Portabler Bytecode Interpreter für 8-bit µController

Georg Icking-Konert*

Vortrag „Pi and More 9“ (Trier 06/11/2016)

* icking@onlinehome.de

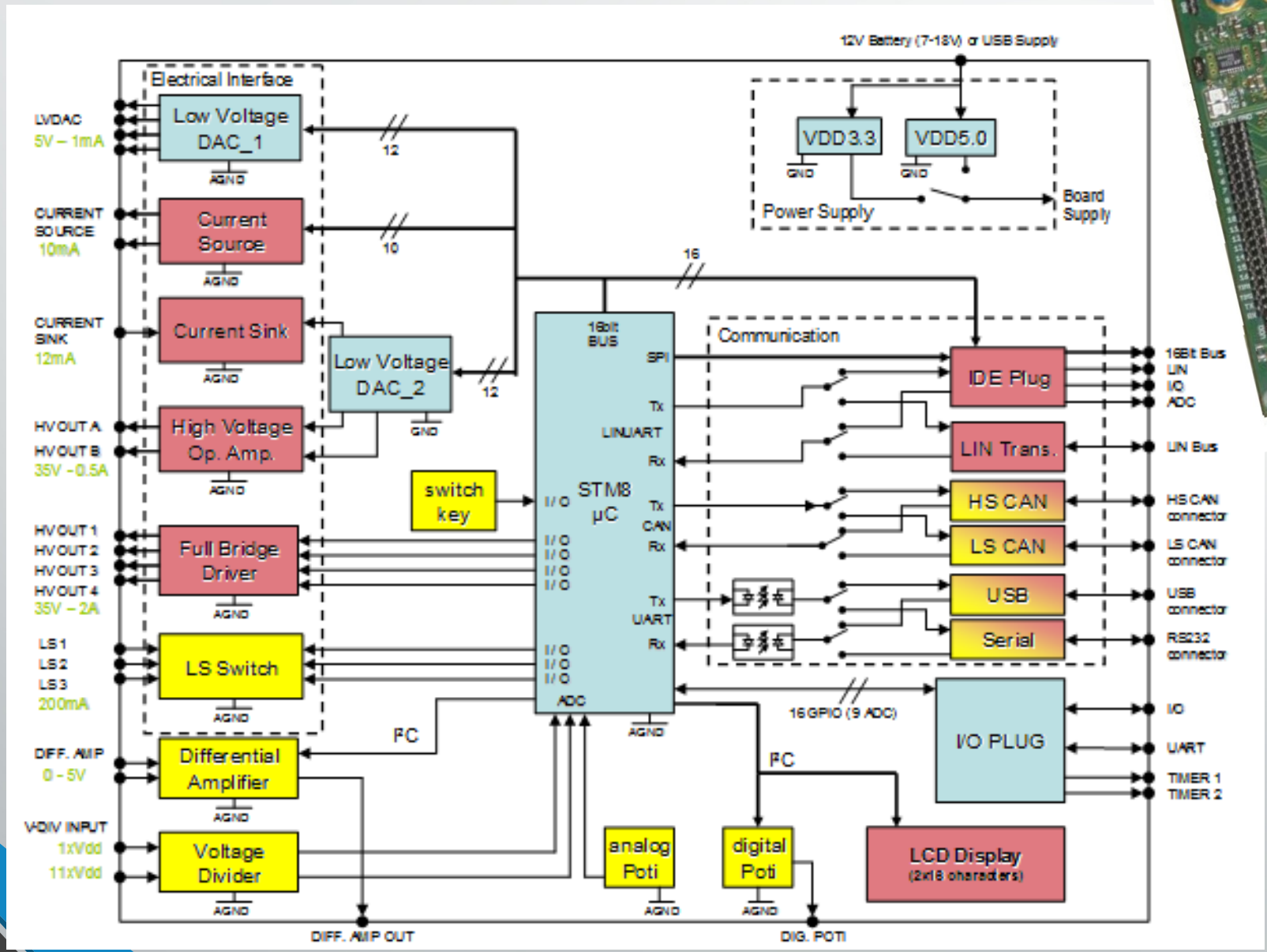
Motivation

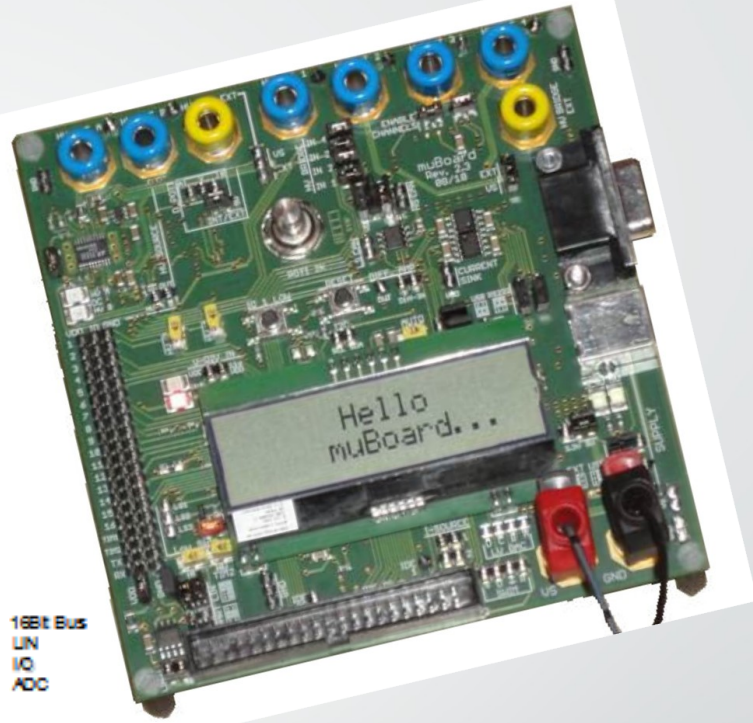
- Dank Arduino, Wiring und Co. gibt es *eigentlich* keinen Grund mehr für einen Bytecode Interpreter für low-end µController…
- …außer man sucht
  - den Spaß an der Herausforderung
  - IP-Schutz für programmierbare Embedded Systeme
  - Kapselung kritischer Hardware Zugriffe durch Benutzer („Sandboxing“) ohne Hardware-Abstraktion durch OS
  - …

Hintergrund

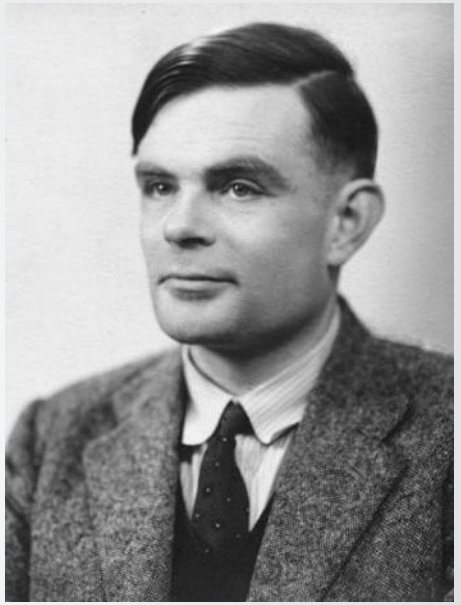
- Das *muBoard* wurde mit einem Bosch Kollegen (als U-Boot) zur Chip- und Schaltungsvalidierung entwickelt
- Bei einer Vorstellung war das Interesse der Kollegen groß, aber
  - Es gab damals keinen günstigen STM8 Compiler
  - Nur wenige beherrschten die Sprache C
  - Kaum jemand hatte einen Debugger zu Programmieren
- Daher entwickelte der Autor die Skriptsprache *muBatch*, welche
  - Keinen separaten Compiler benötigt
  - Wesentlich einfacher als C ist und die *muBoard* Hardware abstrahiert
  - Einen einfachen „Bootloader“ zum Flashen enthält

muBoard Hardware

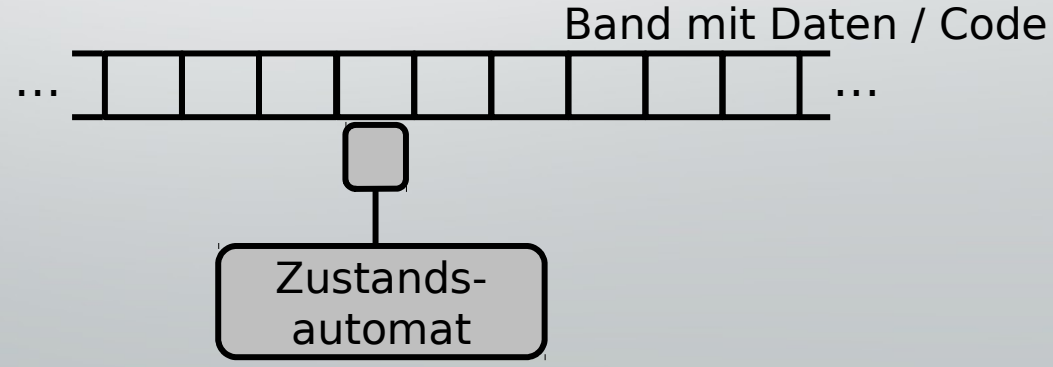




Turingmaschine



- Von Alan Turing 1936 vorgestelltes, theoretisches Rechnermodell zur Untersuchung von Berechenbarkeit
- Bestandteile:
  - zustandsbasierte Verarbeitungseinheit → CPU
  - Band mit Daten und Code → Speicher
  - Lese-/Schreibkopf zum Datentransfer → Bus
- Ist nur das ***Konzept*** eines Rechners, aber jedes berechenbare Programm ist als Turingmaschine darstellbar



Umsetzung in *muBatch*

- Aufgrund ihrer Einfachheit lässt sich eine Turingmaschine leicht auf low-end µControllern realisieren
- Konkrete Umsetzung in C:
  - Turingmaschine mit Code und konstanten Daten in Flash; variable Daten in RAM; Datenausgabe auf PC, Flash oder SD-Karte
  - Jeder Befehl besteht aus einem Befehlsindex (1B) und der “Startadresse” der zugehörigen Parameter im Band (2B)
  - Interpreter ruft aktuelle „Execute-Funktion“ aus Array mit Funktionspointer auf. Program-Counter und Daten werden innerhalb der Execute-Funktionen gelesen bzw. manipuliert
  - Kern der „virtuellen Turingmaschine“ besteht aus lediglich 4 Zeilen C-Code:

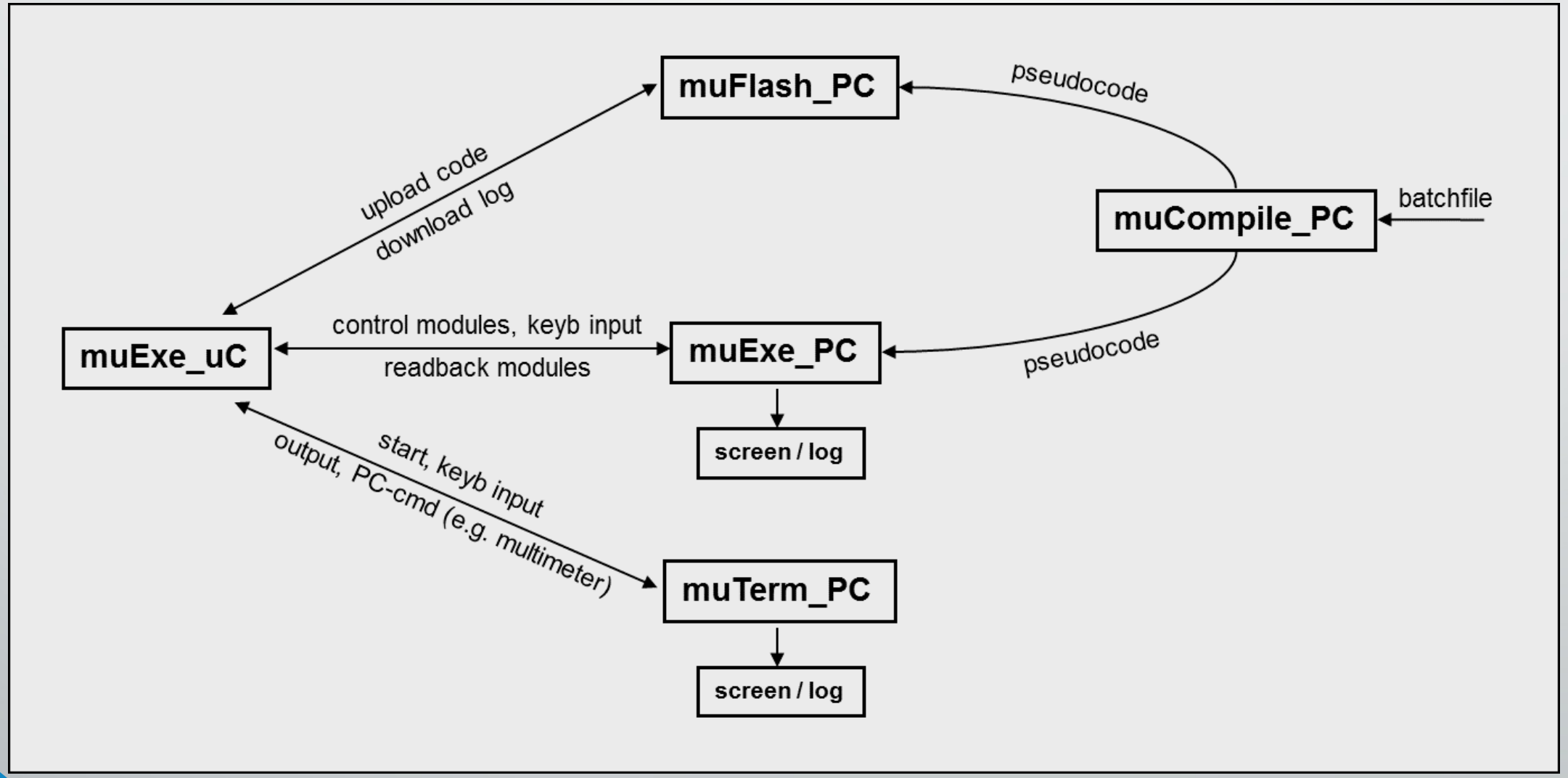
```
g_PC = 0;
do
  (g_cmd_pExe[g_code[g_PC]])();
while (g_PC < g_lenCode);
```

Ablauf

- Skript wird auf PC* editiert und kompiliert (*muCompile_PC*)
- Bytecode (=Bandinhalt) wird auf µController geladen (*muFlash_PC*)
- Bytecode wird von µController ausgeführt (*muExe_uC*)
- Datenausgabe auf PC (*muTerm_PC*). Erlaubt auch Zugriff auf externe Geräte, z.B. Multimeter oder Klimakammer via PC als USB-Gateway
- Optional:
  - Standalone Ausführung Bytecode auf µC ohne angeschlossenen PC
  - Ausführung Bytecode auf PC mit Steuerung µC per USB (muExe_PC)

* PC = für RasPi, Linux, Win, MacOSX,... Benötigt wird ANSI-C Compiler und USB-Zugriff per VCP bzw. POSIX Stream

Ablaufdiagramm



Demonstration

- Eine Vorführung sagt mehr als 1000 Folien…
- Beispiele:
  1. Berechne die ersten 20 Fibonacci-Zahlen ($F_n=F_{n-1}+F_{n-2} \Rightarrow 0, 1, 1, 2, 3,...$) per RPN mit Ausgabe in PC-Konsole und Datei (*muBoard* + PC)
  2. Spiel mir ein Lied… (PC only)
  3. Reaktionstest mit Zeitmessung (*muBoard* mit/ohne PC)

Einsatz in IC-Validierung

- Beispiel: Robustheit einer Chip Resetschaltung

```
; cycle Vs
for count 1 100000 1

  ; select random Vs and stepsize
  set Vnew _rnd 2000 _mod
  set dV _rnd Vnew Vold - _mod 1 _max
  if dV Vnew Vold - 2 / _abs <
    set dV 1
  endif
  if Vnew Vold <
    set dV dV -1 *
  endif

  ; ramp to new Vs
  for V Vold Vnew dV
    set_dac_raw HV_A V
  endfor
  set Vold Vnew

  ; if Vs>Vrth check if IC runs
  reset_exint io_1
  pause_ms 20
  get_adc_raw VD_HV a
  if a Vrth >
    get_exint io_1 n
    if n 12 <
      set err err 1 +
    endif
  endif

  ; show status every 100 cycles
  if count 100 _mod 0 =
    print 3 count TAB err LF
  endif

endfor
```

muBatch Befehle

- Gängige prozedurale Befehle:
  `function/call`, `for/endfor`, `if/else/endif`,...
- RPN-Rechner (wie HP-Taschenrechner):
  `set target a b + c d + *` = (a+b)*(c+d)
- Spezifische µController Befehle:
  `get_gpio`, `send_receive_spi`, `get_adc`,...
- Standard PC-Befehle:
  `get_clock`, `print`, `get_key`, `console`,...
- „PC-Gateway“ Befehle → kalibriertes Messmittel:
  `get_multimeter`, `set_climate`

muBatch Befehle (2)

- Aktuell (v1.18.2) sind >100 Befehle implementiert
- *muBatch* Language Manual verfügbar

category	commands	description
general	comment	line and block comments
	var	declaration of (global) variable
	set	value assignment and RPN calculator
	if / else / endif	conditional flow
	for / endfor	for loop
	while / endwhile	while loop
	label / goto	unconditional jump
	function / return / call / break	subroutine definition and calling
	end	terminate flow
	nop	wait briefly
	get_error	get last error, e.g. timeout
	get_key	get last keyboard input
	user_cmd	placeholder for custom routine
log	console	routines for controlling console
	print	print to log, console, µC flash, and/or SD-card
	print_lcd	print to LCD (if connected)
time meas.	pause_ms / pause_us / pause_ns	wait specified delay
	config_time / get_time	control of RTC, and time measurement
	interval / endinterval	perform action in regular intervals
	print_date	print date and time (PC required)

category	commands	description
muBoard HW	config_gpio / set_gpio / get_gpio / toggle_gpio / wait_gpio / glitch_gpio	routines for µC GPIO control
	get_adc_raw / get_adc_cal	get raw or calibrated ADC values
	set_dac_raw / set_dac_cal	set raw or calibrated DAC values
	save_calibration	save offset and slope for channel calibration
	config_exint / reset_exint / get_exint	routines for control of µC external interrupts
	set_pwm / get_pwm	generate or measure PWM signal
	set_bridge	set state of half bridge channel
	set_dig_poti	configure digital potentiometer
	set_lowside	set state of lowside channel
	config_diff_amp	configure differential amplifier
	reset_HW	reset all HW modules
	beep	beeper control
communi-cation	config_uart / send_receive_uart	control of µC UART interface, optionally with K-Line
	send_receive_RTMM / send_receive_LIN	communication via RTMM or LIM
	config_spi / send_receive_spi	control of µC SPI interface
	start_i2c / stop_i2c / send_i2c / receive_i2c	control of µC I2C interface
	config_can / send_can / receive_can	CAN bus communication
	send_sync / receive_sync	routines for muBoard synchronization
PC gateway	send_receive_PC (plain or LIN)	communicate with devices connected to PC
	get_multimeter / print_multimeter	control of multimeter connected to PC
	get_climate / set_climate	control of climate chambers connected to PC

Aufbau Befehle

- Jeder Befehl besteht aus 3 C-Funktionen:
  - **read_*cmd*()** liest in *muCompile* die Funktionsparameter aus Skript und speichert sie in das Daten-Array
  - **pass2_*cmd*()** optimiert in *muCompile* die Adressen für Jump-Befehle, z.B. *for/endfor*, *while/endwhile*, *if/endif*. Dummy für andere Befehle
  - **exe_*cmd*()** führt in *muExe_uC* bzw. *muExe_PC* den Befehl aus, konkret:
    - Lies Parameter aus Daten-Array
    - Führe Befehl mit Parametern aus
    - Ändere ggf. Zustandsvariablen in RAM, z.B. in Math-Parser
    - Erhöhe Program-Counter bzw. setze auf Sprungadresse
    - Zugriffe auf µC (IOs, UART, SPI,…) und PC (Uhrzeit, USB-Gateway,…) per HW-Layer abstrahiert → identische Schnittstelle auf µC und PC

Vor-/Nachteile *muBatch*

- Vorteile
  - Extrem einfache Sprache, z.B. nur globale Variablen, ein Datentyp, single-Task
  - Einfachste PC-Installation (kopiere 3 Binaries)
  - Tests taktgenau reproduzierbar (bei deaktiviertem 1ms ISR) → Timing / Reproduzierbarkeit von Tests auf 1/16MHz genau
  - Einfach auf andere µCs portierbar → IP Schutz für programmierbare low-end Systeme, Sandboxing,… (siehe „Motivation“)

- Nachteile
  - auf STM8@16MHz >10µs pro Befehl → ~100x langsamer als Maschinencode
  - Ausschließlich 32b signed Integer, kein float
  - *muBatch* ist eine „One-Man-Show“ ohne Verbreitung außerhalb Bosch
  - 8bit Befehls-ID, 16bit „Adressen“ → bis zu 255 Befehle & je 64kB Code/Daten

Portierung auf andere µCs

- Aktuell ist *muBatch* nur für STM8 implementiert, konkret *muBoard*
- Durch die generische Struktur ist eine Portierung auf andere µController aber einfach
- Benötigte Features des µC bzw. der Toolchain:
  - Ab ca. 32kB Flash / 2kB RAM, je nach Anzahl Befehle (hier 128kB / 6kB)
  - ANSI-C Toolchain, ideal mit graphischem Debugger (hier Cosmic / STVD)
  - In-Application-Programming zum Hochladen des Bytecodes
  - Zugriff von PC z.B. über USB ↔ UART Adapter (hier via FT232)

Zusammenfassung / Ausblick

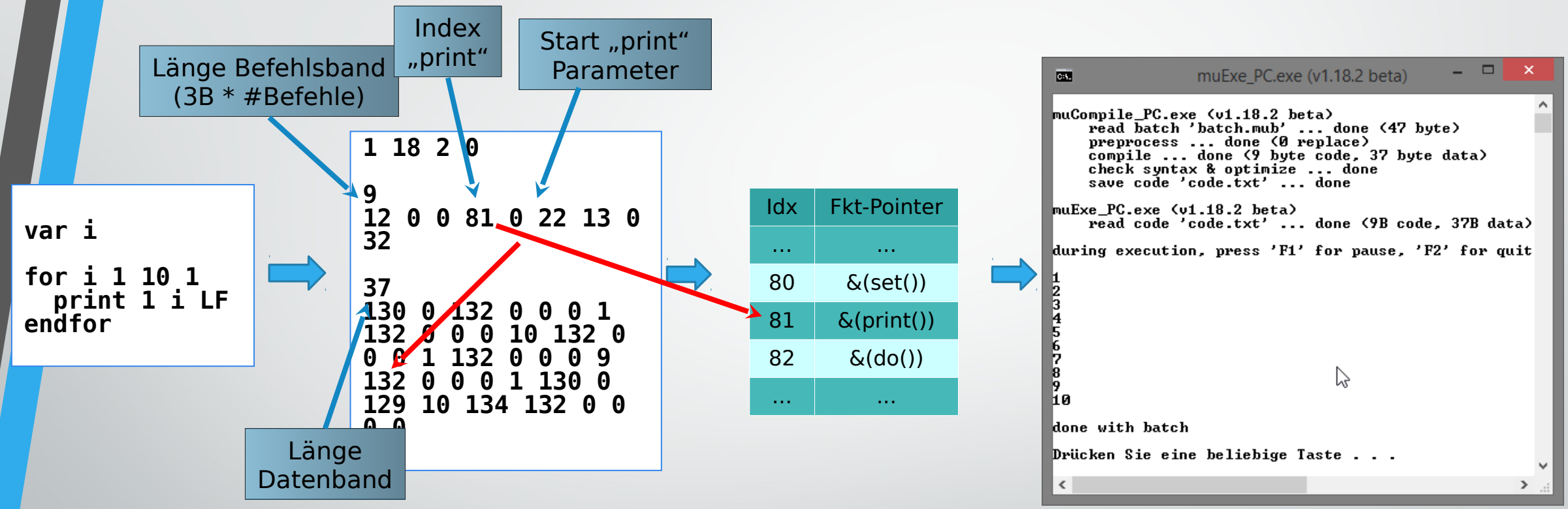
- Vorstellung eines portablen Bytecode Interpreters (aka *muBatch*) für low-end 8-bit µController am Beispiel STM8
- Bereits seit Längerem zur Chip-Validierung bei Bosch im Einsatz → Kinderkrankheiten inzwischen behoben
- Nebenprodukte von *muBatch* sind ein STM8 Serial Programmer sowie SDCC C-Templates für STM8 (beide auf Github verfügbar)

- Wegen Verfügbarkeit von OS-Compilern (z.B. gcc, sdcc) und OS–Plattformen (z.B. Arduino) planen wir auf C umzusteigen. Fokus auf 100% Echtzeitfähigkeit → kein OS, keine Hintergrund-Tasks
- Aktuell keine Planung, *muBatch* zu portieren. Wenn aber ehrliches Interesse besteht, bitte per eMail melden…

Vielen Dank und noch viel Spaß!

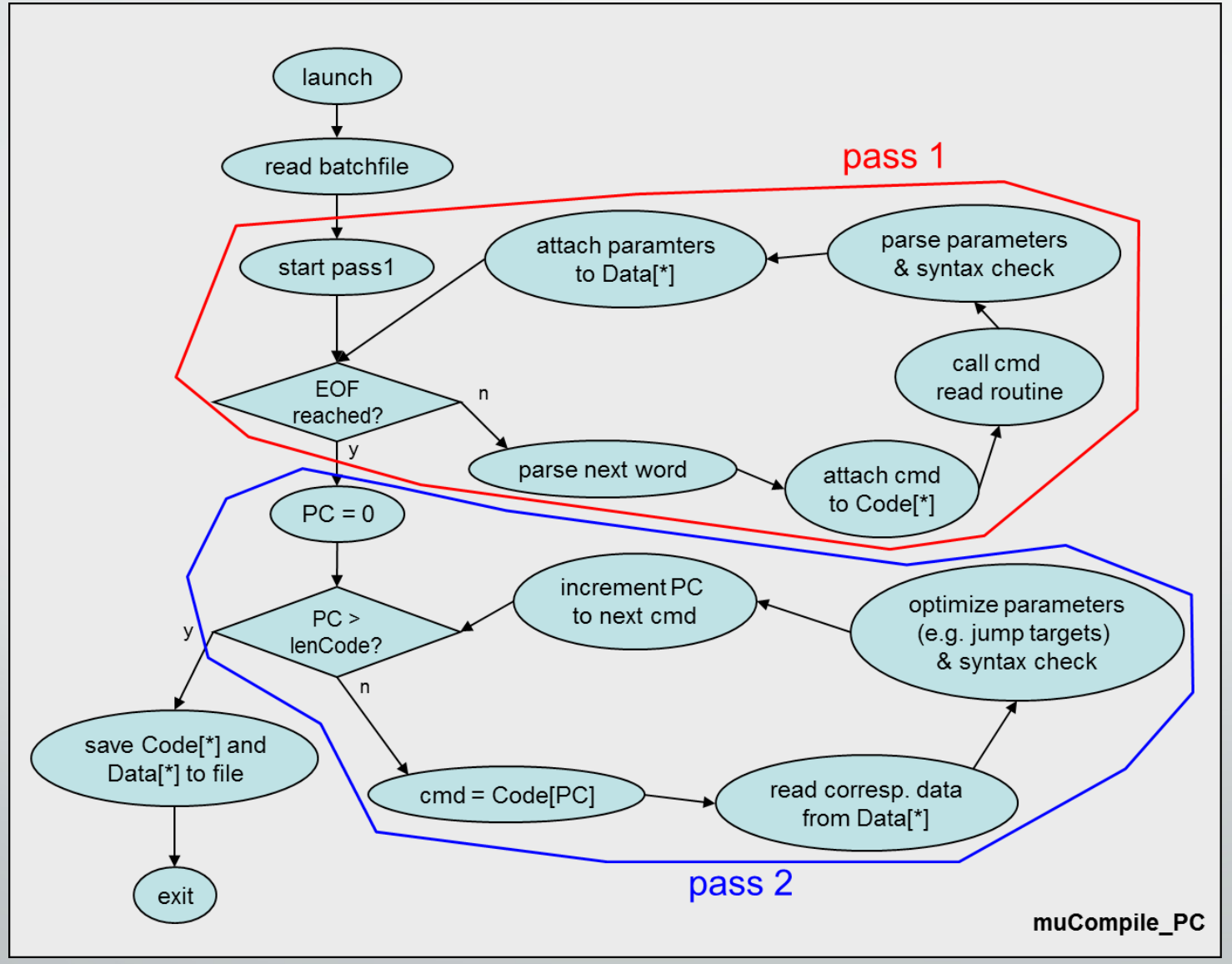
Anhang

Skript → Bytecode → Turingmaschine

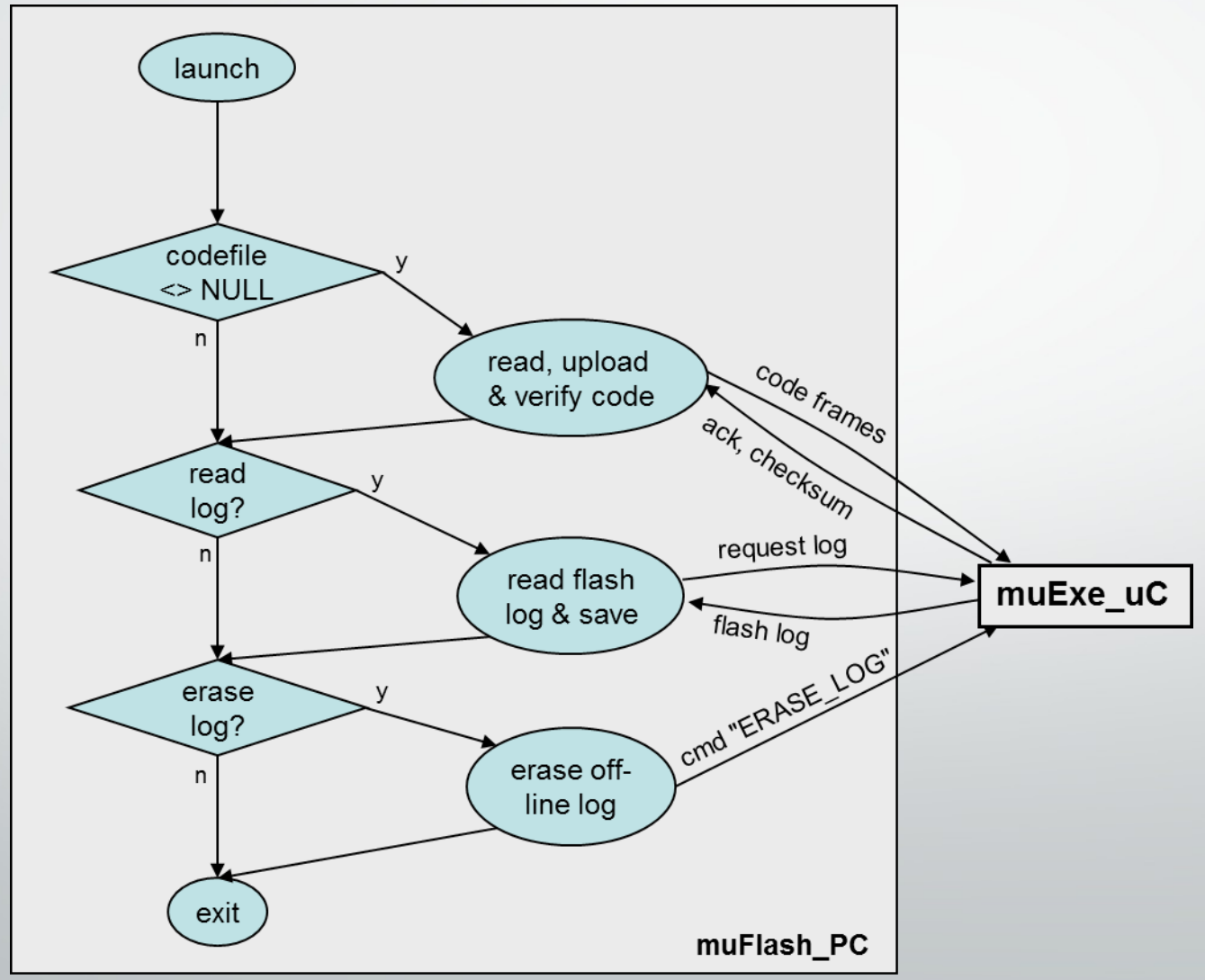


Skript ***Bytecode*** ***Ausführung*** ***Ausgabe***

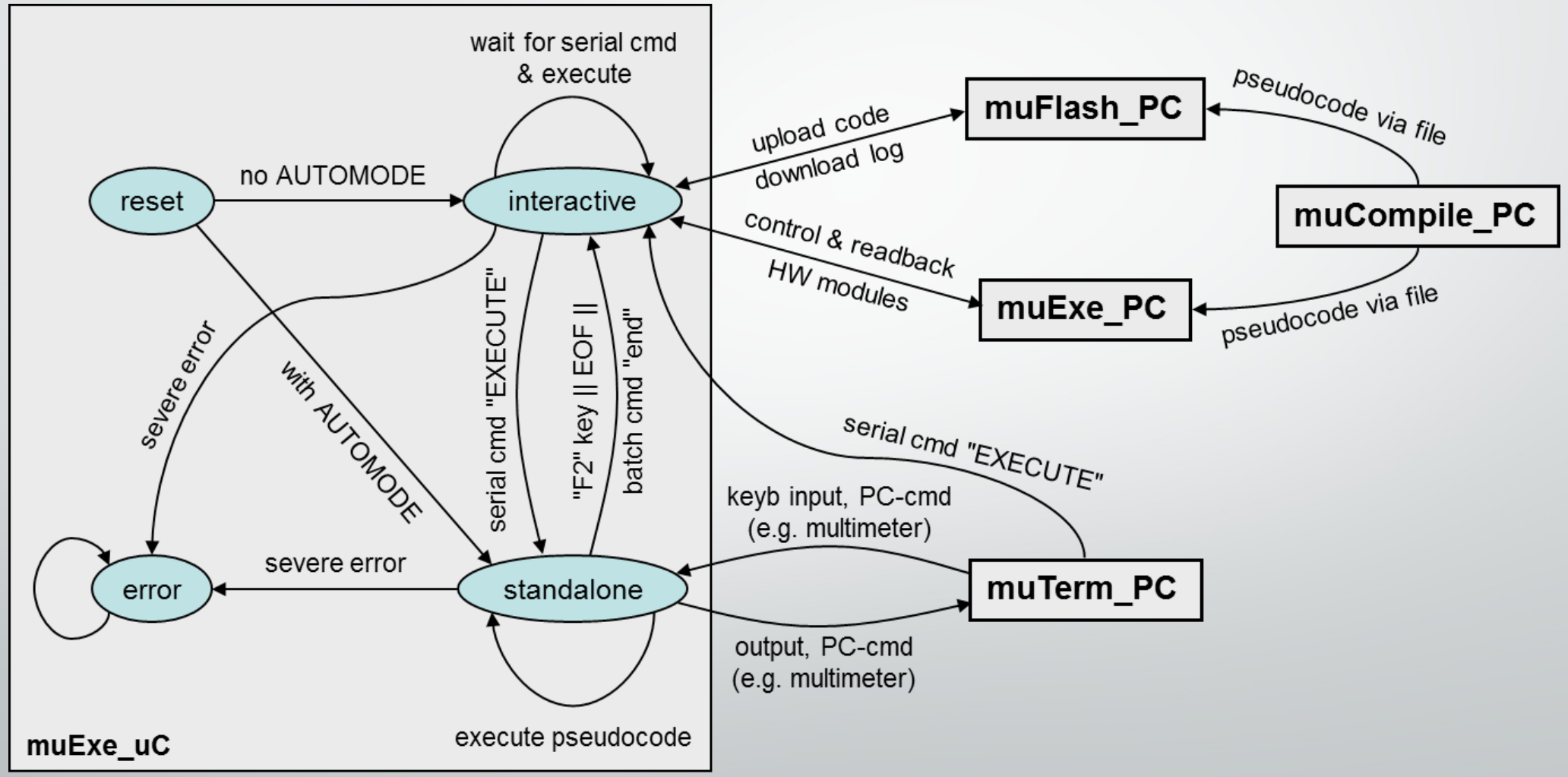
Flussdiagramm *muCompile_PC*



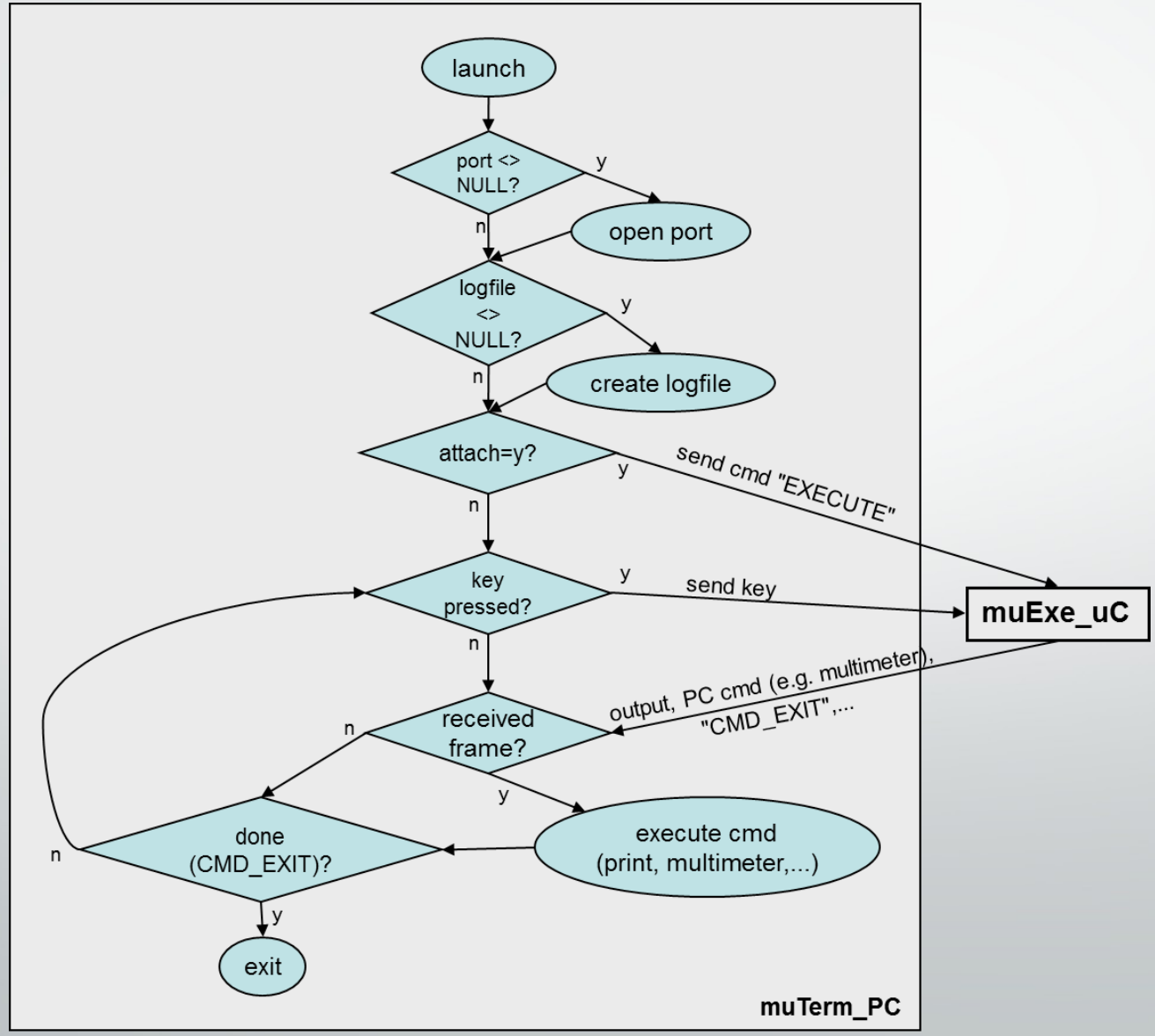
Flussdiagramm *muFlash_PC*



Flussdiagramm *muExe_uC*



Flussdiagramm *muTerm_PC*



Flussdiagramm *muExe_PC*