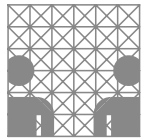


# 64-210 Eingebettete Systeme

https://lernen.min.uni-hamburg.de

https://tams.informatik.uni-hamburg.de/lectures/2020ss/vorlesung/es

Andreas Mäder

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik

**Technische Aspekte Multimodaler Systeme**

Sommersemester 2020

# Info zur Organisation



1. Organisatorisches
   - Vorlesung
   - Übungen
   - Allgemein
2. Literatur

. . . dieses Semester ist wegen *Corona* alles anders

- Videomaterial, Foren und die gesamte Organisation sind im MIN-Moodle hinterlegt:
  `https://lernen.min.uni-hamburg.de`

- eigenständiges Lernen zu Hause
  ⇒ Screencast Videos im Moodle / auf Lecture2Go
- regelmäßige Online-Meetings
  - Vorlesung: Fr., ab 12:30
  - Übung: Di., ab 14:15 und Fr., ab 10:15
- bei Unklarheiten: Fragen in entsprechenden Q&A Foren stellen
  ⇒ die Antworten bilden dort eine Sammlung „zum Nachlesen“

# Vorlesung (cont.)

tams.informatik.uni-hamburg.de/lectures/2020ss/vorlesung/es



- Vorlesung folgt: „*Embedded System Design*“ von P. Marwedel
- eBook: dx.doi.org/10.1007/978-3-319-56045-8 UHH Campus
- Originalmaterial: ls12-www.cs.tu-dortmund.de/daes/de/daes/mitarbeiter/prof-dr-peter-marwedel/embedded-system-text-book/slides/slides-2013.html
- diverse gute Lehrbücher — Empfehlungen s.u.

# Übungen

tams.informatik.uni-hamburg.de/lectures/2020ss/vorlesung/es/uebungen



- Einzelarbeit zu Hause, simulativ
- Betreuung in Online-Meetings: Di., ab 14:15 und Fr., ab 10:15
  Beginn: ab dem 28.04.
- ▷ spätere Aufgaben: „vor Ort“, an der Hardware

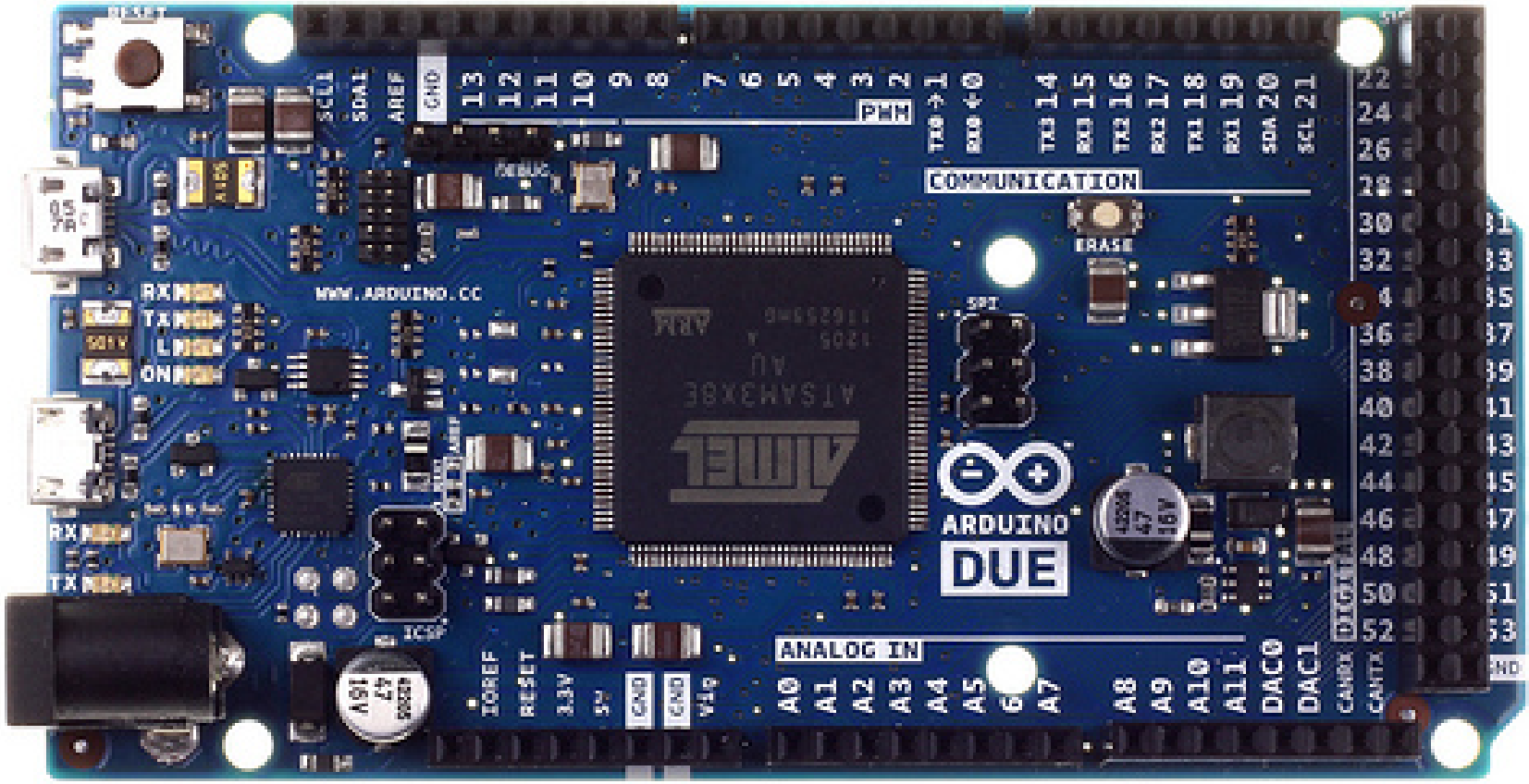
Grundidee

- praktische Ergänzung zur Vorlesung
- Programmieraufgaben mit Mikrocontrollern (und FPGAs)
  - Anschluss von Sensoren und Aktoren
  - Kontrollaufgaben (Steuerung durch endlichen Automaten)
  - Interruptbehandlung
  - einfache Schnittstellen (seriell, $I^2C$...)
  - Kommunikation
  - ...
- spätere Aufgaben bauen auf Vorherigen auf
  $\Rightarrow$ fertige Lösungen abspeichern und gut dokumentieren
  $\Rightarrow$ automatisierte Tests erstellen

# Übungen: Hardwareplattformen

## $\mu$Controller System

- Arduino Due `www.arduino.cc`



# Übungen: Hardwareplattformen (cont.)

## $\mu$Controller System

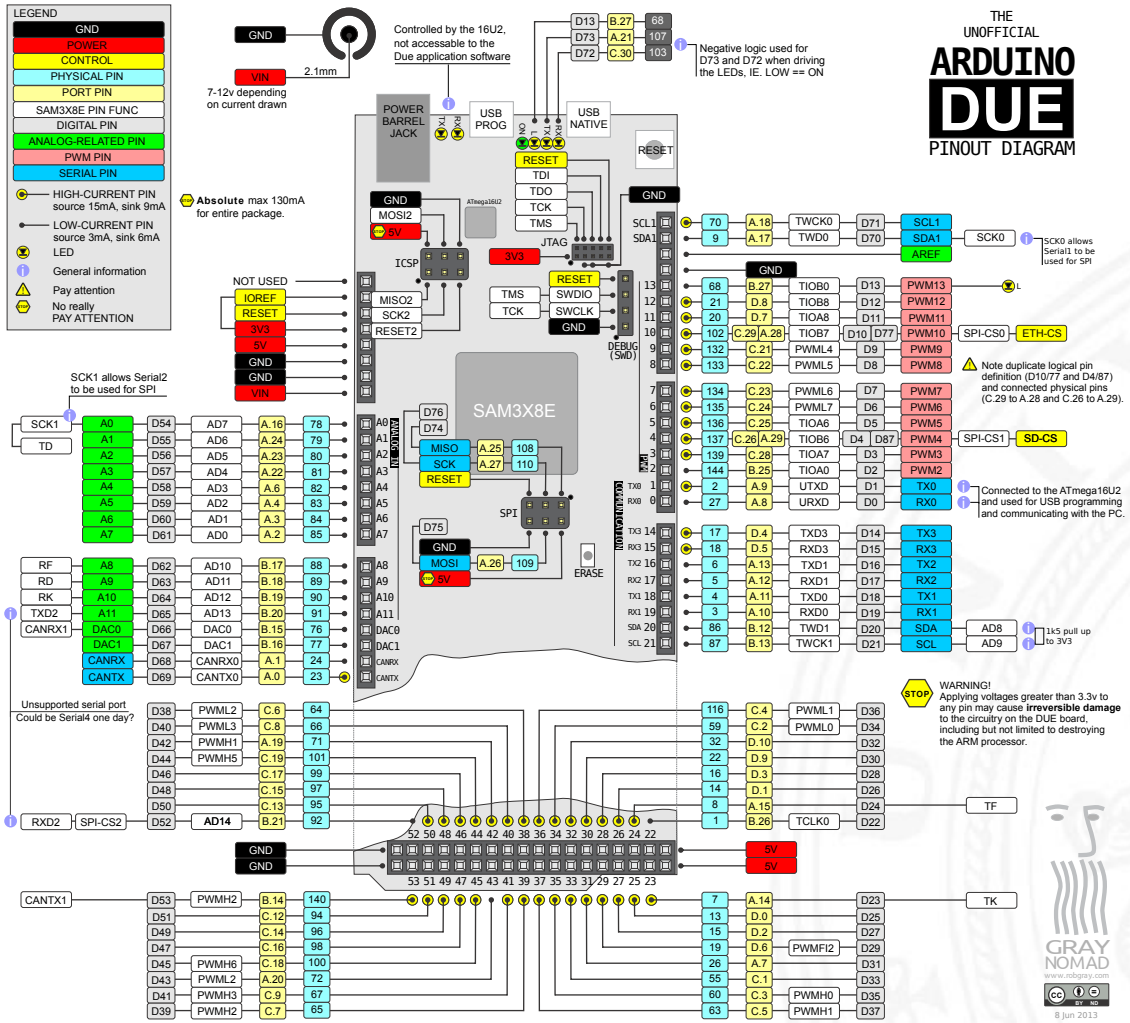


- Arduino Due `www.arduino.cc`
  - Microcontroller AT91SAM3X8E ARM Cortex-M3, 32bit
  - Clock Speed 84 MHz
  - Operating Voltage 3,3V
  - Digital I/O Pin 54 (12 $\times$ PWM output)
  - Analog Input Pins 12
  - Analog Outputs Pins 2 (DAC)
  - SRAM 96 KB (two banks: 64KB and 32KB)
  - Flash Memory 512 KB
- typische Prototypenplatine mit frei belegbaren Ein-/Ausgängen
- Anschluss externer Komponenten über Pfostenleisten
- sehr große Anzahl fertiger „*Shields*“
  - Sensoren
  - Aktoren, Motortreiber
  - Displays
  - Schnittstellen

# Übungen: Hardwareplattformen (cont.)

## $\mu$Controller System

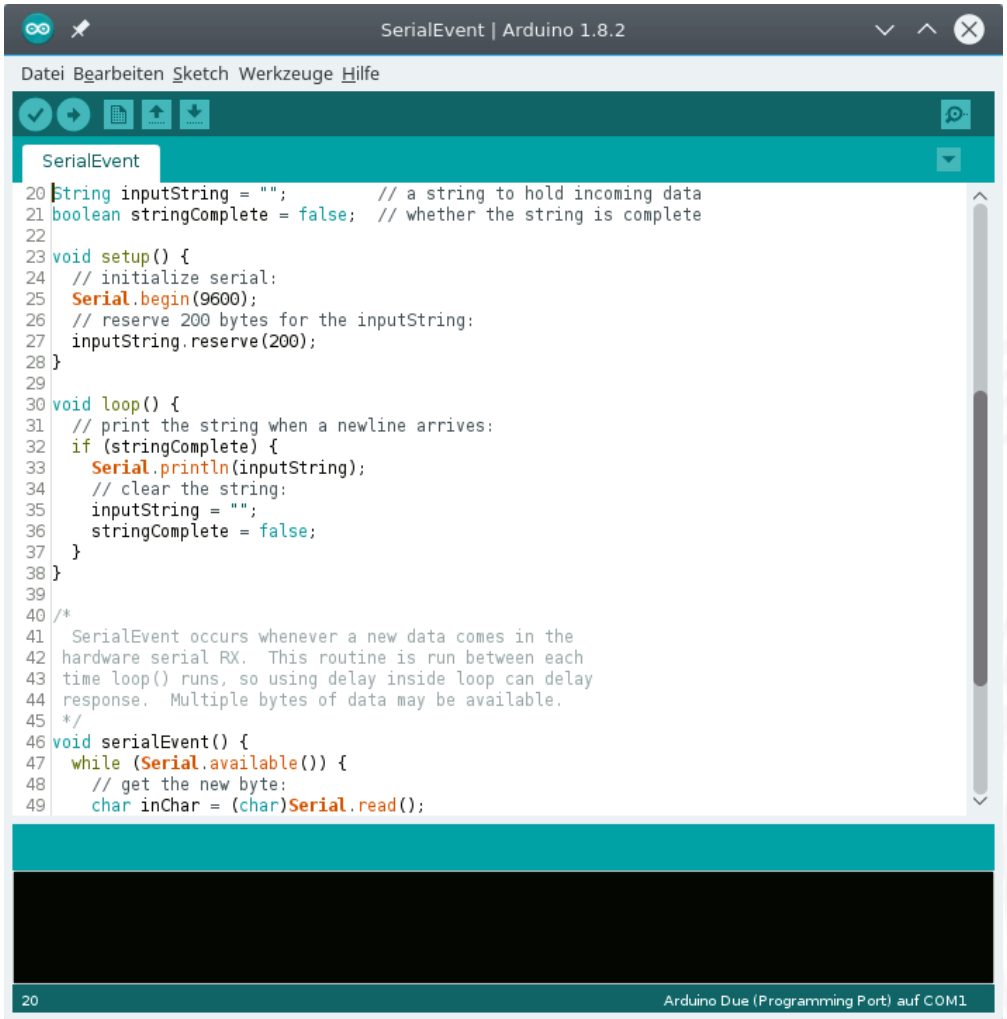
- I/O-Schema



# Übungen: Hardwareplattformen (cont.)

## $\mu$Controller System

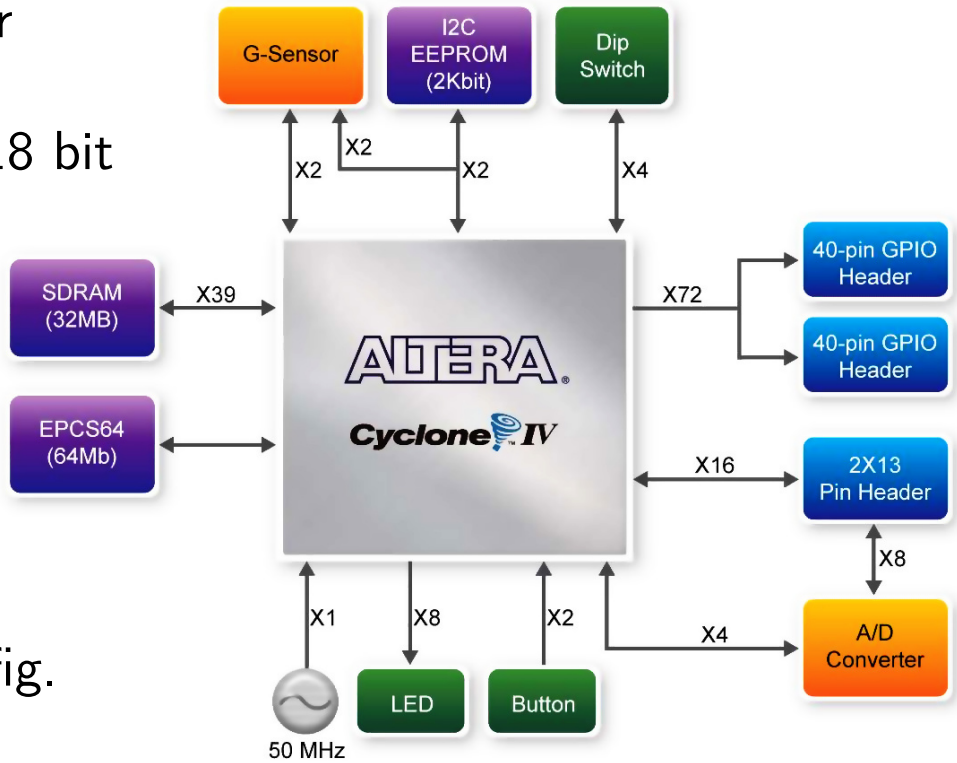
- Arduino-IDE / C-Code



# Übungen: Hardwareplattformen
## FPGA-Prototypensystem

- Altera DE0-Nano `www.terasic.com.tw`

  `www.intel.de/content/www/de/de/products/programmable.html`
- programmierbare Hardware: FPGA
  - Cyclone IV EP4CE22F17C6N [60 nm Prozess, 2009]
  - 153 I/O Pins, gesamt 256 Pins
  - 22 320 LEs $\approx$ 270 000 Gatter
  - 594 Kbit (interner) Speicher
  - 66 HW-Multiplizierer: $18 \times 18$ bit
  - 4 PLLs
- On-Board Speicher
  - 32MB SDRAM
  - 2Kbit $I^2C$ EEPROM
- Konfiguration
  - über USB Schnittstelle
  - EPCS64: Flash, serielle Konfig.



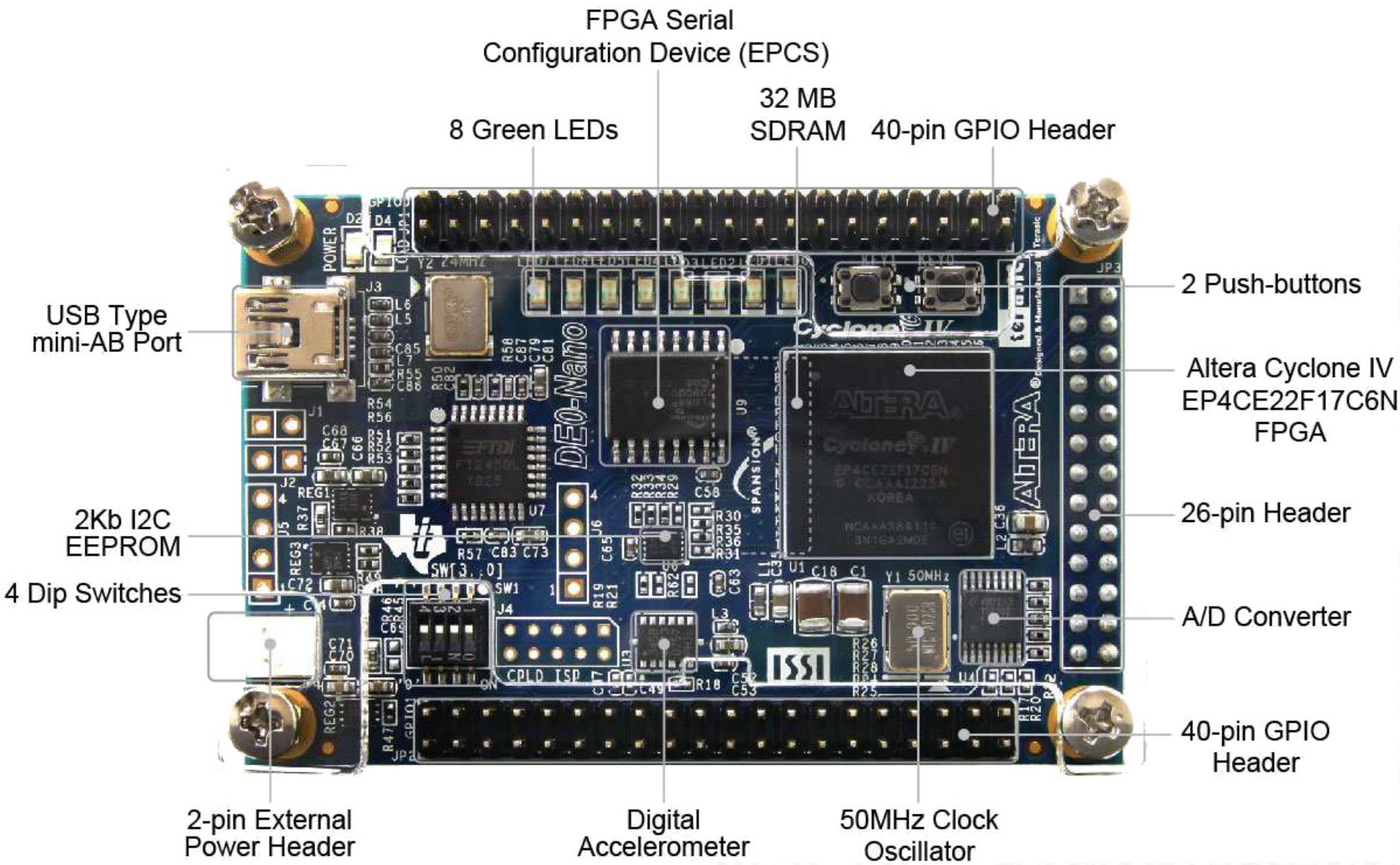
# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem

- Ein-/Ausgabe
  - 8 LEDs
  - 2 Taster
  - 4 DIP Schalter
- Beschleunigungssensor: ADXL 345, 3-Achsen, 13-bit Auflösung
- A/D Wandler: ADC128S022, 8-Kanal, 12-bit Auflösung
- Erweiterungsstecker
  - $2 \times 40$-Pin: 72 I/O Pins + Spannungsversorgung; 5V, 3,3V, Gnd
  - 26-Pin: 16 I/O Pins + 8 analoge Eingänge

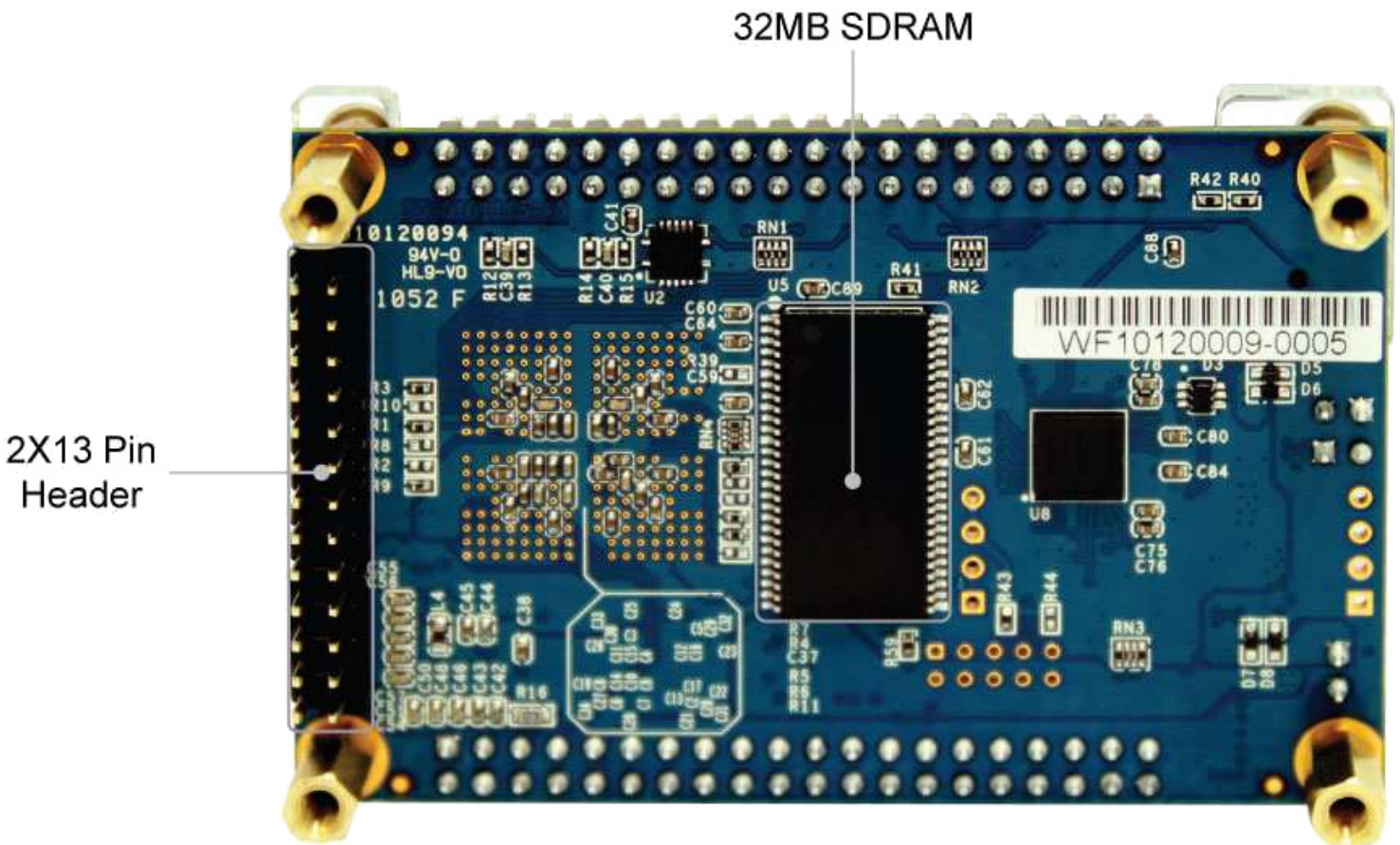
# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem



# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem



# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem

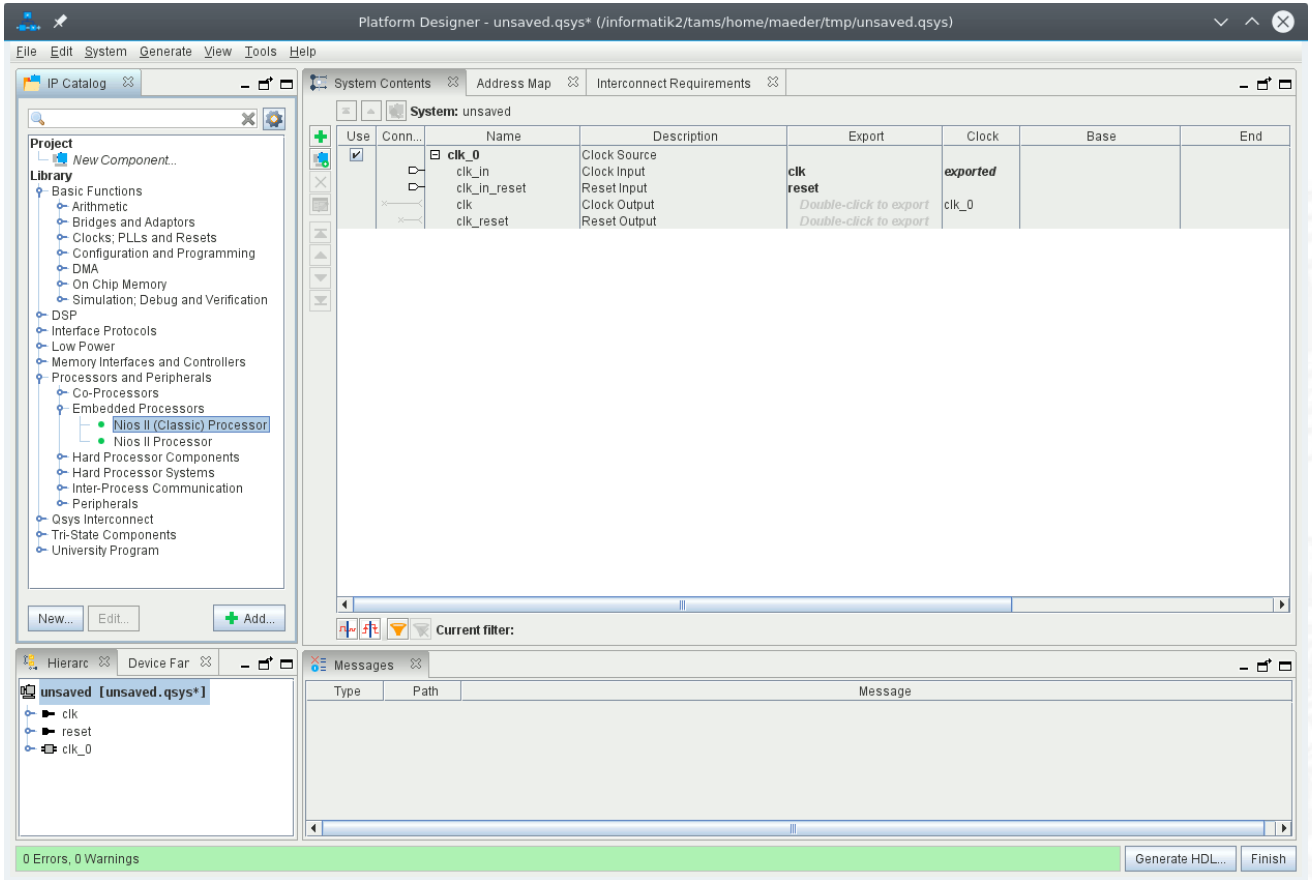
Implementationsmöglichkeiten

1. Softwareentwurf
   - IP-Komponenten (*Intellectual Property*)
     Speicher, Busse, I/O-Schnittstellen...
   - 32-bit Prozessor (NIOS II)
   - Softwareentwicklung: C Programmierung
     Eclipse + Cross-Compiler
2. Hardwareentwurf: Hardwarebeschreibungssprachen hier: VHDL
3. gemischt: Hardware + Software + IP-Komponenten

# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem

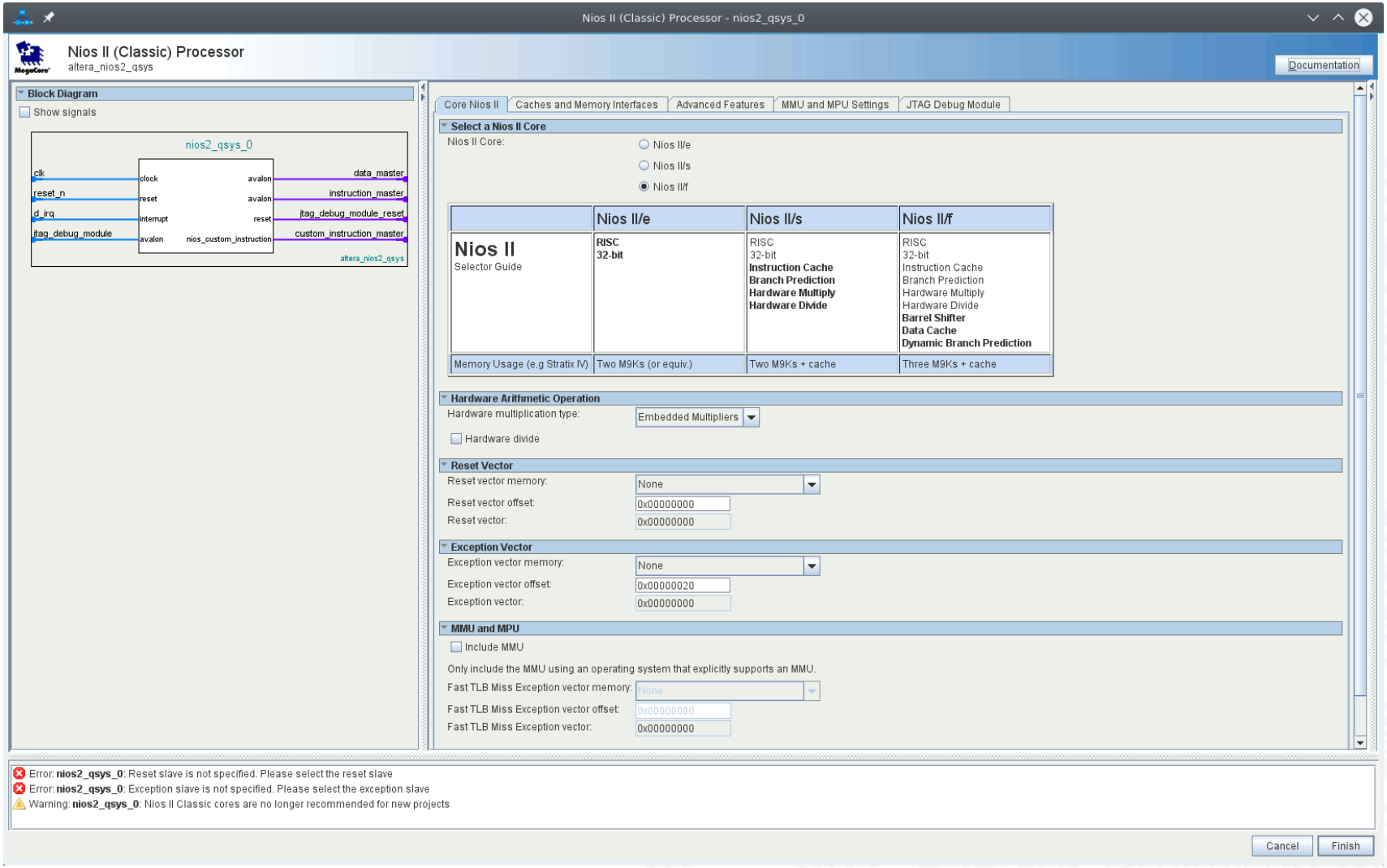
- IP-Komponenten



# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem

- NIOS Prozessor



# Übungen: Hardwareplattformen (cont.)

## FPGA-Prototypensystem

- VHDL-Code

de0Board.vhd — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

de0Board.vhd | cDisp14x6.vhd

```
-- normal display, no timeout
when dNorm1 =>
  if ack = req then sTransition(dClear1, dispClear);
  end if;

-- clear display
when dClear1 =>
  if ack = req then sTransition(dAllChars, dispChar, timeC/4);
                    char <= character'left;
  end if;

-- display char-set: 0..255, timeC/4 between chars
when dAllChars =>
  if timer = 0 then
    if ack = req then
      if char = character'right
                 then sTransition(dInv, dispInverse, timeC);
                 else sTransition(dAllChars, dispChar, timeC/4);
                    char <= character'succ(char);
      end if;
    end if;
  else              timer := timer-1;
  end if;

-- invert display, for timeC
when dInv =>      tTransition(dNorm2, dispNormal, timeC);

-- normal display, no timeout
when dNorm2 =>
  if ack = req then sTransition(dClear2, dispClear);
  end if;

-- clear display
when dClear2 =>
  if ack = req then sTransition(dXY, dispPosXY);
                    xPos <= 4;
                    yPos <= 2;
  end if;

-- set xPos=4, yPos=2
when dXY =>
  if ack = req then sTransition(dString, dispChar);
--                  invC <= '1';
                    char <= strReg(1);
                    strReg := strReg(2 to strReg'right)&' ';
                    strCnt := strCnt-1;
  end if;

-- string output, no timeout
when dString =>
  if ack = req then
    if strCnt = 0 then state <= halt;
                       invC <= '0';
                  else sTransition(dString, dispChar);
```

Line 1, Column 1 INSERT Soft Tabs: 4 (8) UTF-8 VHDL

Search and Replace

cDisp14x6.vhd — Kate

File Edit View Projects Bookmarks Sessions Tools Settings Help

de0Board.vhd | cDisp14x6.vhd

```
-- architecture
architecture pcd8544 of cDisp14x6 is
  type   stateTy      is (power, reset, init, active);
  signal state        : stateTy;
  signal dispEna      : std_logic;
  signal ackL         : std_logic;
  signal charBit      : natural range 0 to 4031;
  signal charReg      : std_logic_vector (47 downto 0);
  signal romA         : std_logic_vector ( 6 downto 0);
  signal romD         : std_logic_vector (39 downto 0);
begin
  -- set bg LED
  bgLed       <= '1' when bgLight else '0';

  -- component instantitions
  romI: charROM port map (romA,  clk, romD);


  ctrlP: process (clkN, rstN) is
    procedure stInit (  nxtState      : stateTy;
                        isData        : std_logic;
                        regBit        : natural range 0 to 4031;
                        regData       : std_logic_vector (47 downto 0)) is
    begin
      state   <= nxtState;
      s_dNc   <= isData;
      charBit <= regBit;
      charReg <= regData;
    end procedure stInit;

  begin
    if rstN = '0' then                  -- async. reset
                dispEna<= '0';
                state  <= power;
                s_rstN <= '1';
                s_dNc  <= '0';
                charReg<= (others => '0');
    elsif rising_edge(clkN) then
      case state is
      -- power on / hardware reset
      when power =>
                ackL   <= '1';
                dispEna<= '1';
                state  <= reset;        -- 1. reset pcd8544
                s_rstN <= '0';
      -- reset controller
      when reset =>
                s_rstN <= '1';
                stInit(init, '0', 39,   -- 2. setup pcd8544
                       instExtdC & setBiasC & setVopC & instNormC & dispNormC & x"00");
      -- initialize controller
```

Line 117, Column 1 INSERT Soft Tabs: 4 (8) UTF-8 VHDL

Search and Replace

# Feedback erwünscht



- über die Moodle Foren / E-Mail an mich
- Fehler und Ungenauigkeiten in den Folien und Materialien bitte melden
- Vorschläge und Hinweise auf Tools, schöne Lehrmaterialien etc. sind immer willkommen!

Problem: unterschiedliches Vorwissen!

- Voraussetzungen: insbesondere Rechnerstrukturen
  aber auch: Assemblerprogrammierung, Betriebssysteme, Netzwerke...
- generell: Interesse an Hardware

Dr. Andreas Mäder

E-Mail `maeder@informatik.uni-hamburg.de`

Tel. +49 40 42883 2502

Büro Informatikum, Haus F 317

# Literatur: empfohlene Lehrbücher



[Mar18] P. Marwedel:

*Embedded System Design – Embedded Systems Foundations of Cyber-Physical Systems*.
3rd edition. Springer-Verlag, 2018. ISBN 978–3–319–56043–4

Primärliteratur: Komplette Behandlung des Themas. Buch und Foliensatz sind Vorlage dieser Vorlesung. Die Folien basieren auf der zweiten Auflage des Buchs.

Originalmaterial der Vorlesung von Peter Marwedel: `ls12-www.cs.tu-dortmund.de/daes/daes/mitarbeiter/prof-dr-peter-marwedel/embedded-system-text-book/slides/slides-2013.html`

eBook in der Informatik-Bibliothek: `dx.doi.org/10.1007/978-3-319-56045-8`

[TH07] J. Teich, C. Haubelt:

*Digitale Hardware/Software-Systeme – Synthese und Optimierung*.
2. Auflage. Springer-Verlag, 2007. ISBN 978–3–540–46822–6

Schwerpunkte: Realisierung eingebetteter Systeme, also Synthese, Scheduling, Bindung von Ressourcen etc.

eBook in der Informatik-Bibliothek: `dx.doi.org/10.1007/978-3-540-46824-0`

# Literatur: empfohlene Lehrbücher (cont.)



[HT10] C. Haubelt, J. Teich:

*Digitale Hardware/Software-Systeme – Spezifikation und Verifikation*.

Springer-Verlag, 2010. ISBN 978–3–642–05355–9

Schwerpunkte: Methoden zur Spezifikation eingebetteter Systeme und der (formalen) Verifikation, Simulation etc.

eBook in der Informatik-Bibliothek: `dx.doi.org/10.1007/978-3-642-05356-6`

[VG02] F. Vahid, T. Givargis:

*Embedded System Design – A Unified Hardware/Software Introduction*.

John Wiley & Sons, 2002. ISBN 978–0–471–38678–0

Alle Themen im Überblick, etwas älter.

Mehrere Exemplare in der Informatik-Bibliothek vorhanden.

# Literatur: weitere Lehrbücher



[Brä08] T. Bräunl:

*Embedded Robotics – Mobile Robot Design and Applications with Embedded Systems*.

3rd edition. Springer-Verlag, 2008. ISBN 978–3–540–70534–5

Eigentlich ein „Robotik"-Buch, dient aber als Beispiel, dass die Robotik auf „Eingebetteten Systemen" aufbaut (Teil 1)

eBook in der Informatik-Bibliothek: `dx.doi.org/10.1007/978-3-540-70534-5`

[Lew13] D. Lewis:

*Fundamentals of embedded software – with the ARM Cortex-M3*.

2nd edition. Pearson Education, 2013.

ISBN 978–0–13–291654–7

Schwerpunkt: Software, viel praktische Programmierung in C und Assembler.

# Literatur: weitere Lehrbücher (cont.)



[SC17] J. Sanchez, M.P. Canton:

*Embedded Systems Circuits and Programming*.
CRC Press, 2017. ISBN 978–1–138–07406–4

Schwerpunkt auf PIC $\mu$Controller, enthält außerdem viele (elektro-) technische Grundlagen.

[Wol16] M. Wolf:

*Computers as Components –*
*Principles of Embedded Computing System Design*.
4th edition. Morgan Kaufmann Publishers Inc., 2016.
ISBN 978–0–12–805387–4

[Zöb08] D. Zöbel:

*Echtzeitsysteme – Grundlagen der Planung*.
Springer-Verlag, 2008. ISBN 978–3–540–76395–6

eBook in der Informatik-Bibliothek: dx.doi.org/10.1007/978-3-540-76396-3

# Literatur: Vertiefung



[BO15] R.E. Bryant, D.R. O'Hallaron:

*Computer systems – A programmers perspective*.
3rd global ed., Pearson Education Ltd., 2015. ISBN 978–1–292–10176–7

Rechnerarchitektur mit Schwerpunkt Software und Systeme,
leider nicht ganz billig. Viele C-Programme und Systemprogrammierung. Beispiele anhand Intel x86 Architektur.

[TA14] A.S. Tanenbaum, T. Austin:

*Rechnerarchitektur – Von der digitalen Logik zum Parallelrechner*.
6. Auflage, Pearson Deutschland GmbH, 2014.
ISBN 978–3–8689–4238–5

Guter Überblick zum Thema Rechnerarchitektur, klares didaktisches Konzept. Java VM, Intel x86, SPARC.
Mit jeder Auflage komplett überarbeitet und aktualisiert.

# Literatur: Vertiefung (cont.)



[PH16a] D.A. Patterson, J.L. Hennessy:

*Computer Organization and Design – The Hardware Software Interface: ARM Edition*.

Morgan Kaufmann Publishers Inc., 2016.

ISBN 978–0–12–801733–3

Schönes Lehrbuch von den Entwicklern der RISC/MIPS Prozessoren.

[HP17] J.L. Hennessy, D.A. Patterson:

*Computer Architecture – A Quantitative Approach*.

6th edition, Morgan Kaufmann Publishers Inc., 2017.

ISBN 978–0–12–811905–1

Die Bibel zum Thema Rechnerarchitektur

# Literatur: Arduino Programmierung



[Mar11] M. Margolis:

*Arduino Cookbook – Recipes to Begin, Expand, and Enhance Your Projects*.
2nd edition. O´Reilly, 2011. ISBN 978–1–4493–1387–6

Die Arduino Referenz

[McR13] M. McRoberts:

*Beginning Arduino*.
2nd edition. Apress, 2013. ISBN 978–1–4302–5016–6

Praxisnahe Beispiele und Projekte

eBook in der Informatik-Bibliothek: dx.doi.org/10.1007/978-1-4302-5017-3