Hau den Lukas!

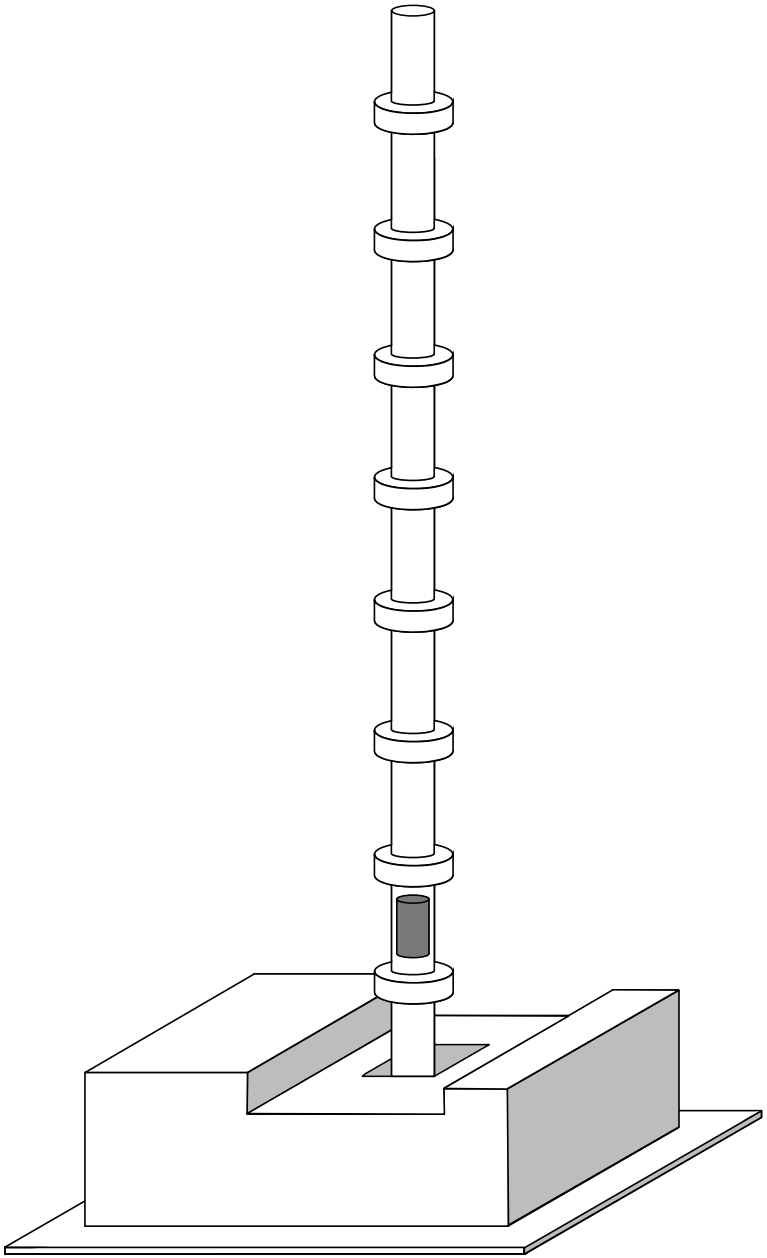
Echtzeitsysteme II

Wilhelm Haas
Christian Meier
Patrick Kugler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik
Lehrstuhl 4

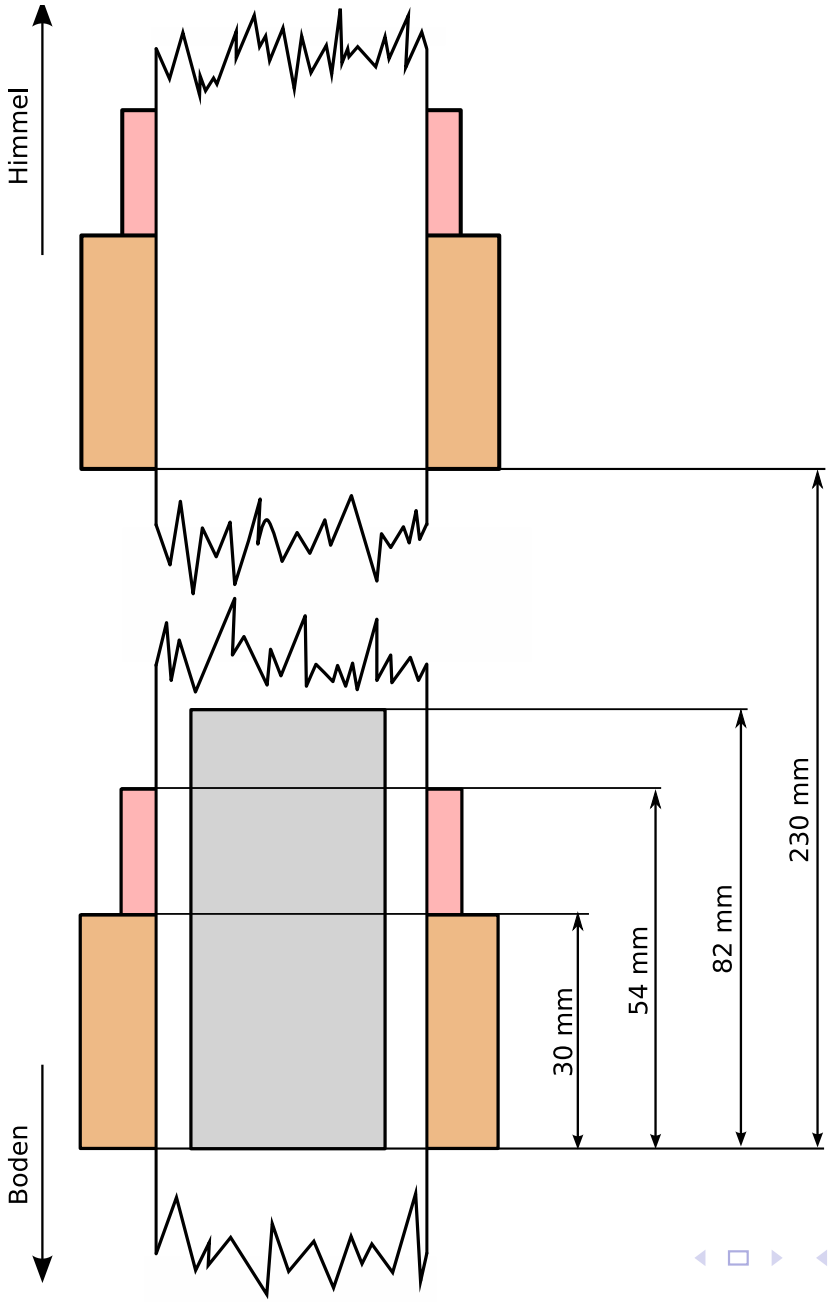
27. Juli 2006

Überblick



- **Hardware**
  - TriCore 1796
  - 8 Spulen
  - 8 Lichtschranken
  - 1 Projektil
- **Software**
  - ProOSEK
  - jtag-Server
  - GCC-Tools

Geometrie



Ausschnitt aus dem Handbuch (2057 Seiten ohne Befehlssatz):

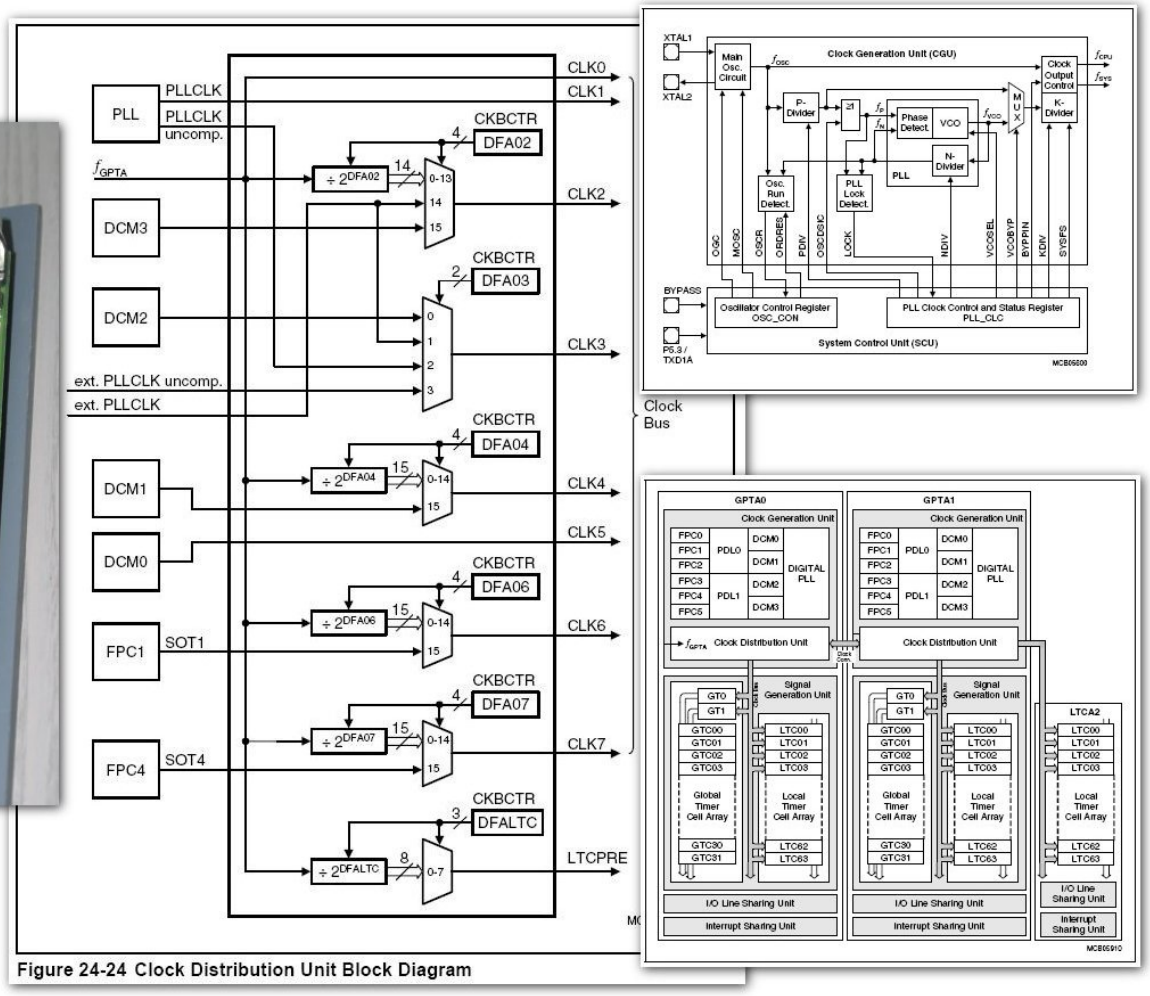
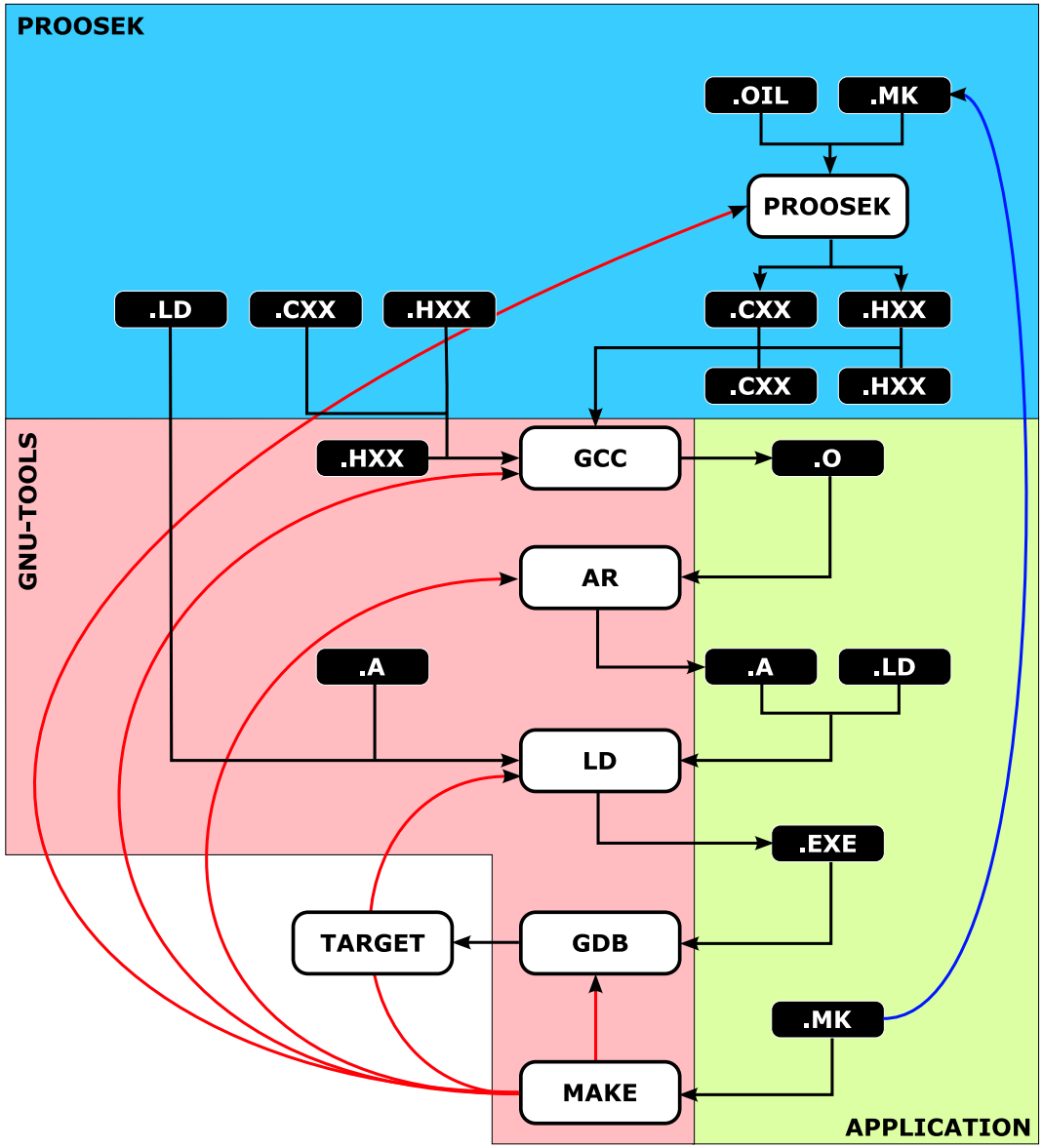


Figure 24-24 Clock Distribution Unit Block Diagram

- Primitiven zur Steuerung
  - UpContinuous
  - DownContinuous
  - Hold
- Kombinationen
  - UpStepwise
  - DownStepwise
  - OscContinuous
  - OscStepwise
- Beliebige Kombinationen möglich

Überblick


PROOSEK
.OIL
.MK
PROOSEK
.LD
.CXX
.HXX
.CXX
.HXX
.CXX
.HXX
GNU-TOOLS
.HXX
GCC
.O
AR
.A
.A
.LD
LD
.EXE
TARGET
GDB
.MK
MAKE
APPLICATION

Make-Targets

- Allgemeine Make-Targets
  - **jtag** startet den jtag-Server
  - **config** startet ProOSEK
  - **clean** löscht alle erzeugten Dateien
  - **list** gibt eine Liste aller Testfälle aus

- Make-Targets für Testfälle
  - **show-%** zeigt die Testfallspezifikation an
  - **config-%** öffnet die OIL-Datei mit ProOSEK
  - **build-%** erzeugt den Testfall
  - **gdb-%** startet den gdb mit dem Testfall
  - **run-%** startet den gdb mit dem Testfall und führt ihn aus
  - **clean-%** löscht die erzeugten Dateien
  - **image-%** erzeugt das Image und Kopiert es

Testsystem

- Testfälle liegen unter
  - für Modultests:
    LUKAS/test/module
  - für Integrationstests:
    LUKAS/test/integration
  - für endgültige Anwendungen:
    LUKAS/test/final

- Testfälle bestehen aus
  - Testfallspezifischen Code-Dateien:
    LUKAS/test/final/XXX/*.c
  - Testfallbeschreibung:
    LUKAS/test/final/XXX/testcase.mk

PROJ_NAME := *counter*
PROJ_DIR := *test/final/counter*

DESCRIPTION := *Ein binärer Counter mit Start, Stop, Reset*
COMPONENTS := *GPIO*
REQUIREMENTS := *GPIO_WRITE GPIO_READ*

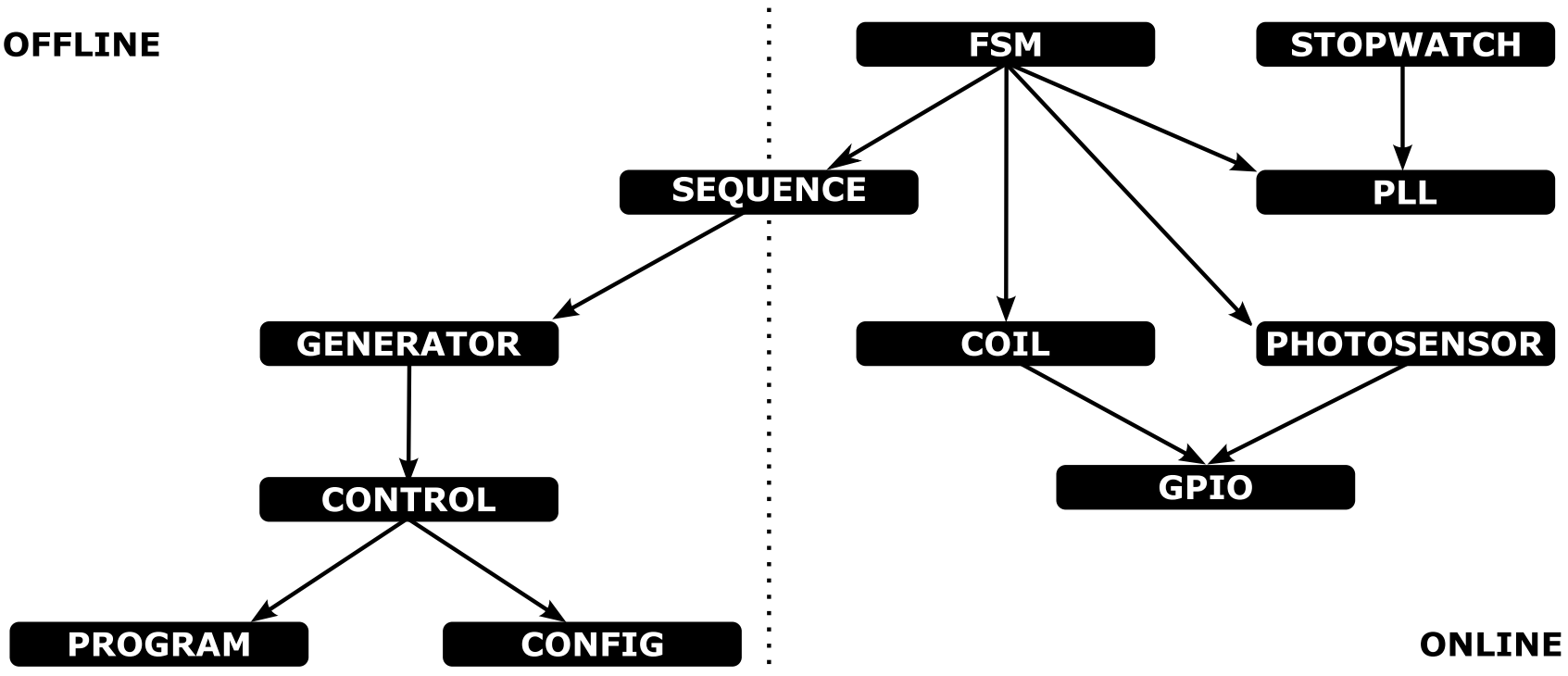
TEST_PROCEDURE := *Mit den Tasten 0 - 2 kann der Counter gestartet, gestoppt und auf 0 gesetzt werden. Der Counterwert wird binär angezeigt.*
TEST_OK := *Counterwert erhöht sich in 500 ms Abständen, die Tasten funktionieren wie beschrieben.*

SRC_FILES := *src/device/gpio.c src/device/gpio.h*
OBJ_FILES := *gpio.o counter.o*
OIL_FILE := *test/final/counter/counter.oil*

include *common.mk*

Überblick

Module


OFFLINE
FSM
STOPWATCH
SEQUENCE
PLL
GENERATOR
COIL
PHOTOSENSOR
CONTROL
GPIO
PROGRAM
CONFIG
ONLINE

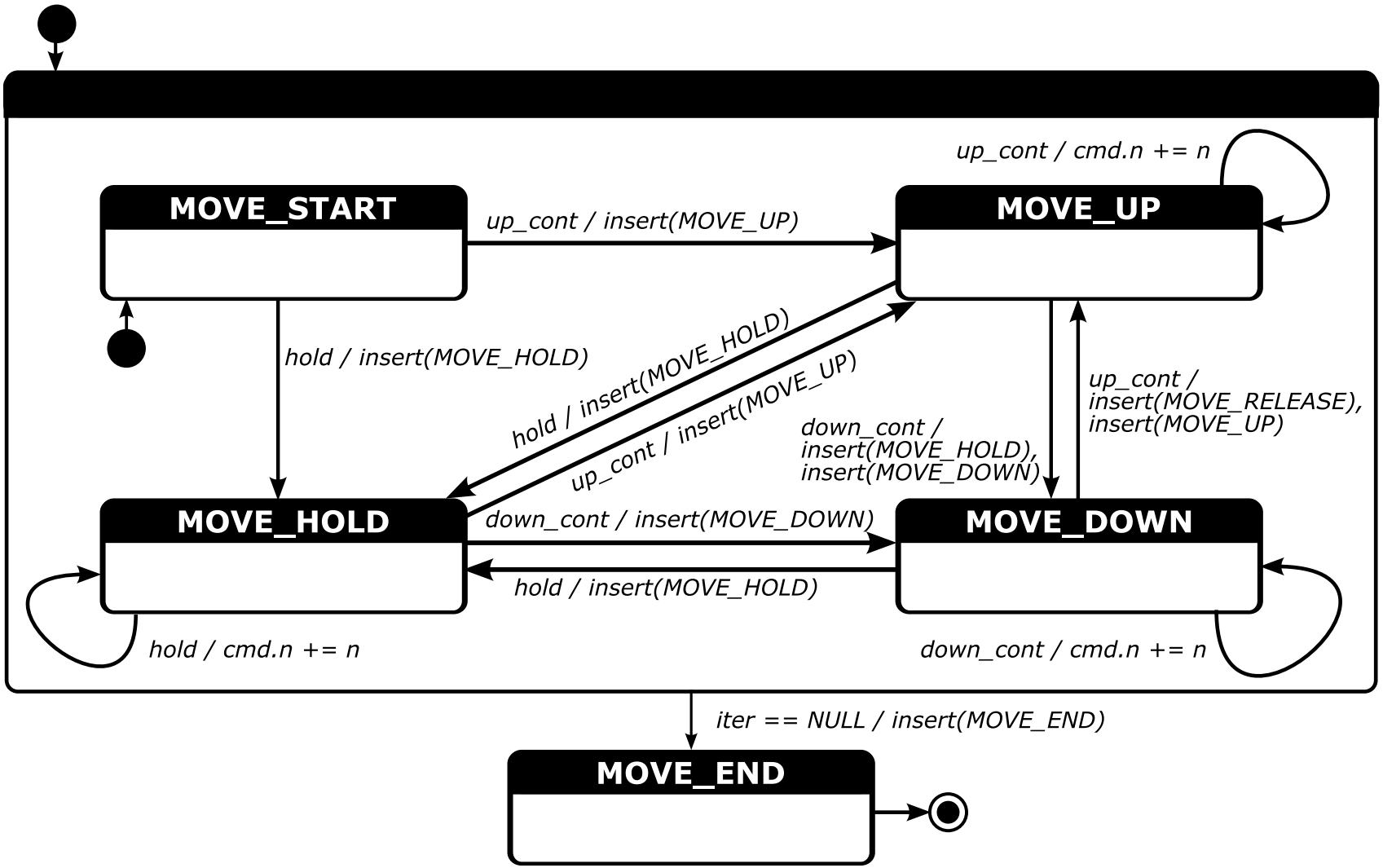
PROGRAM

NewProgram() : program_t *
DeleteProgram(program_t * prg) : void
UpContinuous(program_t * prg, int coils) : void
DownContinuous(program_t * prg, int coils) : void
UpStepwise(program_t * prg, int coils) : void
DownStepwise(program_t * prg, int coils) : void
OscContinuous(program_t * prg, int coils, int rounds) : void
OscStepwise(program_t * prg, int coils, int rounds) : void
Hold(program_t * prg, int ms) : void

- **NewProgram** erzeugt ein neues Programm
- **DeleteProgram** löscht das Programm
- **UpContinuous** hebt das Projektil um **coils** Spulen an
- **DownContinuous** senkt das Projektil um **coils** Spulen ab
- **UpStepwise** hebt das Projektil schrittweise um **coils** Spulen an
- **DownStepwise** senkt das Projektil schrittweise um **coils** Spulen ab

- **OscContinuous**
  - **coils** > 0**:** Schwingt um **coils** Spulen nach oben
  - **coils** < 0**:** Schwingt um **coils** Spulen nach unten
  - wiederholt den Vorgang **rounds** mal
- **OscStepwise** analog zu OscContinuous, nur schrittweise
- **Hold** hält das Projektil **ms** Millisekunden an der aktuellen Spule fest

Compiler


MOVE_START
MOVE_UP
MOVE_HOLD
MOVE_DOWN
MOVE_END
up_cont / cmd.n += n
up_cont / insert(MOVE_UP)
hold / insert(MOVE_HOLD)
hold / insert(MOVE_HOLD)
up_cont / insert(MOVE_UP)
down_cont / insert(MOVE_HOLD), insert(MOVE_DOWN)
up_cont / insert(MOVE_RELEASE), insert(MOVE_UP)
down_cont / insert(MOVE_DOWN)
hold / insert(MOVE_HOLD)
hold / cmd.n += n
down_cont / cmd.n += n
iter == NULL / insert(MOVE_END)

Zustandsdiagramm - COMPILER

UpContinuous(prog, 1)

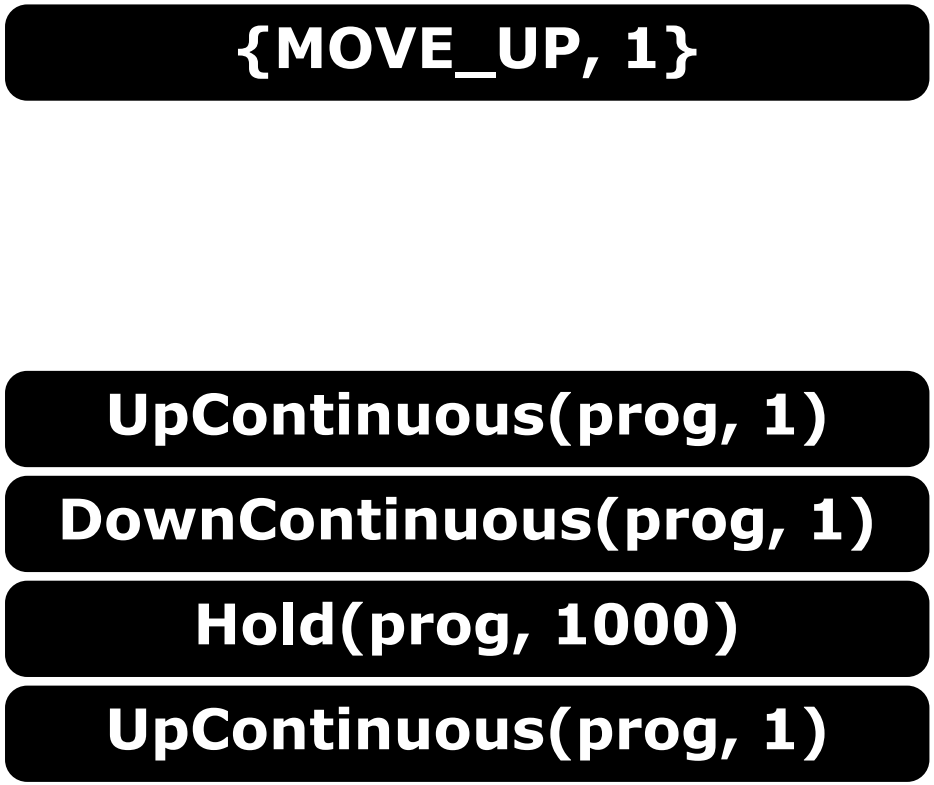
UpContinuous(prog, 1)

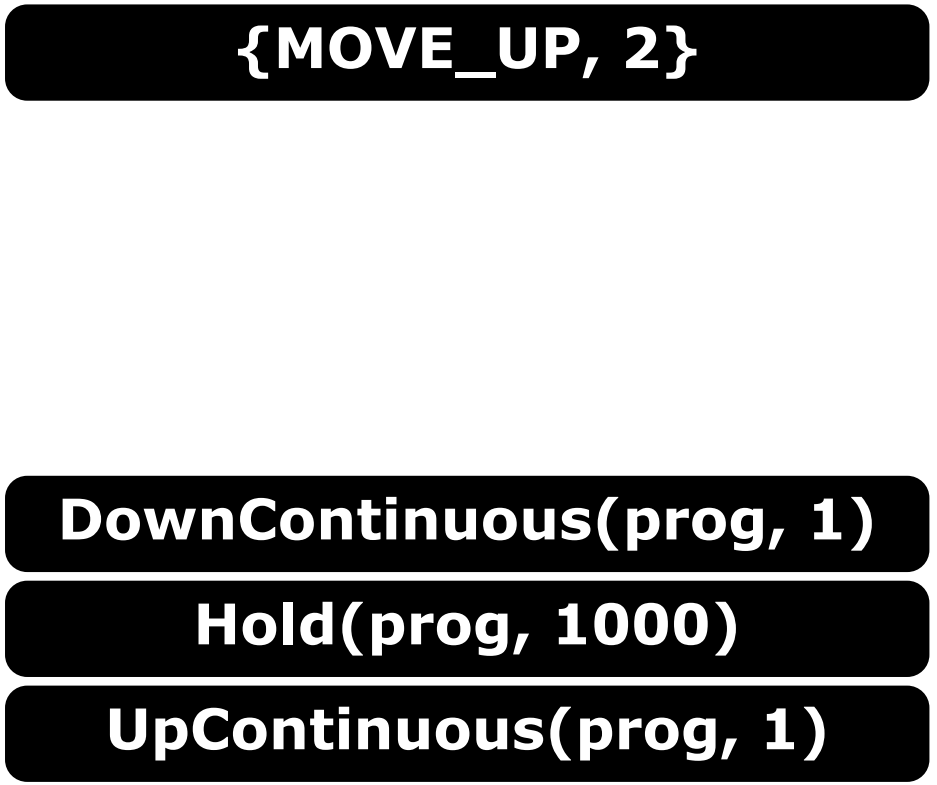
DownContinuous(prog, 1)

Hold(prog, 1000)

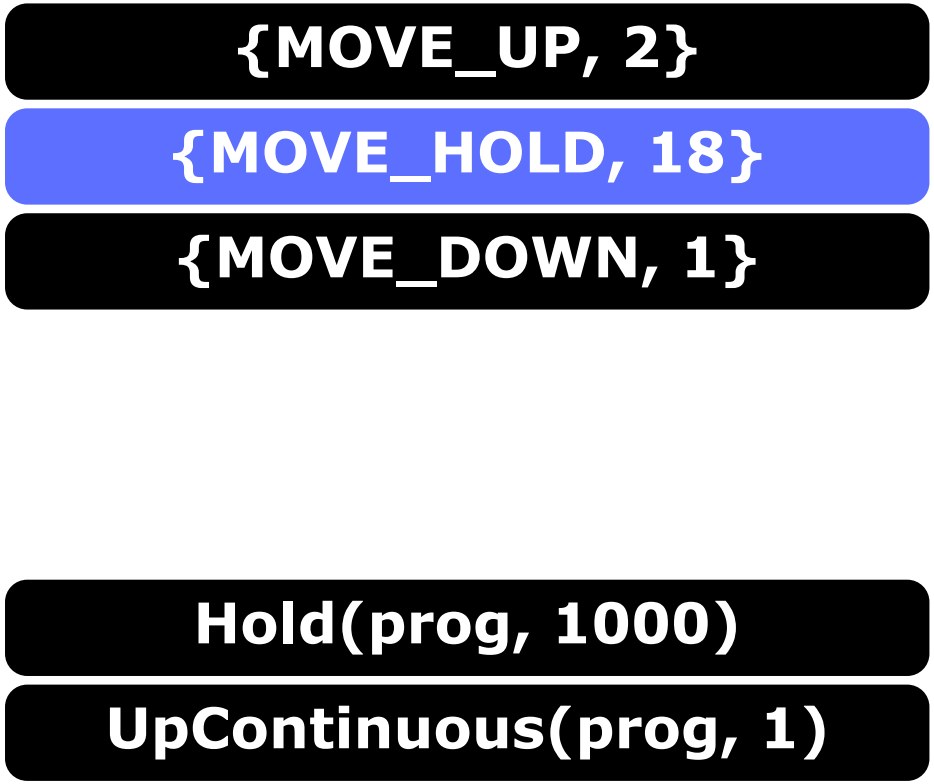
UpContinuous(prog, 1)

Zustandsdiagramm - COMPILER

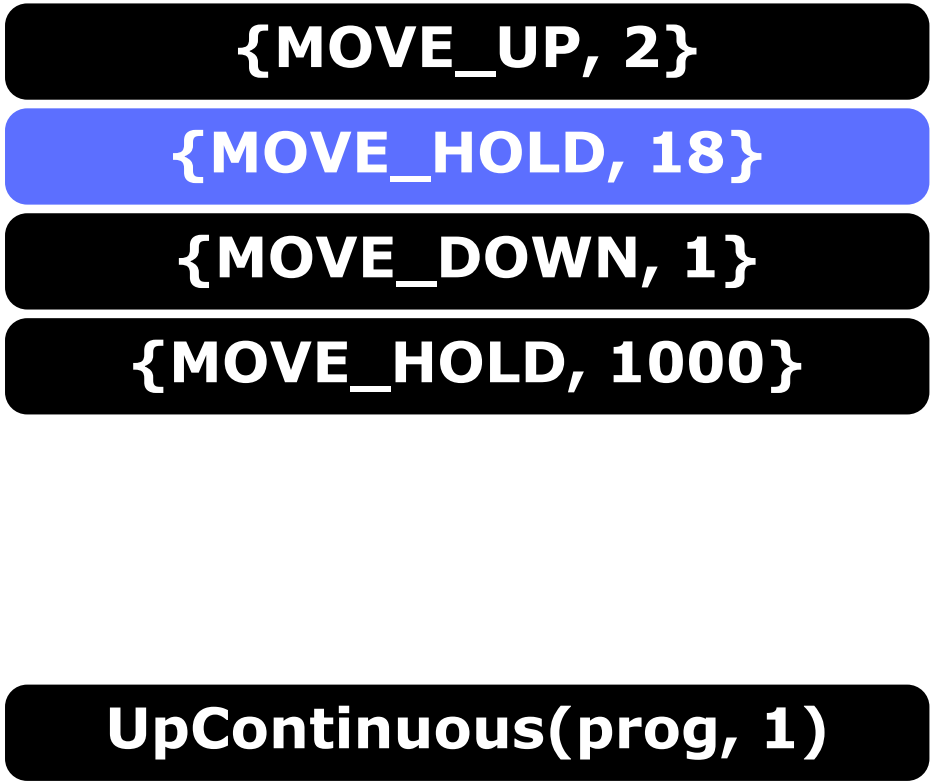



{MOVE_UP, 2}
DownContinuous(prog, 1)
Hold(prog, 1000)
UpContinuous(prog, 1)

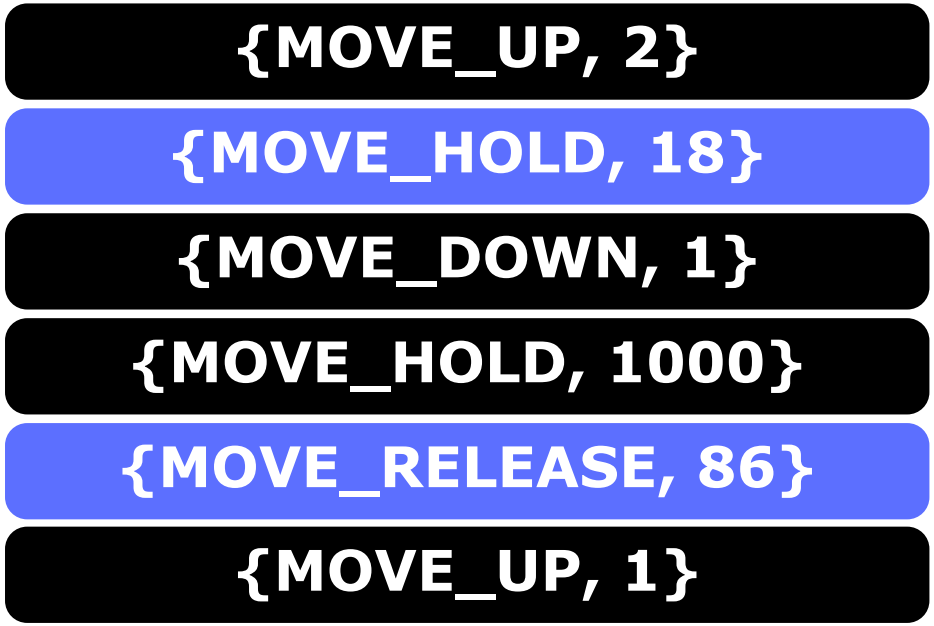
Zustandsdiagramm - COMPILER



Zustandsdiagramm - COMPILER

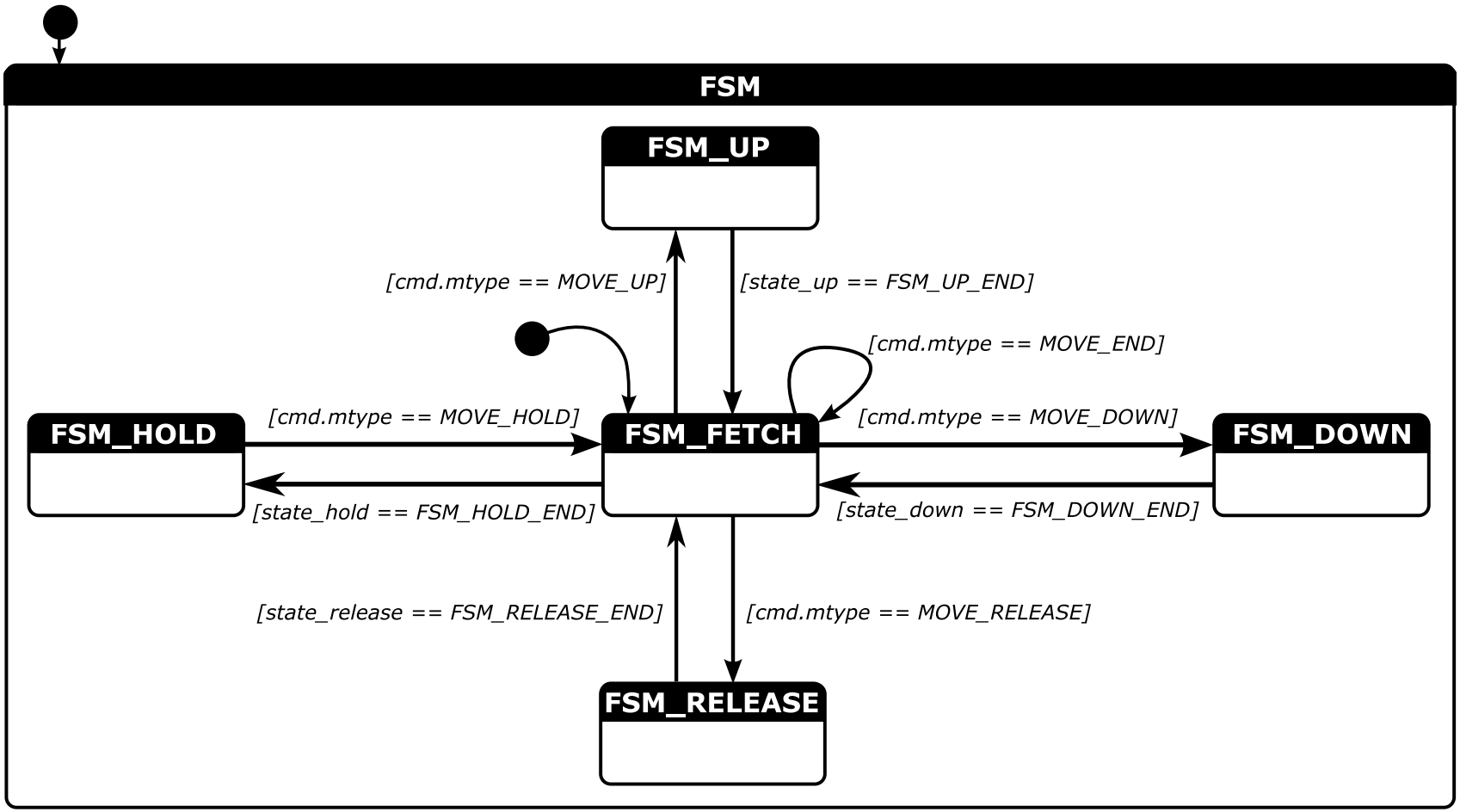


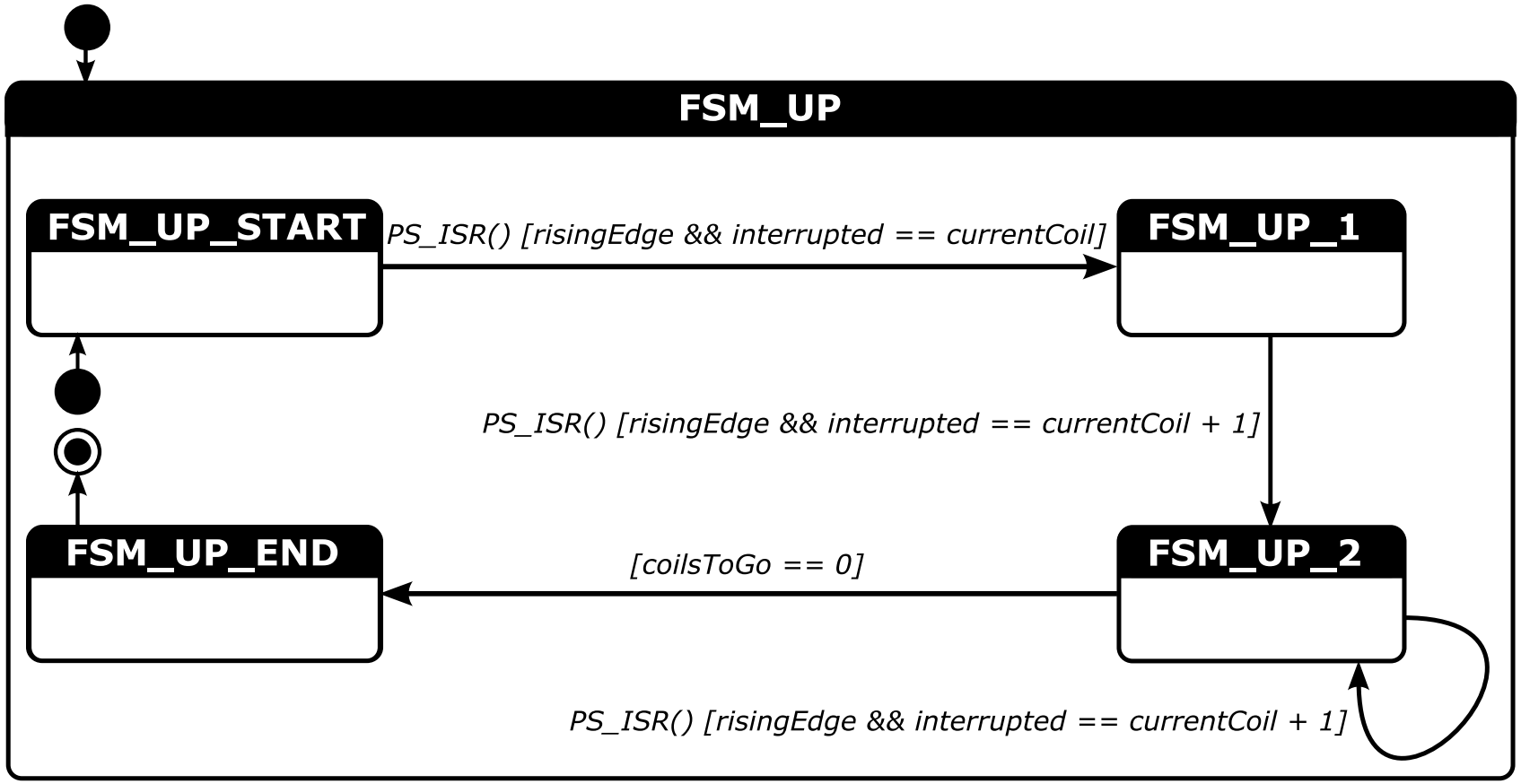
Zustandsdiagramm - COMPILER



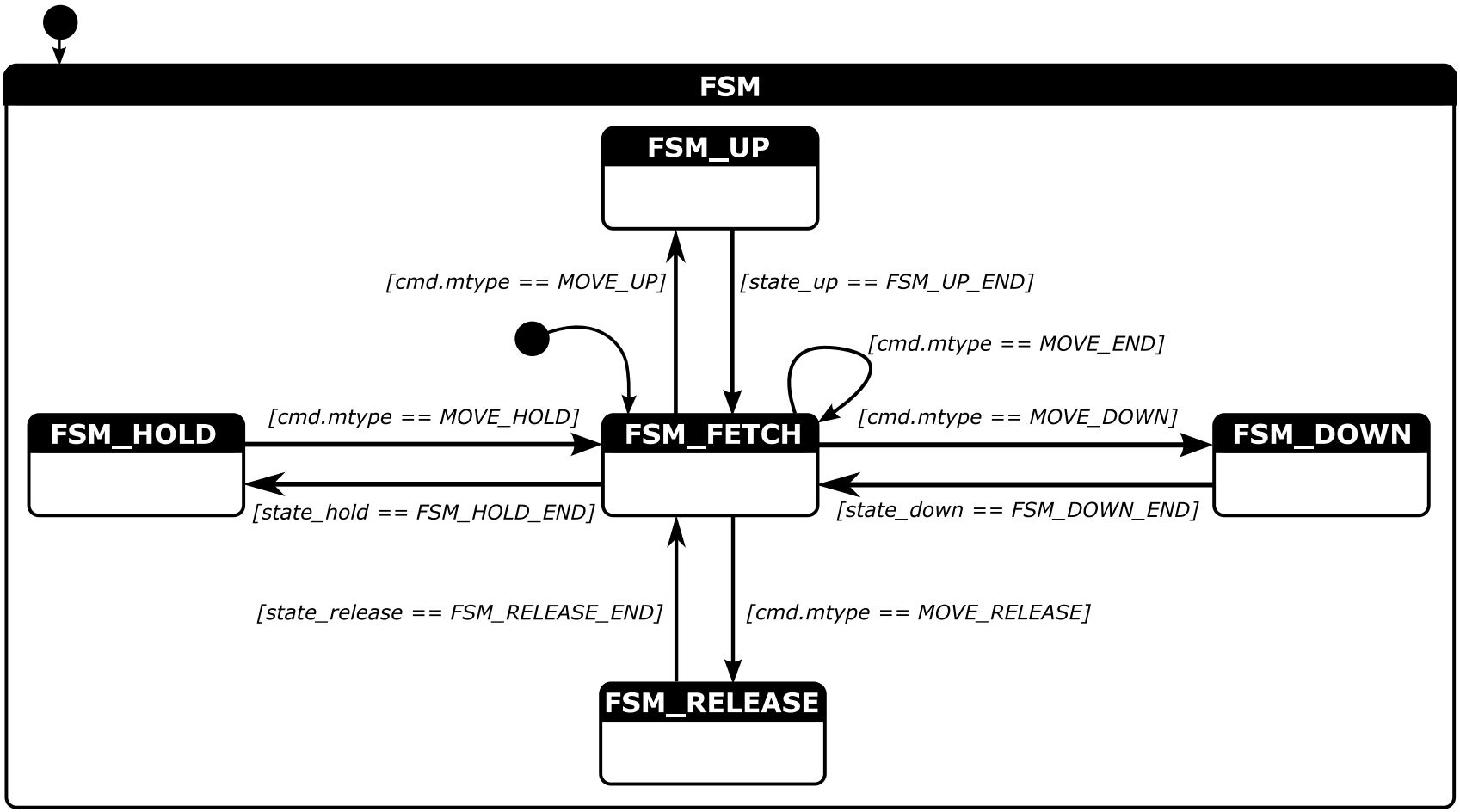
{MOVE_UP, 2}
{MOVE_HOLD, 18}
{MOVE_DOWN, 1}
{MOVE_HOLD, 1000}
{MOVE_RELEASE, 86}
{MOVE_UP, 1}
{MOVE_HOLD, 18}
{MOVE_DOWN, 2}
{MOVE_HOLD, 300}
{MOVE_END, 0}

FSM

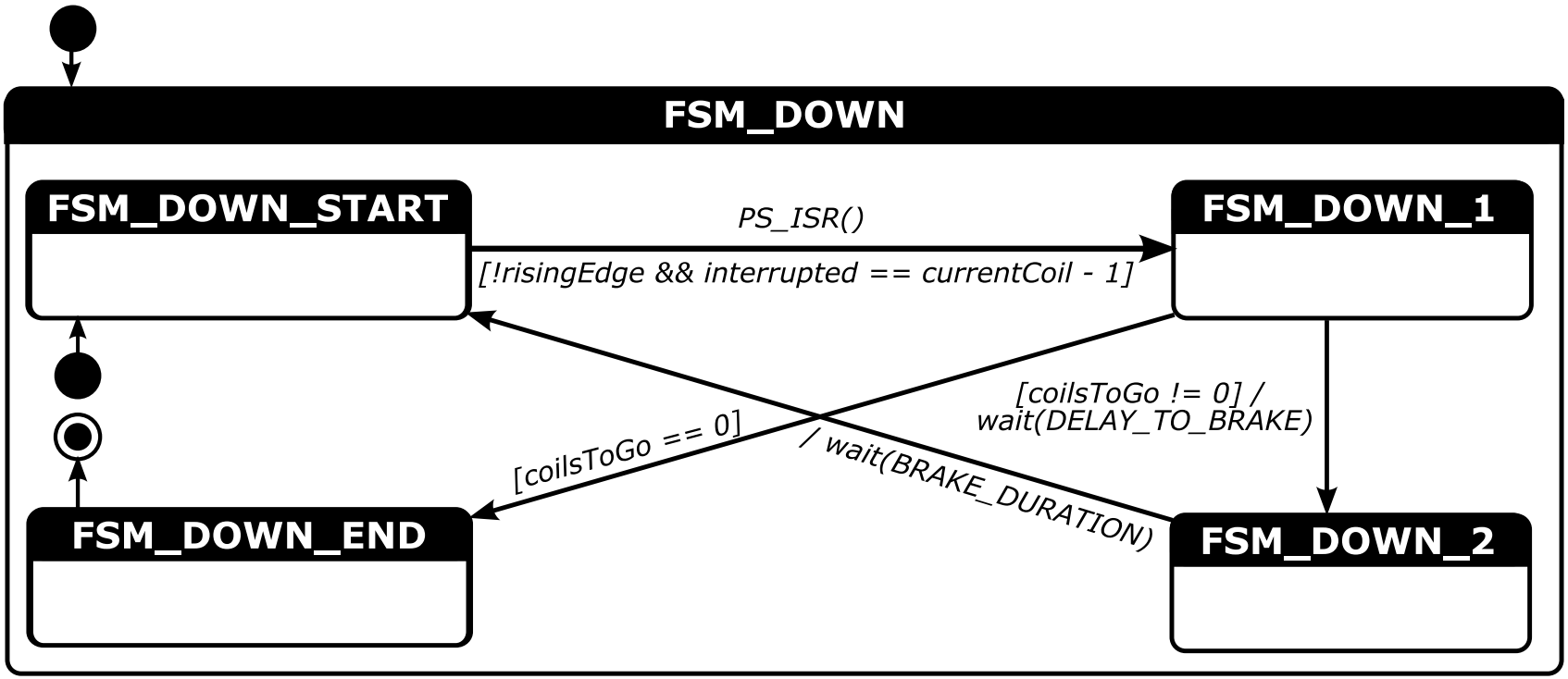

FSM
FSM_UP
[cmd.mtype == MOVE_UP]
[state_up == FSM_UP_END]
[cmd.mtype == MOVE_END]
FSM_HOLD
[cmd.mtype == MOVE_HOLD]
FSM_FETCH
[cmd.mtype == MOVE_DOWN]
FSM_DOWN
[state_hold == FSM_HOLD_END]
[state_down == FSM_DOWN_END]
[state_release == FSM_RELEASE_END]
[cmd.mtype == MOVE_RELEASE]
FSM_RELEASE


FSM_UP
FSM_UP_START
PS_ISR() [risingEdge && interrupted == currentCoil]
FSM_UP_1
PS_ISR() [risingEdge && interrupted == currentCoil + 1]
FSM_UP_END
[coilsToGo == 0]
FSM_UP_2
PS_ISR() [risingEdge && interrupted == currentCoil + 1]

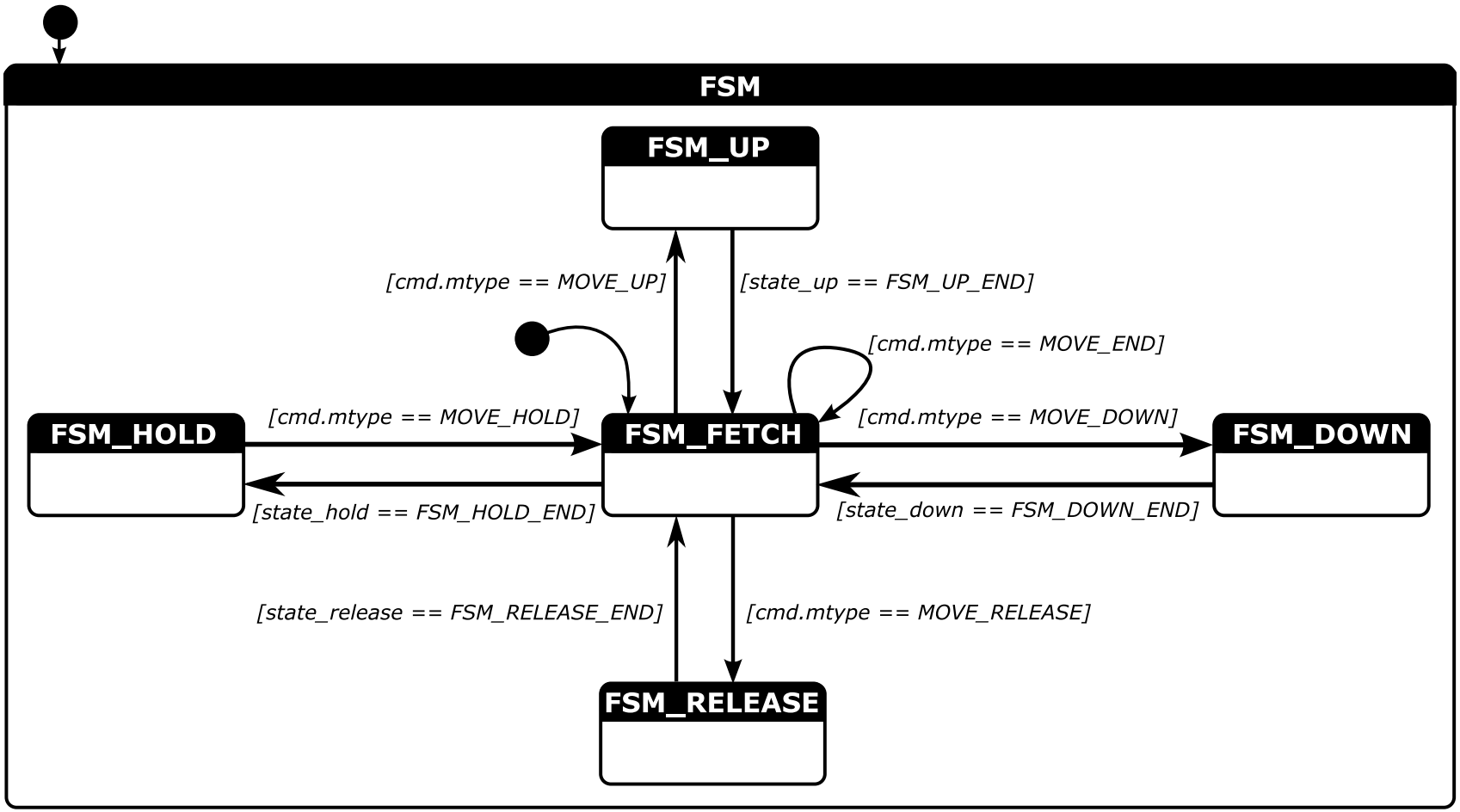
FSM

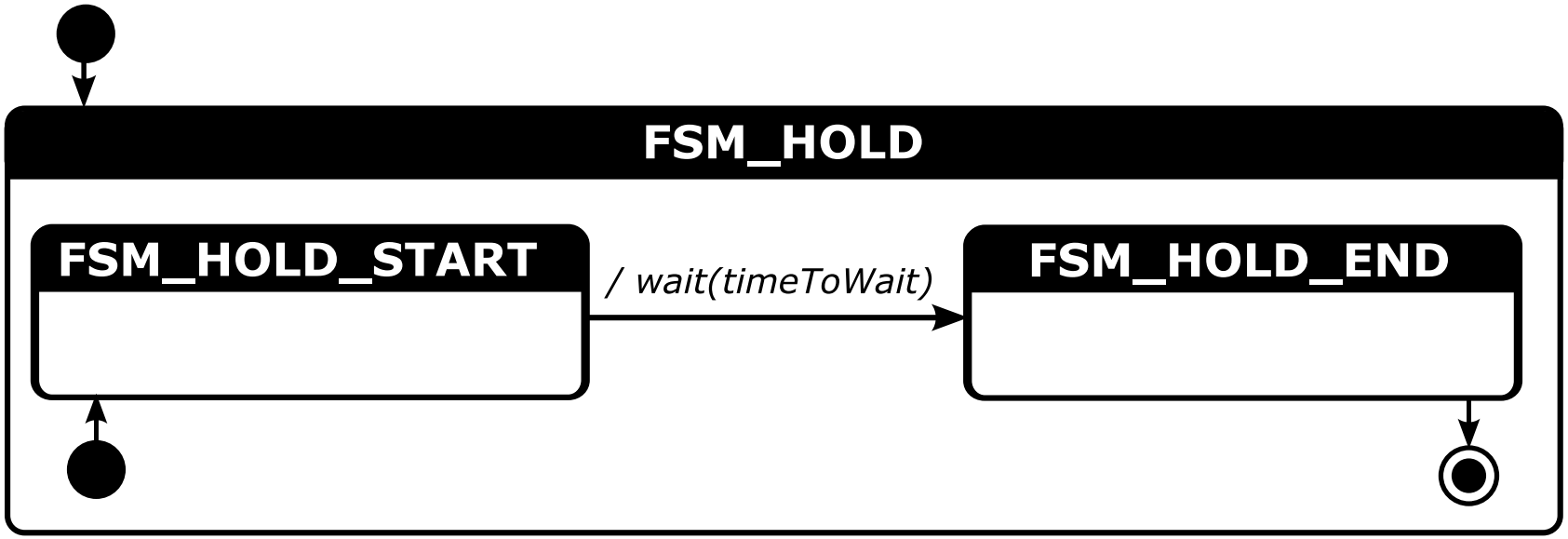

FSM
FSM_UP
[cmd.mtype == MOVE_UP]
[state_up == FSM_UP_END]
[cmd.mtype == MOVE_END]
FSM_HOLD
[cmd.mtype == MOVE_HOLD]
FSM_FETCH
[cmd.mtype == MOVE_DOWN]
FSM_DOWN
[state_hold == FSM_HOLD_END]
[state_down == FSM_DOWN_END]
[state_release == FSM_RELEASE_END]
[cmd.mtype == MOVE_RELEASE]
FSM_RELEASE

Zustandsdiagramm - FSM

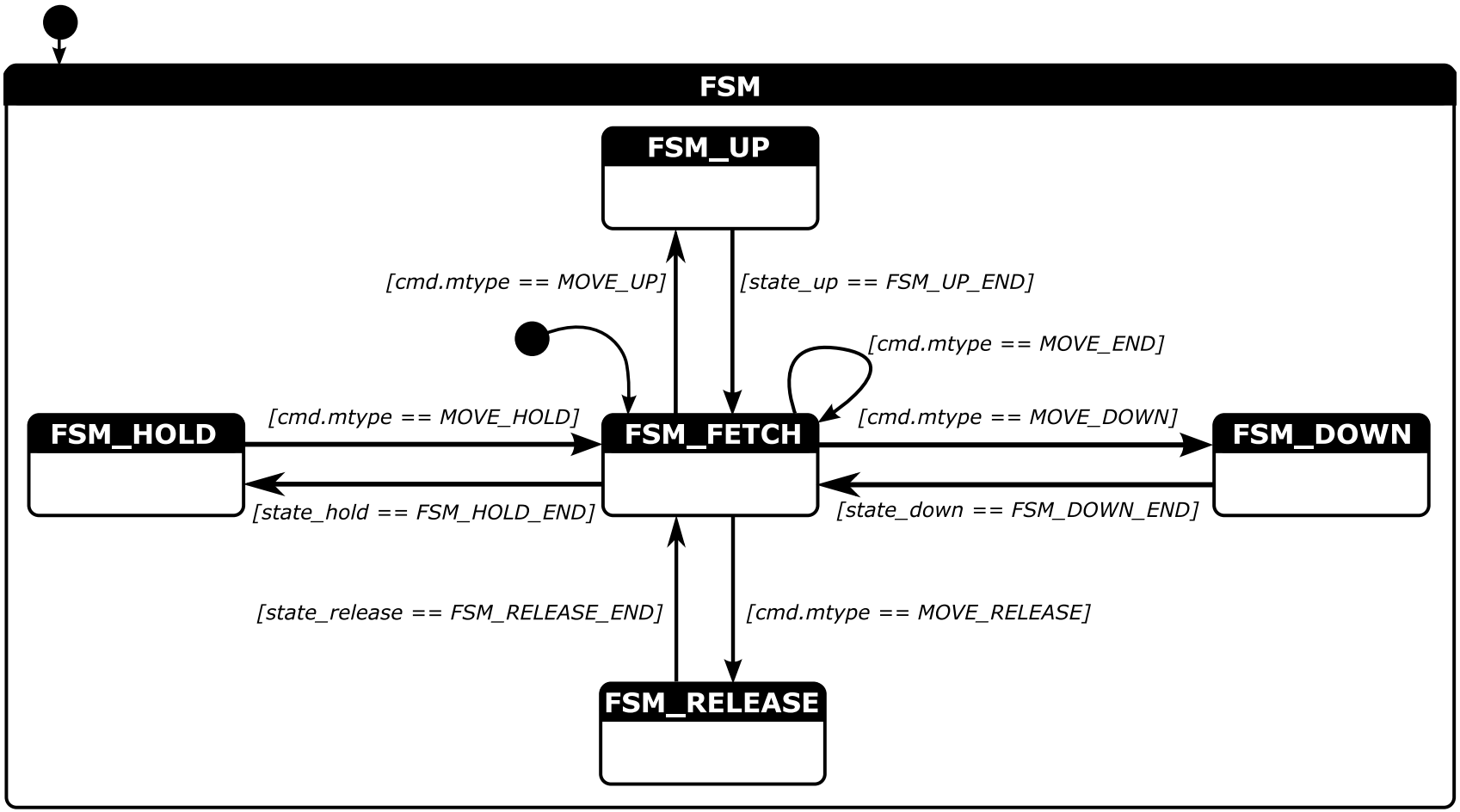


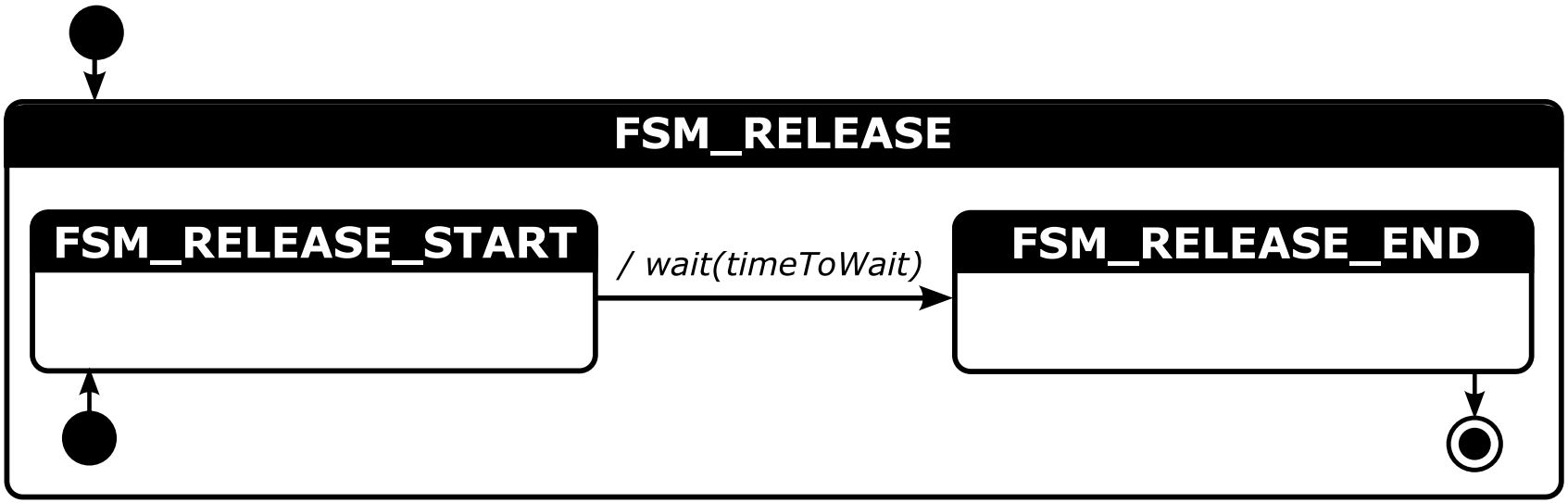
FSM


FSM
FSM_UP
[cmd.mtype == MOVE_UP]
[state_up == FSM_UP_END]
[cmd.mtype == MOVE_END]
FSM_HOLD
[cmd.mtype == MOVE_HOLD]
FSM_FETCH
[cmd.mtype == MOVE_DOWN]
FSM_DOWN
[state_hold == FSM_HOLD_END]
[state_down == FSM_DOWN_END]
[state_release == FSM_RELEASE_END]
[cmd.mtype == MOVE_RELEASE]
FSM_RELEASE


FSM_HOLD
FSM_HOLD_START
/ wait(timeToWait)
FSM_HOLD_END

FSM


FSM
FSM_UP
[cmd.mtype == MOVE_UP]
[state_up == FSM_UP_END]
[cmd.mtype == MOVE_END]
FSM_HOLD
[cmd.mtype == MOVE_HOLD]
FSM_FETCH
[cmd.mtype == MOVE_DOWN]
FSM_DOWN
[state_hold == FSM_HOLD_END]
[state_down == FSM_DOWN_END]
[state_release == FSM_RELEASE_END]
[cmd.mtype == MOVE_RELEASE]
FSM_RELEASE


FSM_RELEASE
FSM_RELEASE_START
/ wait(timeToWait)
FSM_RELEASE_END

Komposition

```
// Main Task
TASK(FSMTask) {
        ms_t timeToNextStep;
        do {
                timeToNextStep = FSM_Step();
        } while (timeToNextStep == 0);
        if (timeToNextStep > 0) {
                SetRelAlarm(SYSTEMALARM, timeToNextStep, 0);
        }
        TerminateTask();
}

// Interrupt Service Routine
ISR(PS_ISR) {
        PHOTOSENSOR_Interrupt();
}
```

Überblick

- Wir verwenden:
  - TASK (FSM)
  - ISR (Lichtschranken-Interrupt)
  - ALARM (Timer)

- Zusammenhang:
  - ISR-Handler aktiviert TASK
  - ALARM aktiviert TASK
  - TASK kann nur einmal aktiviert werden

Annahme:

- Timer-Interrupt tritt nur bei Ablauf des ALARM auf
- ISR selbst kann TASK unterbrechen, ist aber sehr kurz
- unser ISR-Handler wird nur aktiviert wenn der TASK ihn vorher **scharf geschaltet** hat
- ISR-Handler kann TASK also nicht unterbrechen

Gemessene Zeiten für Antwortzeitanalyse:

- Minimale Zwischenankunftszeit des TASK: 2000 μs
- Interrupt-Latenz: max. 41 μs, durchschnittlich 35 μs
- Interrupt-Häufigkeit: min. 36818 μs, max. 93971 μs
- WCET des TASK: 60 μs
- Deadline ca. 300 μs nach Aktivierung

Berechnung:

- Systemauslastung im schlimmsten Fall: ca. 3 %
- max. erlaubte Verzögerung eines Tasks: 240 μs
- Schlimmste Verzögerung:
  Interrupt-Latenz + *Interrupt-Zeiten die waehrend des TASKS auflaufen* $= 41 + 1 * 41 < 240 \mu s$

Falls weitere TASKs anderer Anwendungen gleichzeitig laufen, müssen alle möglichen gegenseitigen Verzögerungen berücksichtigt werden um die Echtzeitfähigkeit zu garantieren.

FAIL-SAFE

- **Funktion:**
  Im Fehlerfall wird das System in einen sicheren Zustand gefahren.
- **Umsetzung:**
  Fehlerfall wird erkannt, wenn Lichtschranken unterbrochen werden, die im Normalfall nicht unterbrochen werden sollten. FSM wird umgangen und das Projektil in den Ausgangszustand gebracht.

Überblick

Vorführung von

1. Gedämpfte Schwingung
2. FAIL-SAFE-Test
3. Fontaine