Stopp beim Biegen in Grad (Standard 1.5)
#DEBUG1/0/DEBUG	Debug	Wie bisher
#STEP/#STEPS/#PULSE/#INTERVAL	Schrittmotor	Wie bisher

Details zu Kalibrierung und Schrittmotor siehe unten.

Die README wird beim Kompilieren automatisch eingebettet ([scripts/embed_readme.py](#)).
 Umlaute und Sonderzeichen werden fuer das serielle Terminal in ASCII umgewandelt
 (ae/oe/ue/ss, → , ≥ , Grad usw.).

Debug-Ausgabe

Die Debug-Einstellung wird im Flash (NVS) gespeichert und bleibt nach Neustart erhalten.

```
[DBG] Empfangen: K1
[DBG] OK: cmd=K1 value=1
[DBG] Start: K1 (Wert=1)
[DBG] Fertig: K1_OK
```

Beispiel-Ablauf

```
PC → v010\n
ESP → OK\n
ESP → v010_OK\n

PC → b90\n
ESP → OK\n
ESP → b90_OK\n

PC → xyz\n
ESP → ER\n
```

Befehle

Befehl	Wert	Beschreibung
Kx	0 oder 1	Halteklammer aus (0) / ein (1)
Gx	0 oder 1	Hydraulikpumpe aus (0) / ein (1)
Px	0 oder 1	Auswerfer aus (0) / ein (1)
Z0	immer 0	Nullpunkt des Schrittmotors an aktueller Position setzen
W0	immer 0	Winkel-Nullpunkt an aktueller Encoder-Position setzen
F0	immer 0	Referenzfahrt: rueckwaerts bis Endschalter hinten, dann Z0
Vx	mm ≥ 0	Blech um x mm vorschieben (Dezimal: V2.5)
Rx	mm ≥ 0	Blech um x mm zurueckziehen (Dezimal: R1.25)
Bx	Grad ≥ 0	Auf Winkel x biegen (Encoder-Rückmeldung)
Ax	Grad ≥ 0	Automatik: Hydraulik → 1 s → x° biegen → 0° → Backe D+2 mm
Hx	mm ≥ 0	Backe auf absolute Höhe x mm heben
Dx	mm ≥ 0	Blechdicke speichern (Standard 1.5 mm, z.B. D1.5)
Sx	mm ≥ 0	Backe relativ um x mm heben (Freiraum zum Schieben)

Schrittmotor / Nullpunkt

- **Steuerung:** GPIO 16 = Takt, GPIO 17 = Richtung
- Beim Fahren: Taktpin liefert Impulse, Richtungspin **HIGH** = vorwaerts, **LOW** = rueckwaerts
- Motor steht: kein Takt, Richtungspin **LOW**
- **400 Schritte/mm** Standard (per `#STEPS400` aenderbar)
- **Rampe:** Beschleunigung/Abbremsung ueber `STEPPER_RAMP_STEPS`
- **Hardware-Timer:** Schritttakt-Timing per ESP32-Timer (50 us Aufloesung)
- Beispiel: `V10` → 4000 Schritte vorwärts
- Nach dem Einschalten ist die aktuelle Position = Nullpunkt
- `Z0` setzt den Nullpunkt neu an der aktuellen Position
- `V` / `R` beziehen sich immer auf diesen Nullpunkt
- `#POS` liefert die aktuelle Position in mm relativ zum Nullpunkt

Endschalter-Logik

- Vorwärts (`V`, Jog vorwärts): nur solange Endschalter **vorne nicht ausgelöst**
- Rückwärts (`R`, Jog rückwärts): nur solange Endschalter **hinten nicht ausgelöst**
- Bei ausgeloestem Endschalter: Stopp mit `_OK PARTIAL POS=...` (nicht volle Strecke)
- `#STOP` bricht jederzeit ab (kein `_OK` fuer den laufenden Befehl)

Sicherheit

- **#STOP** — Schrittmotor aus, Ventile aus, Hydraulik aus
- **Bewegungs-Timeout** — 30 s fuer B/H/S/A, dann `_TIMEOUT`
- **Hydraulik Auto-Aus** — nach Bewegungsbefehlen und nach Leerlauf (Standard 20 s, `#HYDRAULIK0` = aus)
- **Bewegungs-Timeout** — fuer B/H/S/A (Standard 30 s, `#MOTION0` = aus, fuer langsame Bleche)

Zurueck zur alten Firmware

Siehe [RESTORE.md](#): `git checkout baseline-v1.1`

Ablauf der Befehle

Sofort-Befehle (K, G, P, Z, W, D)

Werden direkt ausgefuehrt. Nach `COMMAND_DONE_DELAY_MS` (100 ms) folgt `_OK`.

Bewegungs-Befehle (V, R, B, H, S, A, F)

1. PC sendet Befehl

2. ESP antwortet `OK\n`
3. Maschine führt Bewegung aus (Ventile / Schrittmotor)
4. Nach Abschluss + Verzögerung: `<befehl>_OK\n`

Automatik-Biegen (`Ax`)

Ablauf:

1. **Hydraulik ein** (`G1` intern)
2. **1 Sekunde warten** (Druck aufbauen)
3. **Biegen** — um x° (relativ), danach Rueckbiegen auf 0°
4. **Anheben** — Backe auf **D + 2 mm** (absolut)

Standard-Blechdicke **D = 1.5 mm** (per `D1.5` oder `#CAL` aenderbar).

Beispiel Standard: Anheben auf **3.5 mm**.

Maschinenlogik

Biegen (`Bx`)

- Ventil **senken** → Winkel vergrößern
- Ventil **heben** → Winkel verkleinern
- Zielwinkel erreicht ($\pm$ Toleranz) → Ventil aus, `_OK`

Heben (`Hx` , `Sx`)

- `Hx` — absolute Zielhöhe in mm (Höhen-Encoder)
- `Sx` — relative Hebung um x mm ab aktueller Position
- Zielhöhe erreicht ($\pm$ Toleranz) → Ventile aus, `_OK`

Manuelles Joggen (Jog-Encoder)

- Nur im **Leerlauf** (kein laufender Serienbefehl)
- Jeder Encoder-Impuls → `Impulse × MANUAL_ENCODER_FACTOR` Schritte
- Standard-Faktor: **25** (in `config.h` änderbar)
- Ein-/Ausschalten: `MANUAL_ENCODER_ENABLED`

Konfiguration (`config.h`)

Alle wichtigen Parameter liegen in `include/config.h`:

Parameter	Standard	Bedeutung
<code>SERIAL_BAUD</code>	115200	Baudrate
<code>COMMAND_TIMEOUT_DEFAULT_MS</code>	200	Startwert Befehls-Timeout (per <code>#TIMEOUT</code> änderbar)
<code>MOTION_TIMEOUT_DEFAULT_MS</code>	30000	Bewegungs-Timeout B/H/S/A (per <code>#MOTION</code> , <code>0</code> =aus)
<code>HYDRAULIK_IDLE_DEFAULT_MS</code>	20000	Hydraulik Leerlauf-Aus (per <code>#HYDRAULIK</code> , <code>0</code> =aus)
<code>BEND_EARLY_STOP_DEG</code>	1.5	Standard Vorlauf-Stopp (per <code>#BENDSTOP</code> änderbar)
<code>COMMAND_DONE_DELAY_MS</code>	100	Pause vor <code>_OK</code> -Antwort
<code>STEPPER_DEFAULT_STEPS_PER_MM</code>	400	Startwert Schritte/mm (per <code>#STEPS</code> änderbar)
<code>STEPPER_DEFAULT_PULSE_US</code>	800	Startwert Impulsdauer
<code>STEPPER_DEFAULT_INTERVAL_US</code>	1200	Startwert Schritt-Pause
<code>MANUAL_ENCODER_FACTOR</code>	25	Jog-Multiplikator
<code>ENDSTOP_ACTIVE_LOW</code>	true	Endschalter aktiv LOW
<code>ENCODER_WINKEL_IMPULSE_PRO_GRAD</code>	1.0	Impulse pro Grad (Winkel-Encoder)
<code>ENCODER_HOEHE_IMPULSE_PRO_MM</code>	10.0	Impulse pro Millimeter (Höhen-Encoder)
<code>ANGLE_TOLERANCE_DEG</code>	0.5	Toleranz Biegen
<code>HEIGHT_TOLERANCE_MM</code>	0.1	Toleranz Heben
<code>DEBUG_ENABLED</code>	1	Debug-Code kompilieren (0 = komplett deaktiviert)
<code>DEBUG_DEFAULT_ENABLED</code>	false	Debug beim Erststart, falls NVS leer

Debug global deaktivieren (Compile-Time)

In `config.h` setzen:

```
#define DEBUG_ENABLED 0
```

Dann wird kein Debug-Code eingebunden und die `#DEBUG` -Befehle antworten mit `#DEBUG UNAVAILABLE`.

Kalibrierung Encoder

```
// Winkel in Grad:  
Winkel = Impulse / ENCODER_WINKEL_IMPULSE_PRO_GRAD  
  
// Höhe in mm:  
Hoehe = Impulse / ENCODER_HOEHE_IMPULSE_PRO_MM
```

Beispiel: 900 Impulse bei 90° → `ENCODER_WINKEL_IMPULSE_PRO_GRAD = 10.0`

Werte an der realen Mechanik messen und in `config.h` eintragen.

Build und Upload

Voraussetzungen: [PlatformIO](#) (CLI oder in CLion/VS Code).

```
# Kompilieren  
pio run  
  
# Auf ESP32 flashen  
pio run -t upload  
  
# Serieller Monitor (115200 Baud)  
pio device monitor -b 115200
```

Nach dem Upload oder beim Öffnen des seriellen Ports sollte erscheinen:

```
BM_READY
```

PDF-Dokumentation

Moderne PDF aus dieser README erzeugen:

```
pip install markdown playwright  
python -m playwright install chromium  
python scripts/generate_readme_pdf.py
```

Ergebnis: `docs/Biegemaschinen-Steuerung.pdf`

Fehlersuche

Problem	Mögliche Ursache
Kein <code>BM_READY</code>	Falsche Baudrate, TX/RX vertauscht, kein GND
Immer <code>ER\n</code>	Maschine noch beschäftigt oder ungültiger Befehl
Schrittmotor bewegt sich nicht	Endschalter ausgelöst, falsche Pin-Verdrahtung
Schrittmotor falsche Strecke	<code>#STEP</code> prüfen, <code>#STEPS400</code> setzen
Biegen stoppt nicht	<code>ENCODER_WINKEL_IMPULSE_PRO_GRAD</code> kalibrieren
Heben ungenau	<code>ENCODER_HOEHE_IMPULSE_PRO_MM</code> kalibrieren
ESP startet nicht / bootet ständig	GPIO 0/2/12/15 oder 6–11 belegt
Jog-Encoder reagiert nicht	Nur im Leerlauf aktiv; Pins 4/5 prüfen

Typischer Testablauf

1. Seriellen Monitor öffnen → `BM_READY` abwarten
2. `G1` — Hydraulik ein → `OK` → `G1_OK`
3. `Z0` — Nullpunkt setzen
4. `V1` — 1 mm verschieben (400 Schritte)
5. `K1` — Halteklammer zu
6. `D2` — Blechdicke 2 mm setzen
7. `A90` — Automatik-Biegen auf 90°

Erweiterbarkeit

Neue Befehle können in `machine_controller.cpp` ergänzt werden:

1. Buchstabe in `CommandType` (`machine_controller.h`)
2. Validierung in `parse()`
3. Ausführung in `startCommand()` und ggf. `update()`

Pin-Änderungen nur in `config.h` vornehmen.