Checkliste Spindel mit Linearführung

Stand: 05.11.2026, Branch `Stepper`, inklusive uncommitteter Änderungen an `stepper.c`.

Grundlage: Pflichtenheft, Interfacespezifikation und Anforderungen an den Laborbericht, abgeglichen mit dem Code. Kompiliert oder getestet wurde dabei nichts, „funktioniert“ stammt aus den Commit-Messages.

Bewertung: Die MUSS-Kriterien bringen 50 % der Punkte (Note 4,0), für die 1,0 sind 100 von 130 Punkten nötig. Die Punktzahlen pro Feature stehen nicht im extrahierten PDF-Text, die Reihenfolge der Wunschkriterien ist daher eine Einschätzung.

Priorität 1: MUSS-Kriterien (Firmware)

Tests und Capability

- □ **SpecTest-App besorgen:** Der `Test`-Ordner fehlt im Repo, und die MUSS-Features müssen diesen Test fehlerfrei bestehen.
- □ **Capability-Bits setzen:** Aktuell sind alle 0 (`main.c:143`). Nötig sind Bit 0 (Spindle), 1 (Spindle Status), 2 (Stepper), 7 (Referenzfahrt) und 11 (Reset).

Spindel (F10_01–03)

- □ **Debug-Ausgaben entfernen:** Die `printf` in `spindel.c:22` und `:42` müssen raus. Der Spindel-Task kann sie beim Startup-Boost mitten in andere Befehlsausgaben schreiben.
- □ **Alle Fälle testen:** Negative RPM, Kappung bei Min/Max, Mehrfach-Start und -Stopp sowie `spindle status`.

Stepper `reset` (F20_01)

- □ **Mehrfach testen:** `return result` ist neu und uncommittet, davor kam immer 0 zurück. Prüfen, ob wiederholtes `reset` weiterhin `OK` liefert, auch aus dem Fehlerzustand.

Stepper `reference` (F20_02)

- □ **Endlosschleife beheben:** Die zweite Schleife ignoriert Fehler (`stepper.c:150–151`). Dann hängt die Konsole, und `stepper reset` ist nicht mehr möglich.
- □ **Rückgabewerte prüfen:** Das fehlt bei `SetPowerOutputs` (`stepper.c:117`) und beim Wegfahren um 5 mm (`stepper.c:121`).
- □ **Abbruch einbauen:** Es braucht ein internes Weg- oder Zeitlimit, falls der Sensor nie auslöst. Optional kann auch `LIMIT_SWITCH` abbrechen, er darf aber nicht zur Wegbestimmung dienen.
- □ **Bei Fehler Treiber abschalten:** Aktuell bleiben sie aktiv.

Stepper `move` (F20_03)

- □ **Bug beheben:** `move` auf die aktuelle Position ergibt `FAIL` und `state = 0`, weil `L6474_StepIncremental` bei 0 Schritten einen Fehler liefert (`stepper.c:189–191`). Das muss direkt `OK` ergeben.

- [ ] **Eingabe prüfen:** `atof` macht aus `stepper move abc` die Position 0 mit `OK` (`stepper.c:301`).
- [ ] **Fahrbereich klären:** Die Grenzen 0–100 mm sind hartkodiert (`stepper.c:303`). Gegen den echten mechanischen Weg prüfen.
- [ ] **Fehlerbits auswerten:** Der Status wird gelesen, aber verworfen (`stepper.c:195–198`). OCD, TH_SD und UVLO prüfen und bei Fehlern in einen Fehlerzustand wechseln. Das deckt auch den geforderten Überlastschutz ab (Kap. 2.4).
- [ ] **Zustandsautomat umbauen:** Die Werte der Spezifikation nehmen: `scsINIT` 0x0, `scsREF` 0x1, `scsDIS` 0x2, `scsENA` 0x4, `scsFLT` 0x8.
- [ ] **Status-LEDs umsetzen:** Grün heißt fehlerfrei, Gelb blinkt mit 1 Hz bei aktiven Treibern, Rot heißt Fehler. Die Funktionen in `main.c` sind ungenutzt, und zwei davon sind Dauerschleifen. Besser ist ein einziger LED-Task, der `state` auswertet.

Aufräumen und Abschluss

- [ ] **Nicht spezifizierte Befehle entfernen:** Das betrifft `fabi` (`main.c:317`) und `stepper step` (`stepper.c:222`).
- [ ] **Stand committen:** Die Änderungen an `stepper.c` sind nicht committet und `Stepper` ist nicht in `main` gemergt.
- [ ] **SpecTest komplett bestehen lassen.** Dabei auch die CM4-Firmware flashen, die laut Pflichtenheft ohnehin auf den zweiten Kern muss.

Priorität 2: Laborbericht (verbindlich)

Im Repo gibt es dazu noch nichts.

Messungen (am besten erst mit der finalen Firmware)

- [ ] SPI: SCK, MOSI, MISO, CS sowie ein L6474-Kommunikationsvorgang
- [ ] H-Brücken-PWM: Frequenz, Duty Cycle, Pegel
- [ ] DIR und PULSE: Richtungswechsel, Bezug zum Microstepping

Inhalt

- [ ] Software-Architektur und Implementierungsansatz
- [ ] Stepper-Controller als Schwerpunkt, inklusive Abweichungen von der Spezifikation
- [ ] UML: Use-Case-Diagramm (Bring-Up bis Move) und Sequenzdiagramm
- [ ] Soll/Ist-Vergleich mit technischer Bewertung und Fazit

Formales

- [ ] Mindestens 12 Seiten ohne Deckblatt und Literaturverzeichnis
- [ ] Abbildungen nummeriert und im Text referenziert
- [ ] Große Messreihen im Anhang, Quellen angegeben

Priorität 3: Wunschkriterien

Wenig Aufwand

- [ ] `stepper position`: Ausgabe in mm als Float (`100.0000`), aktuell Text mit Schritten
- [ ] `stepper status`: Zustand, Fehlerbits und Pending-Flag ausgeben
- [ ] `reference -e`, `-s` und `-t`: Flags werden noch gar nicht geparst
- [ ] `stepper config powerena`: ändert die Bring-Up-Sequenz, also `scsDIS` einbauen
- [ ] `move -r`

Mehr Aufwand, aber lohnend

- [ ] **Timer-Pulserzeugung, Master/Slave (F20_09):** Aktuell sind etwa 37,5 mm/min möglich, 100 mm dauern also rund 160 s. Gefordert sind bis zu 1400 mm/min. Das kann auch für SpecTest-Timeouts relevant sein, darum am besten zuerst.
- [ ] **`move -s`:** Baut auf den Timer auf.
- [ ] **Asynchroner LibL6474-Betrieb (F20_11):** Dazu gehören `move -a` und `stepper cancel`.
- [ ] **`stepper config` mit den übrigen Parametern:** Das gibt Einzelpunkte pro Parameter.
- [ ] **Testprotokoll:** Entsteht aus dem SpecTest-Lauf.

Zuletzt

- [ ] Netzwerk über OpenAMP (LibETHProducer)
- [ ] Unit-Tests mit CMocka: mindestens 4 Tests, nur unter Windows nutzbar
- [ ] Digital Twin: auch nur unter Windows, und der `Simulation`-Ordner fehlt im Repo

Offene Fragen

- [ ] **Welcher Motor wird genutzt?** Der Code nutzt 700 mA und 800 Schritte/mm (`stepper.c:103`, `stepper.c:17`). Beim kleinen Labormotor sind maximal 0,6 A erlaubt, die 700 mA wären also zu viel. Mit seinen 400 Schritten wären außerdem 1600 Schritte/mm richtig, sonst fährt er nur die halbe Strecke. Beim 200-Schritt-Motor der Baugruppe passen die 800 Schritte/mm. Am besten beide Werte als Konstanten anlegen.
- [ ] **Sind die LEDs MUSS oder Wunsch?** Das Pflichtenheft widerspricht sich: Kap. 1.1.4 nennt sie als Wunschkriterium, F20_03 (MUSS) schreibt sie vor. Hier sind sie als MUSS eingeordnet. Am besten bei Herrn Kimmel nachfragen.