Abschlusspräsentation zur Bachelorarbeit

Apple][Emulation on an AVR Microcontroller

Emulation eines Apple][auf einem AVR Mikrocontroller

Maximilian Strauch

UNIVERSITÄT KOBLENZ · LANDAU

unikorn AG Rechnernetze

Der Apple][

Ein Blick in die Geschichte ...

1. April 1976:
Gründung Apple, Apple I

April 1977: Apple II

1978: Apple II Plus

1978: Apple II Europlus

1983: Apple //e

1984: Apple //c

1985: Apple //e Enhanced

1986: Apple IIGS

1987: Apple //e Platinum

1988: Apple //c+

November 1993:
Einstellung nach > 2
Millionen verkauften
Geräten + Nachbauten

1980

1990

Mai 1980 - April 1984:
Apple III: entwickelt ohne Steve Wozniak
(ein Flop; der Apple][wurde bevorzugt)

24. Januar 1984 - Heute:
Apple Macintosh Serie veröffentlicht

1981 - 1987:
IBM PC wirbt Kunden ab wegen Apple III Miserfolg

Der Apple][im Überblick

- Prozessor: **MOS 6502** bei 1,023 MHz
- Video: NTSC (PAL) Composite Out
- (Arbeits-)Speicher: **64KB** davon 16KB ROM, 48KB RAM
- Sekundärer Speicher: 5½“ Disketten und Kassetten
- Peripherals: eigene ASCII Tastatur, Game Paddle, 8 Peripheral Card Slots

Quelle: http://apple2history.org/wp-content/gallery/advertisements/a2original.jpg

Videoausgabe

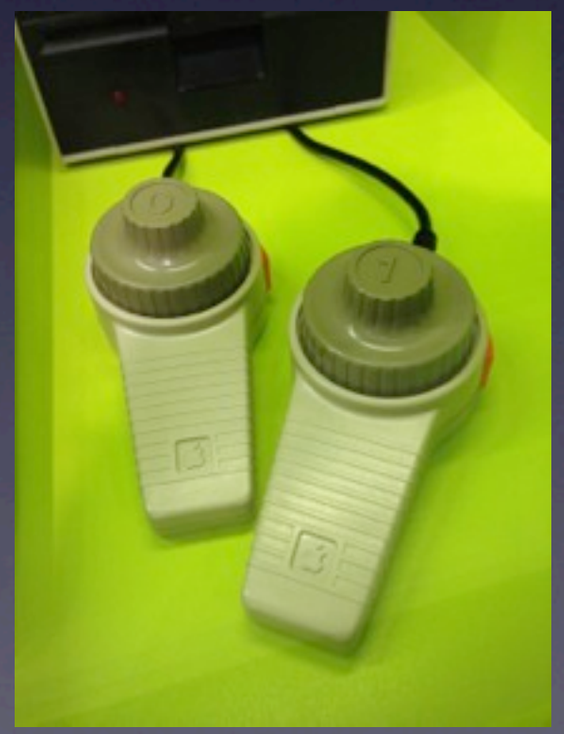
Mixed mode mit dem Integer BASIC Prompt

HiRes, Color Demo

- **Textmodus**
  - 960 Zeichen (5x7 Pixel) in 40x24 Raster, 2 Farben
- **„Low Resolution graphics mode“ (LoRes)**
  - 1.920 7x4 Pixel Blöcke mit 16 Farben
- **„High Resolution graphics mode“ (HiRes)**
  - 280 x 192 Bildpunkte, *bis zu 6 Farben*
- **Mixed mode**
  - 20 „Zeilen“ Grafikmodus (LoRes / HiRes; 160 Pixel) und 4 Zeilen Textmodus am unteren Rand

Eingabegeräte

- Tastatur mit 52 Tasten, nur Großbuchstaben
  - Sondertasten: **REPT, RESET**
- Zusätzliche „Eingabe“ über Game Paddles mit digitalen Eingängen und einem Potentiometer an einem einfachen ADC („manuell“)



„Betriebssystem(e)“

- **System Monitor** (*Steve Wozniak*)
  - Systemcalls & Betriebssystemroutinen
  - Mini-Assembler „IDE“
  - SWEET-16: simulierte 16-Bit CPU für 16-Bit Arithmetik
- **Integer BASIC** *(Steve Wozniak)*
  - Nur Integer-Arithmetik
  - Sehr schnell & beliebt für Spiele
- **Applesoft BASIC** *(Microsoft)*
  - Fließkomma-Arithmetik, langsam

MOS 6502

*Aufnahme des **Die**s des MOS Technology 6502 Prozessors.*

MOS Technology 6502

- Konkurrenzprodukt zum Intel 8080 oder Motorola 6800
- Nur $25 im Vergleich zu ~$150 für M6800 / i8080
- 8-Bit Architektur mit 16-Bit Datenbus, max. 64KB addressierbar
- 151 (aus 256) Instruktionen mit 13 verschiedenen Adressierungsmodi
- Register: Accu, X, Y, SP, P (8 Bit) und PC (16 Bit)
- Interrupts: RESET, NMI, BRK, IRQ mit Startvektoren ab `0xfffa`
- „Zero Page“ beschleunigt

6502 Design-Team (Quelle: William D. Mensch, Jr. April 2006)

Das Design wird von Hand gezeichnet und übertragen (Quelle: Intel Technology Journal - 1st Quarter 2001)

Einsatzgebiete des 6502

Atari 8000
(1979)

Apple I, Apple][Serie
(1977 - 1993)

Nintendo NES
(1983 - 2003)

Ohio Scientific
Challenger 4P

Commodore 64
1982 - 1994

Nintendo Famicon
(1983 - 2003)

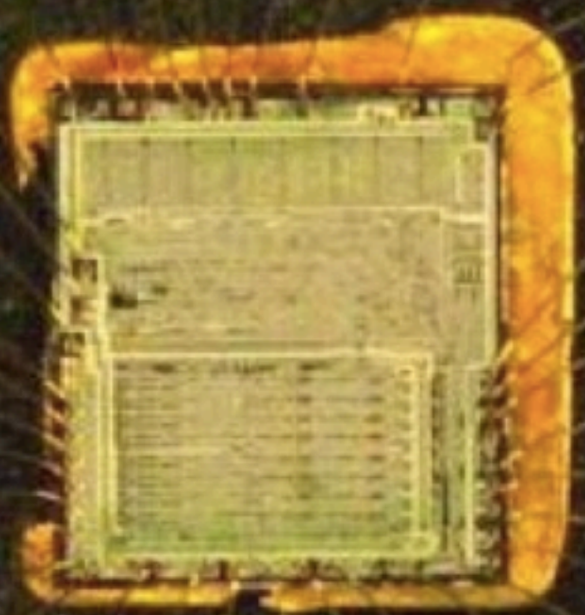
BBC Micro
(1981)

Atari 2600
(1977)

Der 6502 wird auch heute noch produziert (~5 €/Stk.) und in MOTU Audio Geräten, medizinischen Geräten oder Scannern verwendet.

Quelle: visual6502.org

Zielsetzung



***Die**, der ungehäuste Teil eines Wafers, des 6502, der den integrierten Schaltkreis des 6502 enhält. Die Anschlüsse am Die sind mit hauchdünnem - etwa 19 µm im Gegensatz zu 40 - 120 µm eines Haares - Wire Bonds (Golddraht) an die Anschlüsse des DIPs angeschlossen.*

Zielsetzung dieser Arbeit

„Ziel der Bachelorarbeit [...] ist es, ***die Funktionalität*** *dieses alten 8-Bit Systems* ***auf einem Mikrocontroller*** *der Atmel AVR-Familie zu implementieren.“*

Anforderungsprofil

Zielsetzung dieser Arbeit

„Ziel der Bachelorarbeit [...] ist es, ***die Funktionalität*** *dieses alten 8-Bit Systems* ***auf einem Mikrocontroller*** *der Atmel AVR-Familie zu implementieren.“*

Anforderungsprofil

Zielsetzung dieser Arbeit

„Ziel der Bachelorarbeit [...] ist es, ***die Funktionalität*** *dieses alten 8-Bit Systems* ***auf einem Mikrocontroller*** *der Atmel AVR-Familie zu implementieren.“*

Anforderungsprofil

- Emulation auf einem Computer ist einfach. Frage: gilt das auch auf einem Mikrocontroller?
- Der AVR ist nur 19,55 mal schneller als der Apple][- ein moderner Computer mindestens **1.759,5** mal (i5@1,8GHz) schneller als der Apple][und **90** mal schneller als der AVR
- Warum kein theoretisches Rechenmodell aufstellen?
  - Systemkomplexität → keine realistische Aussage mehr möglich
  - Dokumentierte und optimierte MOS 6502 CPU Emulation nötig für Sprachausgabe auf dem AVR
  - Zusätzlicher Lerneffekt aus der Hardwareimplementierung

Das Ziel im Detail

- 6502 CPU Emulation auf dem AVR ***ohne*** den BCD-Modus
- Ansteuerung eines Displays mit GDRAM
- Entwurf einer Tastatur
- Emulation des Apple][Speichers
- Von Neumann Architektur auf Harvard Architektur
- Software-Lademöglichkeit, um Programme laden zu können
- ***Schließlich:*** Umsetzung eines funktierenden Handheld Prototypen

Umsetzung

… von Problemen und Lösungen*

Schaltplan des entworfenen und implementierten Apple][Emulators. Auf der linken Seite ist der für die Emulation zuständige Mikrocontroller abgebildet und auf der rechten Seite der für die selbst entwickelte Tastatur zuständige Mikrocontroller abgebildet.

* in Auszügen.

Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)

Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)

Atmel AVR Mikrocontroller / Hardware

Struktur der Softwareimplementierung

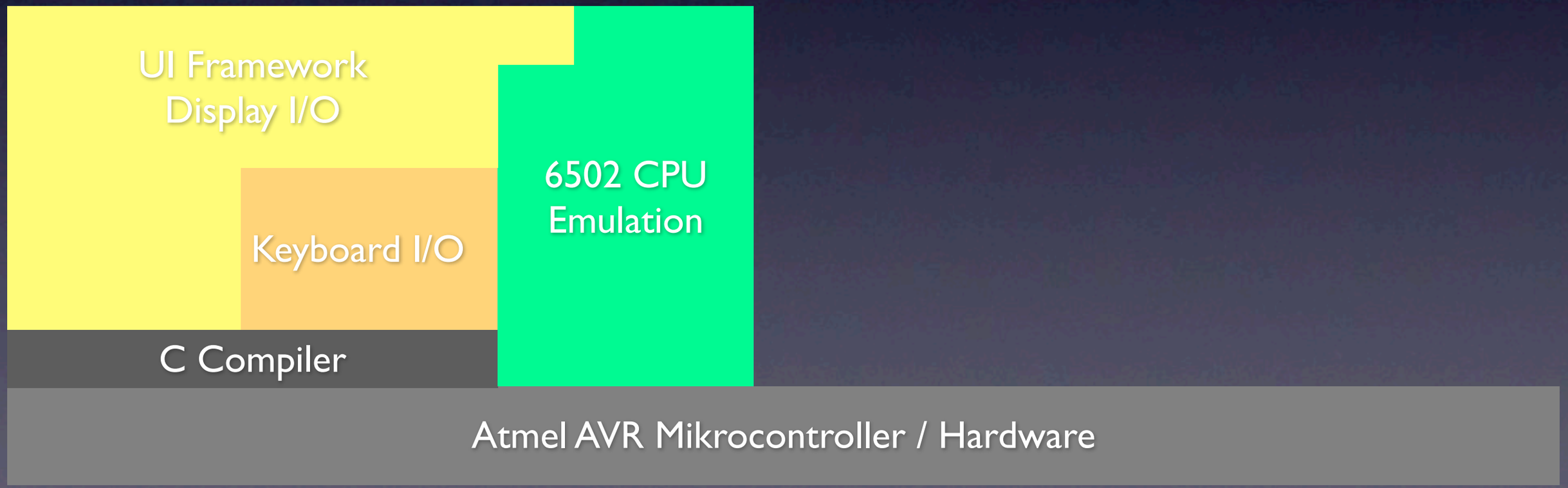
- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)

Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)

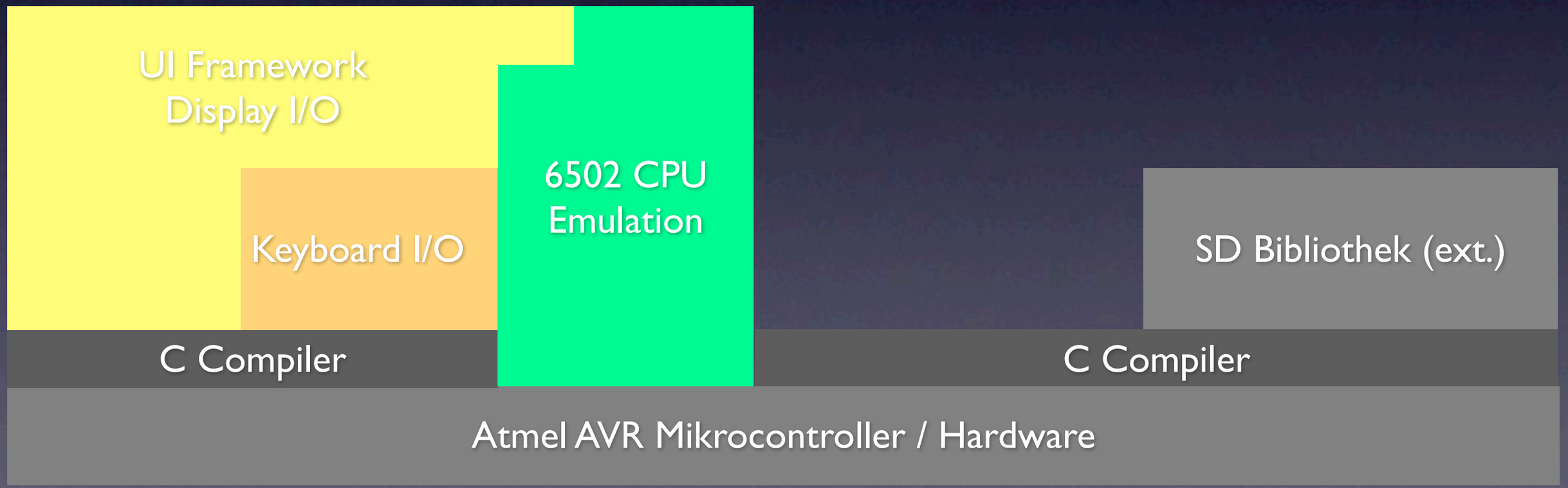
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



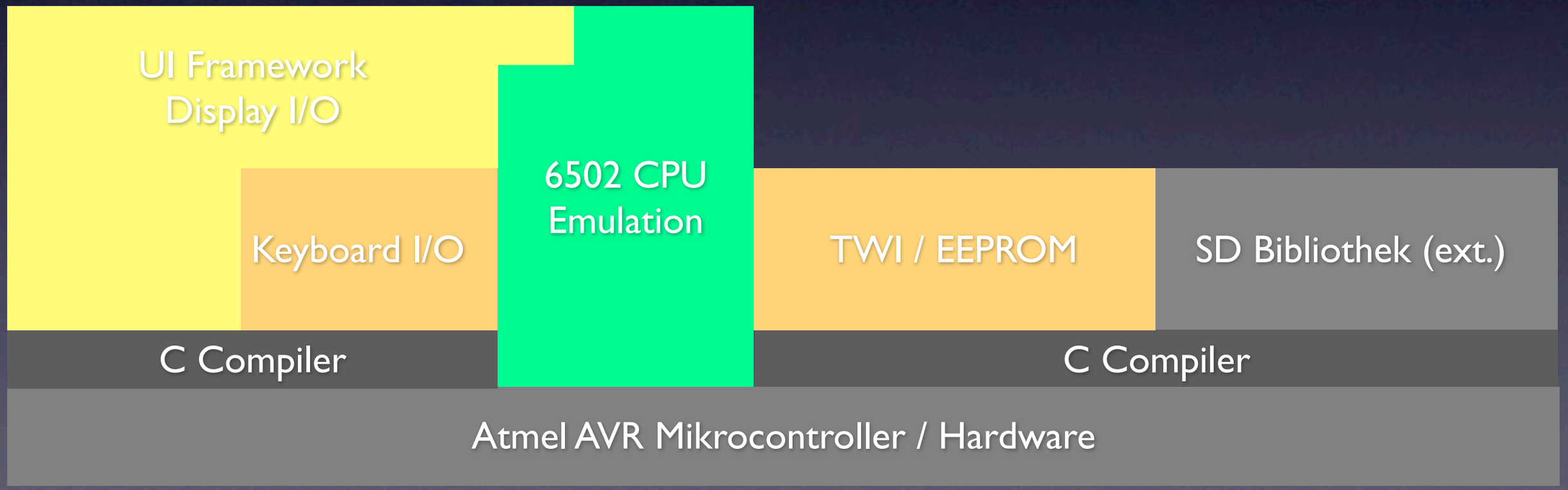
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



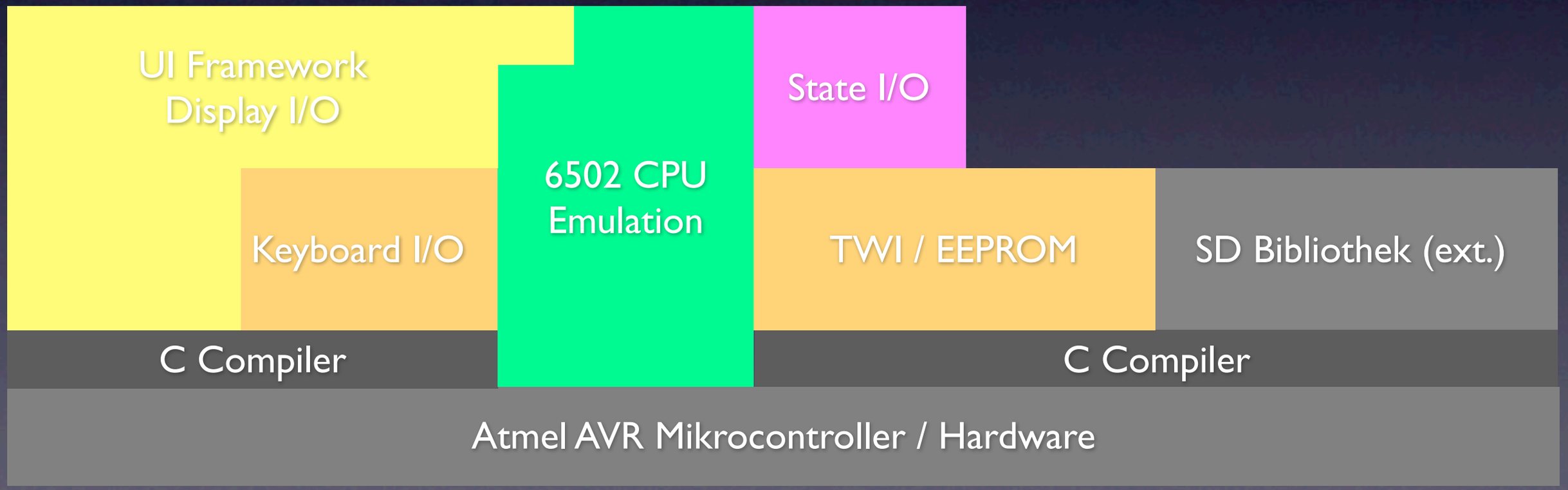
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



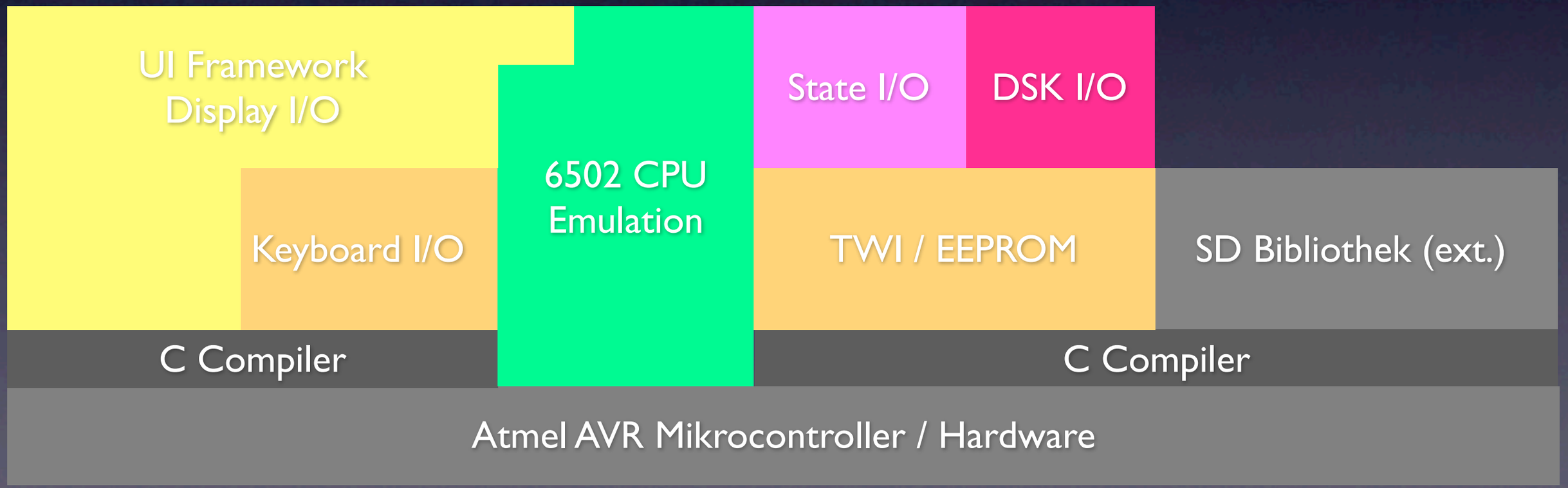
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



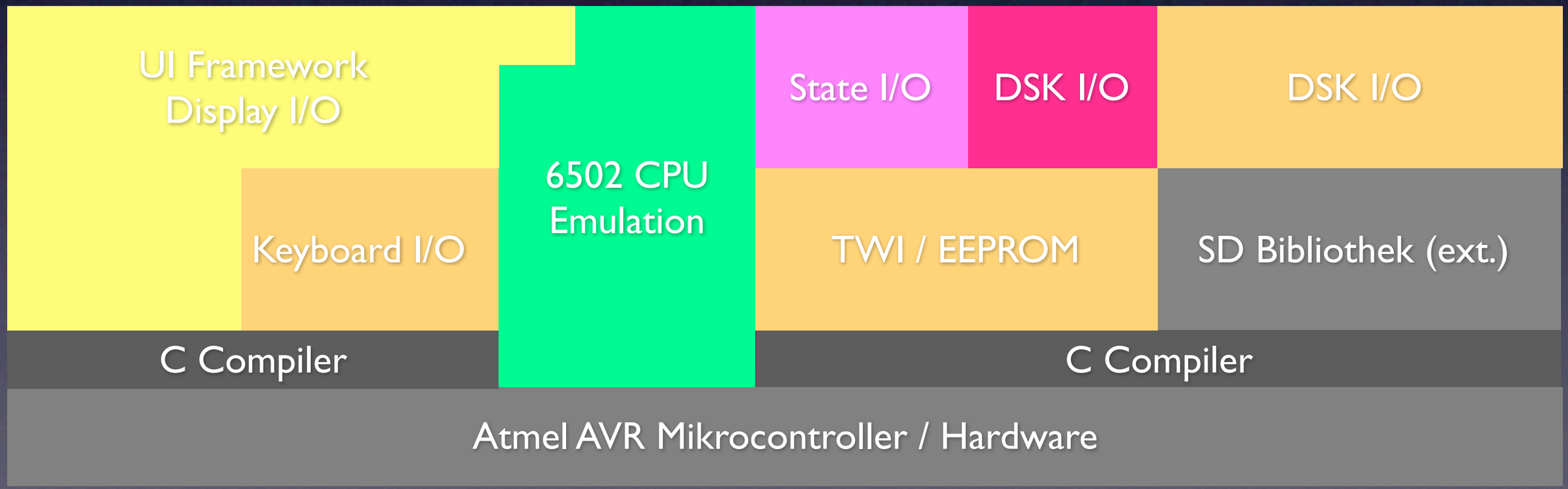
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



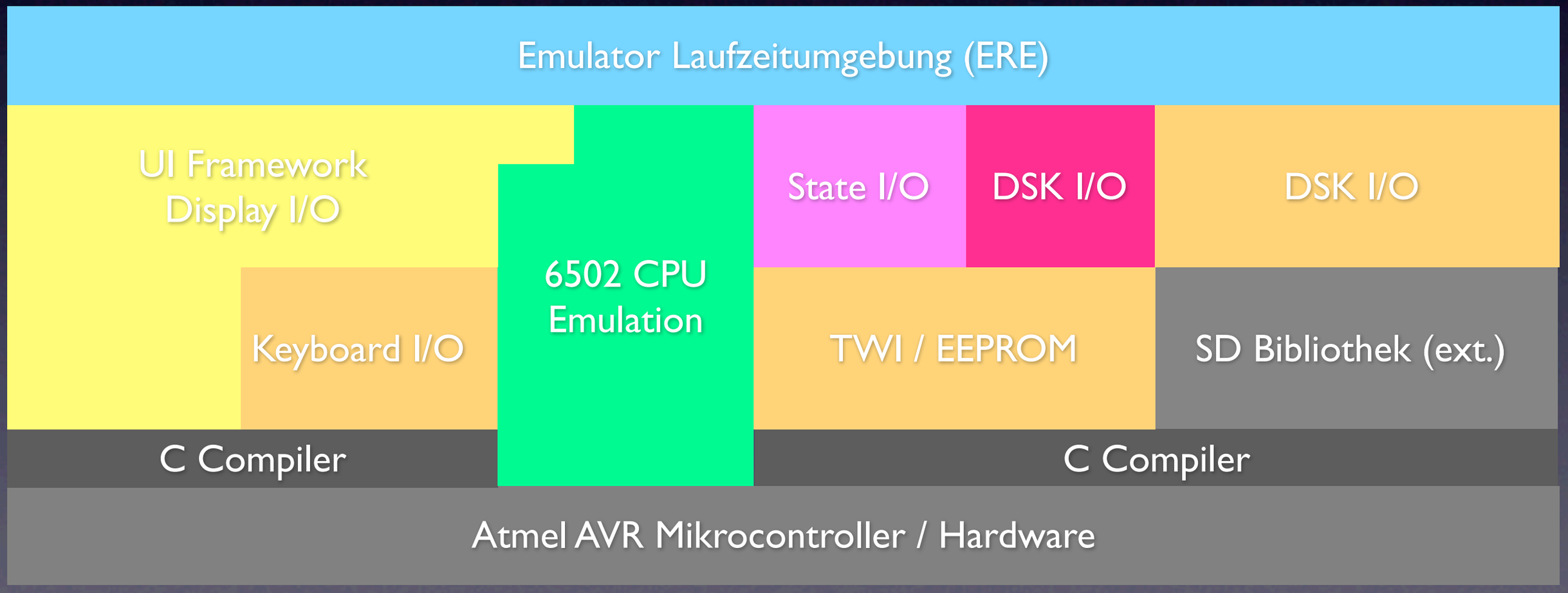
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



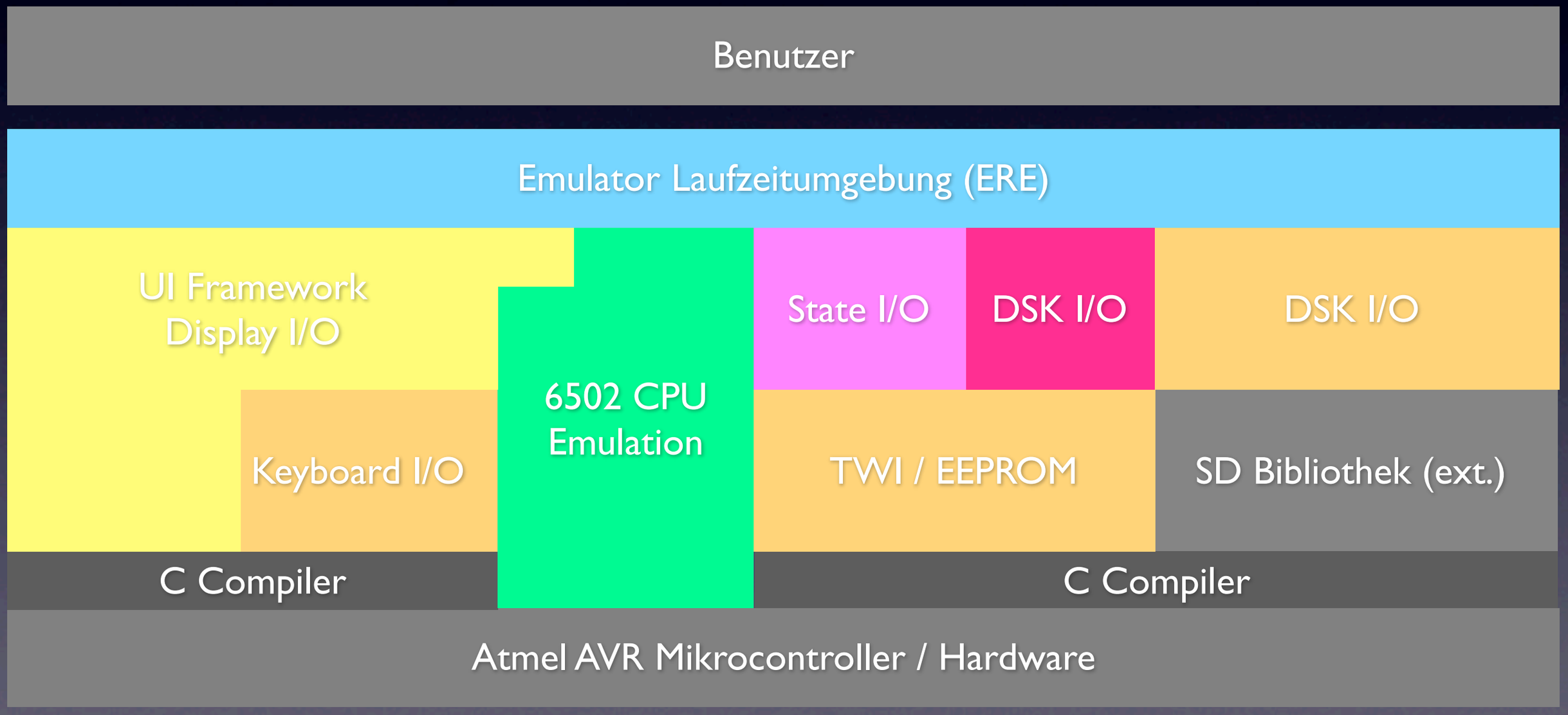
Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



Struktur der Softwareimplementierung

- Projekt aufteilen in Module für Übersicht über Abhängigkeiten und getrennte Entwicklung; insgesamt ~9.000 Zeilen Code; alle Module selbst geschrieben (außer grau hinterlegte)



6502 Emulation - Wie sieht so etwas aus?

- Eine Funktion, die in einer while-Schleife die Zyklen abarbeitet
- Großes Switch-Statement: für jede Instruktion ein Fall (151 Fälle + 1)
- Für jede Instruktion gilt folgendes Schema:

1. *Instruktionsargument(e) holen*
2. Semantik ausführen
3. *Daten zurückschreiben*
4. **Prozessorflags anpassen**
5. **Zyklen zählen**

```
unsigned char regA, regX, regY, regSP, regP;
unsigned short regPC;

unsigned char memread(unsigned short addr) {
    return 0;
}

void exec() {
  unsigned short cycleCount = 51150;
  unsigned char tmp0;

  while (cycleCount) {
    // Fetch the next instruction opcode
    tmp0 = memread(regPC++);

// Switch to the right implementation
switch (tmp0) {

  // ...
  case 0x18: // CLC - CLear Carry
    regP &= 0xfe; // Clear carry flag
    cycleCount -= 2;
    break;
  // ...

  default:
    // PANIC
    return;
}
  }
}
```

6502 Emulation - Einschränkungen

- **Harte Echtzeitanforderung:**
  - 6502 Instruktionen benötigen 2 - 7 Zyklen bei ~1 MHz
  - AVR bei 20 MHz (max.)
  - → 40 - 140 AVR Zyklen für die Emulation einer 6502 Instruktion
- **Probleme:**
  - Globale / lokale Variablen (SRAM Zugriff teuer)
  - Switch-Statement nicht effizient genug
  - Überhang: Flags anpassen, Zyklen zählen
  - **Apple][Speicheremulation**
  - Darüber hinaus: Display, Tastatur, Sound, ...

Apple][Speicheremulation

- Auf `0xc07f` folgen die Speicherplätze für Extension Ports & 2KB Extension RAM
- **Umsetzung aller grün hinterlegten Speicherbereiche**
  - AVR hat 16KB RAM
  - ROMs im PROGMEM (Harvard Architektur)

Apple][Speicheremulation

Seiten	Funktion	
0x00	Zero Page	
0x01	6502 Stack	
0x02	GETLN Puffer	
0x03	Monitor Vektoren	
0x04...0x07	TXT / LoRes (1)	
0x08...0x0b	TXT / LoRes (2)	
0x0c...0x1f		
0x20...0x3f	HiRes (1)	Frei
0x40...0x5f	HiRes (2)	
0x60...0xbf		
0xc0...0xcf	Memory mapped I/O	
0xd0...0xf7	BASIC ROM	
0xf8...0xff	Monitor ROM	

- Auf `0xc07f` folgen die Speicherplätze für Extension Ports & 2KB Extension RAM
- **Umsetzung aller grün hinterlegten Speicherbereiche**
  - AVR hat 16KB RAM
  - ROMs im PROGMEM (Harvard Architektur)

Apple][Speicheremulation

Seiten	Funktion
0x00	Zero Page
0x01	6502 Stack
0x02	GETLN Puffer
0x03	Monitor Vektoren
0x04...0x07	TXT / LoRes (1)
0x08...0x0b	TXT / LoRes (2)
0x0c...0x1f	Frei
0x20...0x3f	HiRes (1)
0x40...0x5f	HiRes (2)
0x60...0xbf	Frei
0xc0...0xcf	Memory mapped I/O
0xd0...0xf7	BASIC ROM
0xf8...0xff	Monitor ROM

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0xc00	Letzte gedrückte Taste															
0xc01	0xc00* „leeren“															
0xc02	Kassettenausgabe															
0xc03	Lautsprecherausgabe															
0xc04	Utility Strobe															
0xc05	gr	tx	no	mix	1	2	loRe	hiRe	an0		an1		an2		an3	
0xc06	Game Controller								0xc060 - 0xc067							
0xc07	Game Controller Strobe															

- Auf `0xc07f` folgen die Speicherplätze für Extension Ports & 2KB Extension RAM
- **Umsetzung aller grün hinterlegten Speicherbereiche**
  - AVR hat 16KB RAM
  - ROMs im PROGMEM (Harvard Architektur)

Das Problem: Speicheremulation

- Die Memread Funktion ist durch die vielen Fallunterscheidungen sehr umfangreich
- Bedingungen müssen berechnet werden, z.B. „`(addr >> 4) & 0xff`“
- Für jede Instruktion müssen viele Speicherzugriffe erfolgen, z.B. für LDA 5 + 1 Stück:
  - 100 AVR Zyklen Zeit
  - **→ Zeit reicht nicht**

```
unsigned char memread(unsigned short addr) {
 // 65K Memory
 switch(addr >> 12) {
  // 12K RAM from the beginning
  case 0x0: case 0x1:  case 0x2:
   return mem[addr];
  // 36K unused RAM
  case 0x3:  /* ... */  case 0xb:
   return 0;
  // Special I/O location
  case 0xc:
   switch ((addr >> 4) & 0xff) {
    case 0x00:
     // Keyboard read
     return keyLatch;
    case 0x01:
     // Keyboard reset
     return (keyLatch = keyLatch - 128);
    case 0x05:
     // Soft-Switches
     switch (addr & 0xf) {
      case 0x0: // Graphics
       return (grTxMode = 0);
      case 0x1: // Text
       return (grTxMode = 1);
      case 0x2: // No mixed mode
       return (noMixMode = 0);
      case 0x3: // Mixed mode
       return (noMixMode = 1);
    /* ... */
}
```

Zwischenfazit

- Der MOS 6502 Emulator benötigt für 1.023.000 Takte (= 1s) circa **765 ms**
- Der Wert ist schlecht:
  - „Demo-Programm“ nur eine Schleife → keine echte Last
  - Andere Operationen fehlen (noch): Display, Tastatur, ...
- Emulation muss über eine Sekunde „gleichmäßig“ verteilt werden; Lücken mit anderen Aufgaben füllen
  - Restliche 235 ms zu kurz für alle Aufgaben

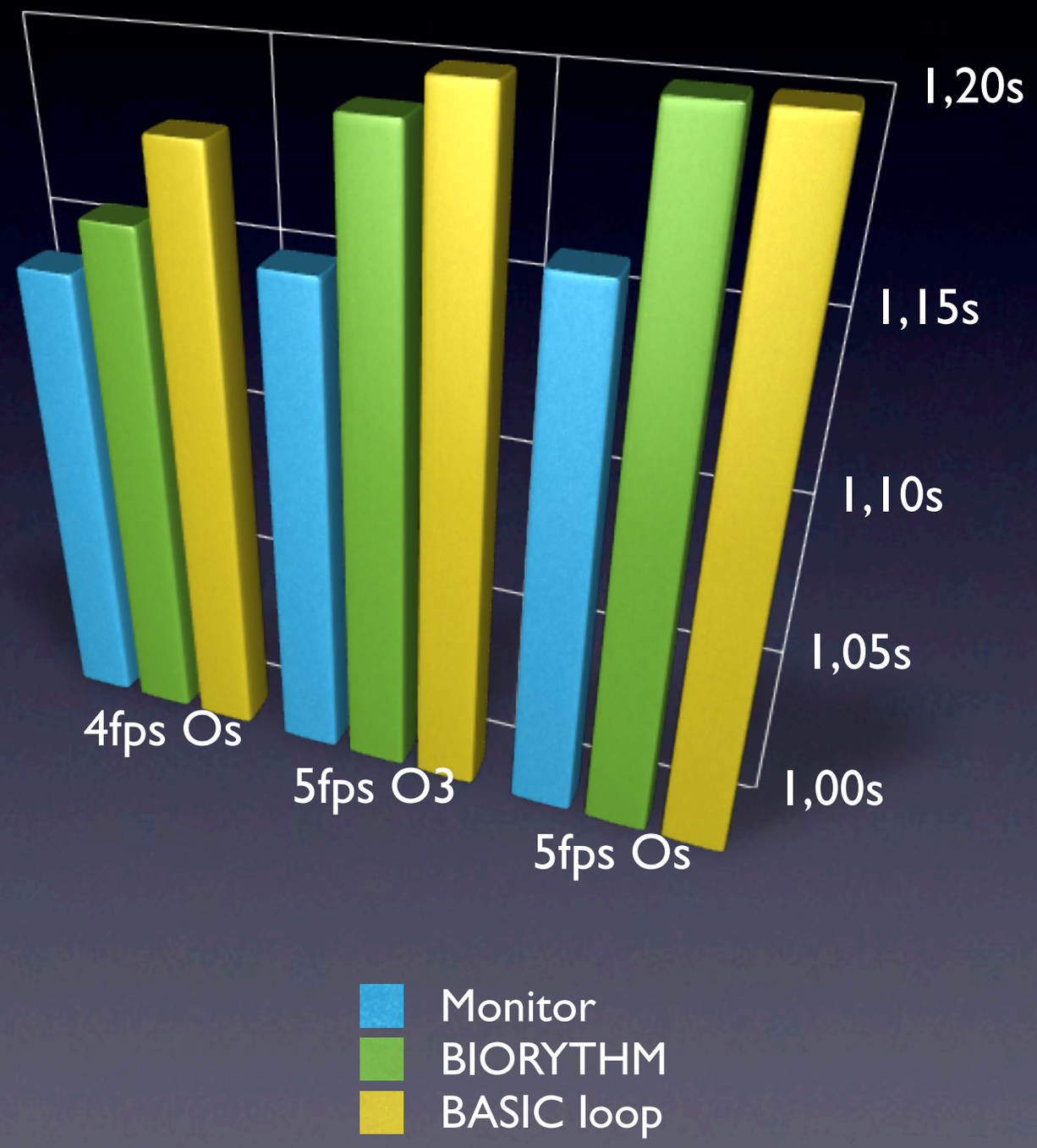
Fortschritt durch Rückschritt

- *Letzte Möglichkeit:* Implementierung der 6502 CPU Emulation komplett in AVR Assembler
  - Volle Kontrolle über den Code
- ~2.567 Zeilen Assembler Quelltext (inkl. Speicher-emulation)
- Kompiliert wesentlich größer (Memread Makro wird kopiert)

```
__memread_pc__              ; Get next 6502 opcode

ldi ZL, pm_lo8(jmptable)    ; 1 Load memory offset
ldi ZH, pm_hi8(jmptable)    ; 1
add ZL, RES                 ; 1 Add argument twice
adc ZH, ZR                  ; 1
add ZL, RES                 ; 1
adc ZH, ZR                  ; 1
ijmp                        ; 2 Indirect jump

jmptable:                   ; The jump table
jmp brk_0x00                ; 3
jmp ora_0x01
                            ; = 11 Zyklen
... ... ...

; ROtate Left (absX, 7)
rol_0x3e:
      __memread_absX__
      bst RES, 7            ; Save shifted out carry bit
      sec                   ; Trans. the 6502 C bit into
      sbrs regP, 0          ;  the AVR status register
      clc
      bld regP, 0           ; Set the new carry bit
      rol RES               ; Rotate left with AVR carry
      __memwrite__          ; Writeback result
      __nz_flags__ RES
      __cyc__ 7
      jmp loop
```

Fazit

- Dauer der C Variante unter gleichen Bedingungen 765 ms; jetzt reduziert auf **600 ms**
- 5fps mit Os ist die beste Wahl und funktioniert am stabilsten
- ~17,5% langsamer als das Original
  - *reine CPU Emulation schneller*
- Läuft trotzdem normal
- Handoptimierter Assemblercode



Nicht dargestellte Themengebiete

- Tasturenwicklung und -eingabeerkennung
- Displayausgabe
- Software-Lademechanismus von der SD-Karte
- Ruhestandsmodus mit Zustandssicherung auf ein EEPROM
- Laufzeitumgebung mit Menüführung
- → jedes genannte Themengebiet ist sehr umfangreich und wird nicht weiter behandelt

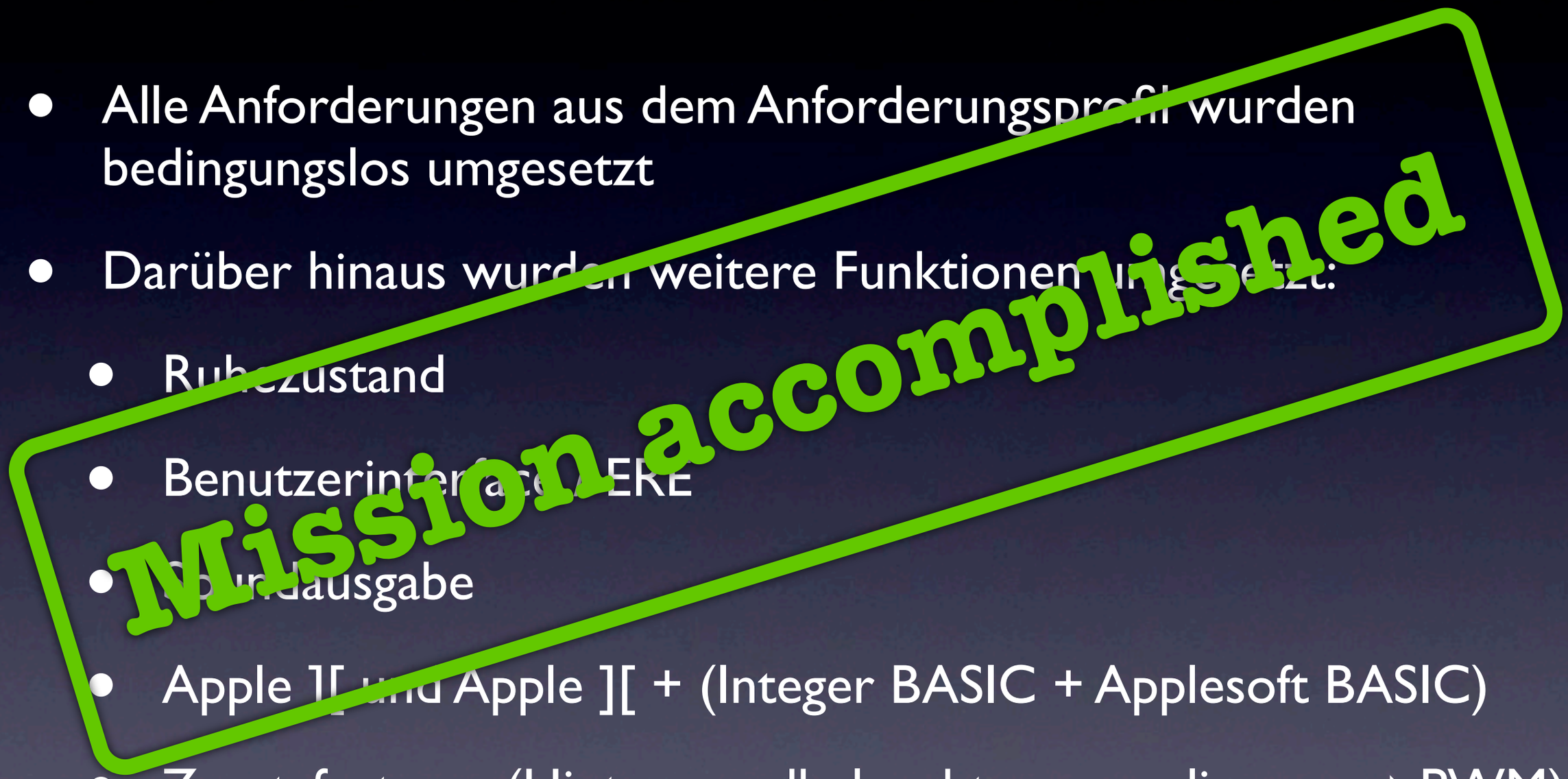
Fazit & Ausblick

Fazit

- Alle Anforderungen aus dem Anforderungsprofil wurden bedingungslos umgesetzt
- Darüber hinaus wurden weitere Funktionen umgesetzt:
  - Ruhezustand
  - Benutzerinterface / ERE
  - Soundausgabe
  - Apple][und Apple][+ (Integer BASIC + Applesoft BASIC)
  - Zusatzfeatures (Hintergrundbeleuchtungsregulierung → PWM)

Fazit

- Alle Anforderungen aus dem Anforderungsprofil wurden bedingungslos umgesetzt
- Darüber hinaus wurden weitere Funktionen umgesetzt:
  - Ruhezustand
  - Benutzerinterface ERE
  - Grundausgabe
- Apple][und Apple][+ (Integer BASIC + Applesoft BASIC)
- Zusatzfeatures (Hintergrundbeleuchtungsregulierung → PWM)



Ausblick

- Nächster Schritt: Speicheremulation entzerren, durch z.B. ein externes SRAM. Probleme:
  - Nicht genügend I/O Pins am ATMega 1284p
  - Zugriffe auf Memory Mapped I/O - z.B. Speaker - erkennen und durchführen
- Wenn diese Hürde genommen ist, können die fehlenden Features folgen (einfach):
  - HiRes Grafikmodus
  - Game-Paddle-Anschluss
  - Disk][Emulation
  - Volle 48KB Benutzerspeicher

Demonstration

Vielen Dank für die Aufmerksamkeit!

Dank an

Prof. Dr. Hannes Frey und Dr. Merten Joost
für die intensive Unterstützung und Betreuung.

Weitere Bildquellen

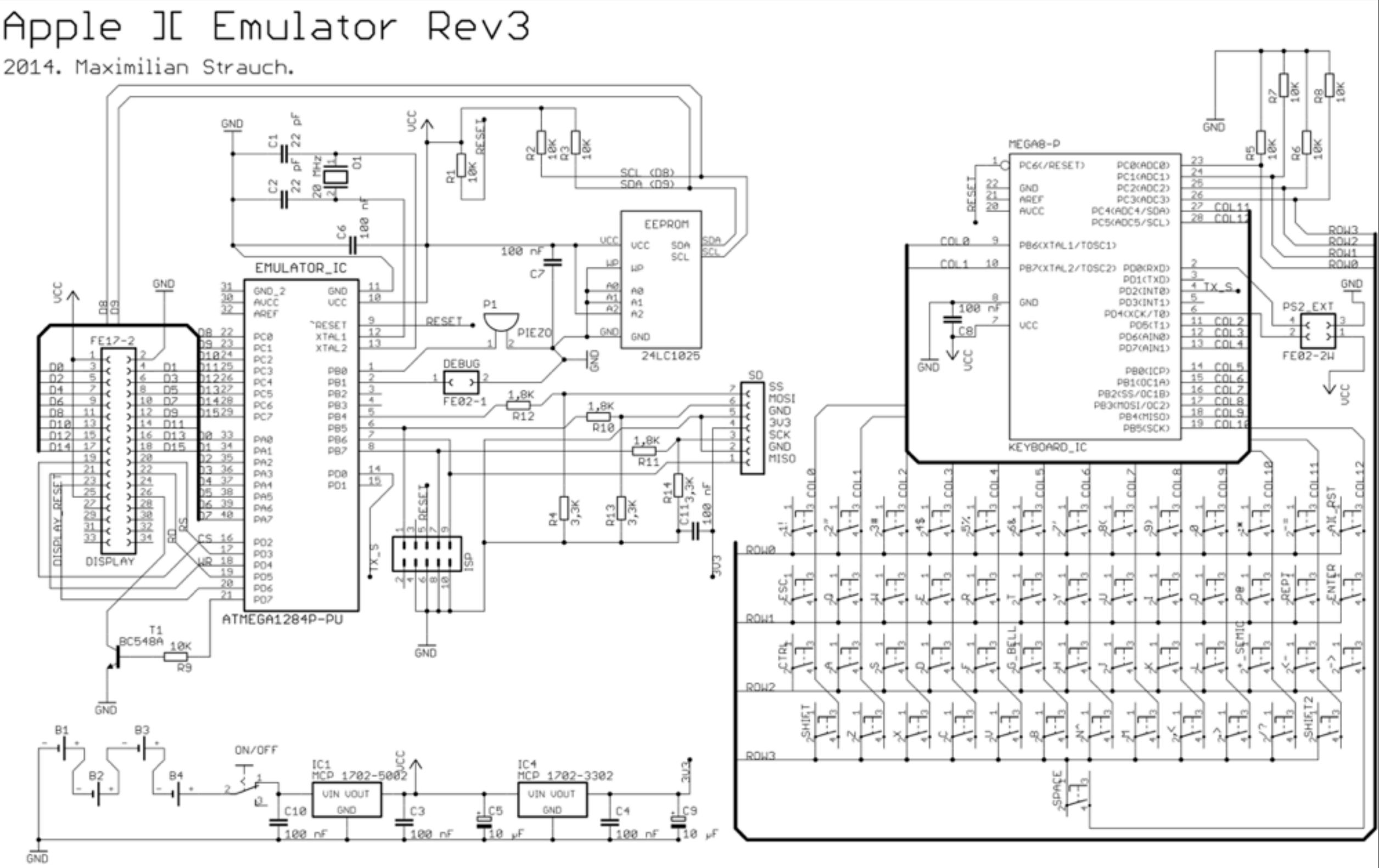
Folie „Einsatzgebiete des 6502“:

http://www.allaboutapple.com/
http://en.wikipedia.org/wiki/File:BBC_Micro_Front_Restored.jpg
http://en.wikipedia.org/wiki/File:Atari-2600-Console.jpg
http://en.wikipedia.org/wiki/File:Atari_800.jpg
"CBMVIC20P8" by Cbmeeks / processed by Pixel8 - Original uploader was Cbmeeks at en.wikipedia. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:CBMVIC20P8.jpg#mediaviewer/File:CBMVIC20P8.jpg
"C64c system" by Bill Bertram - Own work. Licensed under Creative Commons Attribution-Share Alike 2.5 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:C64c_system.jpg#mediaviewer/File:C64c_system.jpg
"NES-Console-Set" by Evan-Amos - Own work. Licensed under Public domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:NES-Console-Set.jpg#mediaviewer/File:NES-Console-Set.jpg
"Famicom-Console-Set" by Evan-Amos - Own work. Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Famicom-Console-Set.jpg#mediaviewer/File:Famicom-Console-Set.jpg
"OSI Challenger 4P" by Bilby - Own work. Licensed under Creative Commons Attribution 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:OSI_Challenger_4P.jpg#mediaviewer/File:OSI_Challenger_4P.jpg
"Commodore-64-Computer" by Evan-Amos - Own work. Licensed under Public domain via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:Commodore-64-Computer.png#mediaviewer/File:Commodore-64-Computer.png
"MOS 6502AD 4585 top". Licensed under Creative Commons Attribution-Share Alike 3.0 via Wikimedia Commons - http://commons.wikimedia.org/wiki/File:MOS_6502AD_4585_top.jpg#mediaviewer/File:MOS_6502AD_4585_top.jpg
http://apple2history.org/history/ah02/

Verwandte Arbeiten

- Verwandte Arbeiten sind rar:
  - 2007 hat sich ein Team in einem Elektronikkurs an der Cornell University (NY, USA) an einem „stationären“ Emulator versucht, ist aber gescheitert
- Einige Projekte sind im Web zu finden, die sich mit der Emulation des 6502 auf dem AVR beschäftigen
- Umsetzung über einen FPGA → nicht vergleichbar, da FPGAs wesentlich performanter und fortschrittlicher sind
- Umsetzung über Raspberry PI und andere ARM-Embedded-Systeme → nicht vergleichbar, da überall hierauf Linux läuft und nur „`apt-get install mess`“ zur Fertigstellung erforderlich ist
- → **Fazit:** keine weiteren verwandten Arbeiten oder Hinweise, dass dieses Projekt jemals umgesetzt / versucht wurde, vorhanden

Schaltplan


Apple][Emulator Rev3
2014. Maximilian Strauch.
EMULATOR_IC
ATMEGA1284P-PU
EEPROM
24LC1025
FE17-2
DISPLAY
FE02-1
PIEZO
DEBUG
RESET
ISP
S_RESET
SD
KEYBOARD_IC
MEGA8-P
PS2_EXT
FE02-2W
IC1
MCP 1702-5002
IC4
MCP 1702-3302
ON/OFF
T1
BC548A
SCL (D8)
SDA (D9)
TX_S
ROW0
ROW1
ROW2
ROW3