

WS12

21.09.2012



2. Entwurf und Verhalten synchroner Automaten

Automatenstrukturen

Entwurfsmethodik; Beispiel Sequenzerkennung

VHDL-Modell, Timing, Synthese

Zwei-Prozess VHDL-Automatenbeschreibungen

Entkopplung von Zustandsautomaten

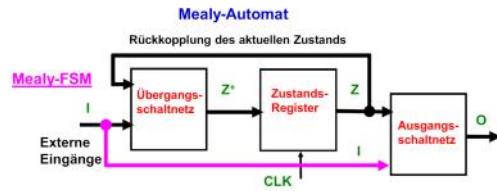
Entwurf eines sequentiellen Addierers

(Mealy, Moore)



2. Entwurf und Verhalten synchroner Automaten

2.1 Automatenstrukturen



Beim Mealy-Automaten gilt:
 $Z^* = f(E, Z)$ und $O = f_{\text{Mealy}}(I, Z)$

3

CE - DS 2 FSM

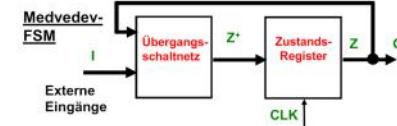


Moore- und Medvedev-Automaten

Beim Moore-Automaten gilt:
 $Z^* = f(E, Z)$ und
 $O = f_{\text{Moore}}(Z)$



Der Medvedev-Automat ist ein Moore-Automat, bei dem die Ausgänge den Zustands-Flipflops entsprechen:
 $Z^* = f(E, Z)$ und
 $O = Z$



4

CE - DS 2 FSM



2.2 Entwurfsmethodik ohne CAE

- 1) Erstelle ein Zustandsdiagramm. Ggf. kann zuerst auch eine mnemonische Folgezustands- und Ausgangstabelle erstellt werden.
- 2) Minimiere ggf. die Anzahl der Zustände
- 3) Wähle eine Menge von Zustandssignalen und ordne diesen die mnemonischen Zustände zu.
- 4) Wähle einen Flipflop-Typ für die Hardware-Realisierung (meistens D-FF).
- 5) Stelle Folgezustands- und Ausgangstabellen für die Zustands- bzw. Ausgangssignale auf.
- 6) Minimiere die Folgezustands- und Ausgangsfunktionen
- 7) Analysiere möglicherweise vorhandene Pseudozustände
- 8) Zeichne einen Schaltplan

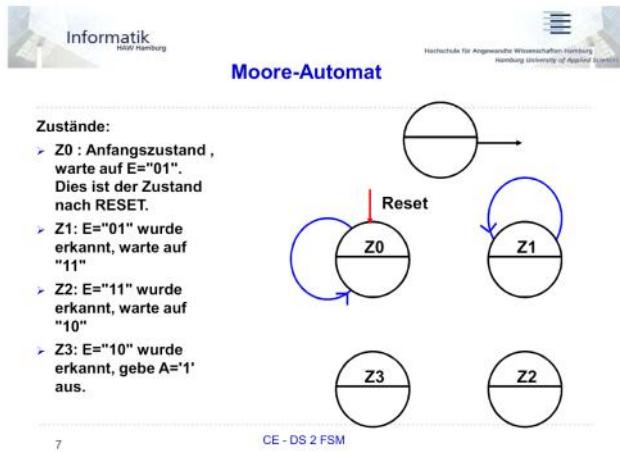
5 CE - DS 2 FSM



Beispiel: Impulsfolgeerkennung

- In einem seriellen, 2 Bit breiten Datenstrom soll die Impulsfolge am Eingang $E = \dots, 01, 11, 10, \dots$ erkannt werden und am Ausgang A des takt synchronen Automaten mit einer '1' für die Dauer einer Taktperiode quittiert werden. Andernfalls soll der Ausgang '0' sein. Die Starteingangsimpulse können länger, als einen Takt anliegen.
- Nachfolgend werden die einzelnen Entwurfsschritte für eine Realisierung als Moore- und als Mealy-Automat erläutert.
- Für beide Varianten wird ein synthesefähiges VHDL-Modell erstellt.
- Das Zeitverhalten bei der Bildung des Folgezustands sowie des Ausgangssignals wird für beide Varianten analysiert.

6 CE - DS 2 FSM



7

Informatik HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Zustandscodierung, Folgezustands- u. Ausgangstabellen zur Sequenzerkennung

Heuristischer Ansatz für die Zustandscodierung:

Zustand	Z ₁ , Z ₀	Zustandssignale
Z0	0 0	
Z1	0 1	
Z2	1 0	
Z3	1 1	

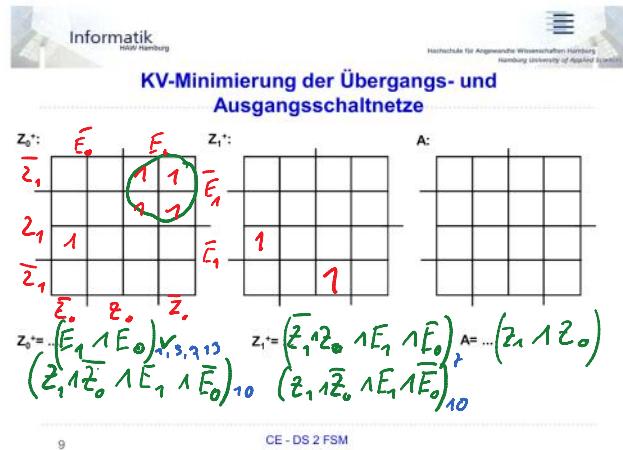
Mit 2 DFFs existieren insgesamt $4! = 4 \cdot 3 \cdot 2 \cdot 1 = 24$ Permutationen

No.	Z3	Z2	Z1	Z0
1	00	01	10	11
2	00	01	11	10
3	00	10	01	11
4	00	10	11	01
...

CE - DS 2 FSM

ÜSN

8



9

CE - DS 2 FSM

2.3 VHDL-Modell des Moore-Automaten

- ✓ Zusätzliche Anforderung: Das Fortschreiten des Automaten soll dann erfolgen, wenn das zusätzliche Freigabesignal ENABLE = '1' ist.
- ✓ Einfachster Ansatz: **Drei Funktionsblöcke** des Moore-Modells mit je einem Prozess beschrieben.

```

entity FSM_1_MOORE is
port( CLK, RESET, ENABLE : in bit;      -- sekundäre Eingangssignale
      E : in bit_vector(1 downto 0);    -- Impulsfolge
      A : out bit );                  -- Ausgangssignal
end FSM_1_MOORE;
architecture SEQUENZ of FSM_1_MOORE is
type ZUSTÄNDE is (Z0, Z1, Z2, Z3);      -- Aufzählungstyp
signal ZUSTAND, FOLGE_Z: ZUSTÄNDE;       -- Prozess-Kommunikation
begin
Z_SPEICHER: process(CLK, RESET)          -- Zustandsaktualisierung
begin
if RESET = '1' then ZUSTAND <= Z0 after 5 ns;
elsif CLK = '1' and CLK'event then
  if ENABLE = '1' then ZUSTAND <= FOLGE_Z after 5 ns;
  end if;
end if;
end process Z_SPEICHER;
  
```

10

CE - DS 2 FSM

Informatik
HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

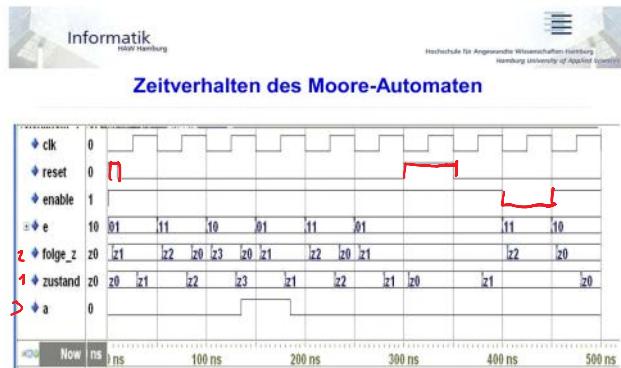
```

UE_SN: process(E, ZUSTAND)      -- Folgezustandsberechnung
begin
  case ZUSTAND is
    when Z0 => if E = "01" then FOLGE_Z <= Z1 after 5 ns;
    else FOLGE_Z <= Z0 after 5 ns;
    end if;
    when Z1 => if E = "10" then FOLGE_Z <= Z2 after 5 ns;
    elsif E = "01" then FOLGE_Z <= Z1 after 5 ns;
    else FOLGE_Z <= Z0 after 5 ns;
    end if;
    when Z2 => if E = "0" then FOLGE_Z <= Z3 after 5 ns;
    elsif E = "01" then FOLGE_Z <= Z1 after 5 ns;
    else FOLGE_Z <= Z0 after 5 ns;
    end if;
    when Z3 => if E = "0" then FOLGE_Z <= Z1 after 5 ns;
    else FOLGE_Z <= Z0 after 5 ns;
    end if;
  end case;
end process UE_SN;
A_SN: process(ZUSTAND)          -- Ausgangssignalberechnung
begin
  case ZUSTAND is
    when Z3 => A <= "1" after 5 ns;
    when others => A <= "0" after 5 ns;
  end case;
end process A_SN;
end SEQUENZ;

```

CE - DS 2 FSM

11





Moore-FSM: Implemented Equations

```

O <= (STATE(0).FBK.LFBK AND STATE(1).FBK.LFBK);
FDCPE_STATE0: FDCPE port map (STATE(0),STATE_D(0),CLK,RESET,'0');
STATE_D(0) <= ((NOT ENABLE AND STATE(0).LFBK)
OR (ENABLE AND NOT I(1)) AND I(0))
OR (ENABLE AND I(1) AND NOT I(0) AND NOT STATE(0).LFBK
AND STATE(1).LFBK));

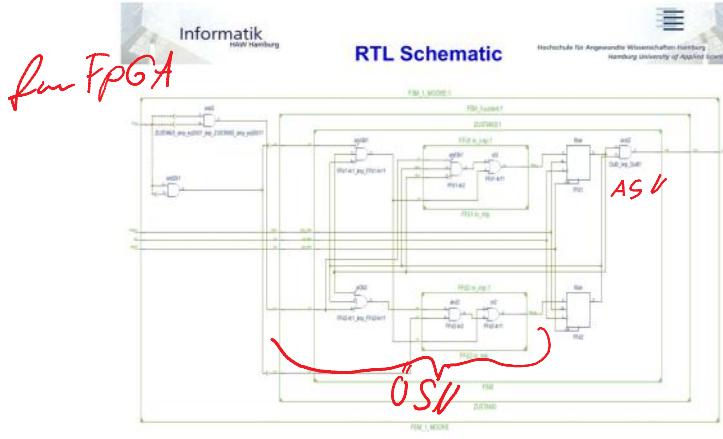
FDCPE_STATE1: FDCPE port map (STATE(1),STATE_D(1),CLK,RESET,'0');
STATE_D(1) <= ((NOT ENABLE AND STATE(1).LFBK)
OR (I(1) AND NOT I(0) AND NOT STATE(0).LFBK AND STATE(1).LFBK)
OR (ENABLE AND I(1) AND I(0) AND STATE(0).LFBK AND
NOT STATE(1).LFBK));

Register Legend: FDCPE (Q,D,C,CLR,PRE);

```

*CP/D - DFF
ohne Enable
Eingang*

13 CE - DS 2 FSM

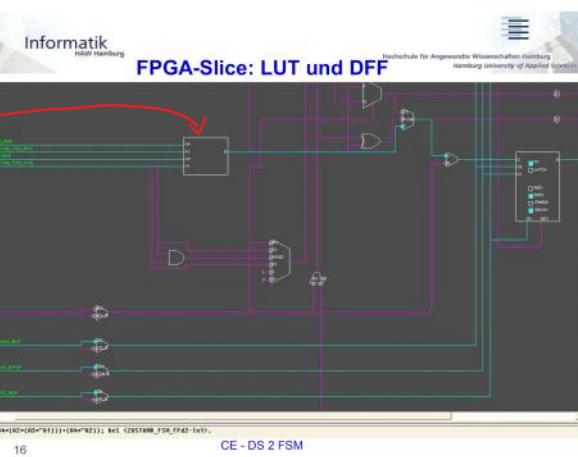


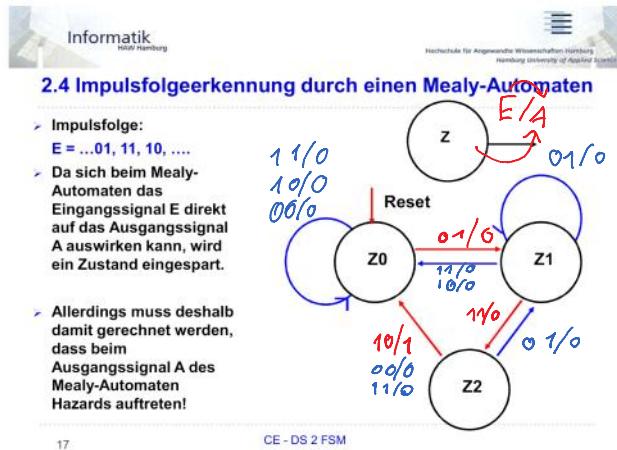
Informatik
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Synthese-Reportauszug

```
-- Synthesizing Unit <FSM_1_MOORE>.
  Related source file is "C:/Users/Dr. B.
  Schmitz/OneDrive - Hochschule für Angewandte Wissenschaften/Hochschule für Angewandte Wissenschaften/Hamburg University of Applied Sciences/CE-DS 2 FSM/seq_source.vhd".
  Found finite state machine <FSM_0> for signal <USTAND>.
  -----
  | States          | 4
  | Transitions    | 3
  | Inputs          | 1
  | Outputs         | 3
  | Clock           | CLK
  | Clock enable   | ENABLE (rising_edge)
  | Reset            | RESET (positive)
  | Reset type      | asynchronous
  | Reset id        | z0
  | Preset Up State | z0
  | Encoding         | sequential
  | Implementation   | LUT
  -----
  Summary:
  Inferred 1 Finite State Machine(s).
  Unit <FSM_1_MOORE> synthesized successfully.
  * Advanced HDL Synthesis
  -----
  Optimizing FSM <USTAND>/FSM on signal <USTAND(1:2)> with sequential encoding.
  -----
  State | Encoding
  z0   | 00
  z1   | 01
  z2   | 10
  z3   | 11
  -----
  CE - DS 2 FSM
```

z₀ *z₁* *z₂* *z₃*





Informatik

VHDL-Modell der Mealy-FSM

```

entity FSM_1_MEALY is
  port( CLK, RESET, ENABLE : in bit; -- sekundäre Eingangssignale
        E : in bit_vector(1 downto 0); -- Impulsfolge
        A : out bit ); -- Ausgangssignal
end FSM_1_MEALY;

architecture SEQUENCER of FSM_1_MEALY is
  type ZUSTANDE is (Z0, Z1, Z2);
  signal ZUSTAND, FOLGE_Z: ZUSTANDE;
  attribute GATE_RECOVERY_STATE: string;
  attribute GATE_RECOVERY_STATE of ZUSTAND: signal is "Z1";
begin
  Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
  begin
    UE_ZN: process(E, ZUSTAND)
    begin
      FOLGE_Z <- ZN after 5 ns; -- Zustandberechnung
      case ZUSTAND is
        when Z0 => if E = "01" then FOLGE_Z <- Z1 after 5 ns;
                     end if;
        when Z1 => if E = "11" then FOLGE_Z <- Z2 after 5 ns;
                     elsif E = "01" then FOLGE_Z <- Z1 after 5 ns;
                     end if;
        when Z2 => if E = "01" then FOLGE_Z <- Z1 after 5 ns;
                     end if;
        when others => null;
      end case;
    end process;
    end process;
    A: process(ZUSTAND)
    begin
      A <- '0' after 5 ns; -- Defaultzuweisung
      if (ZUSTAND = Z2 and E = "10") then A <='1' after 5 ns;
      end if;
    end process;
  end process;
end SEQUENCER;

```

→ für Speicher des FPGAs

CE - DS 2 FSM



Testbench instanziert den Mealy-Automaten

Testobjekt als Instanz (DUT) in einer Entity ohne äußere Schnittstellen.

```

entity TEST_B_ME is
end TEST_B_ME;

architecture SEQUENZ of TEST_B_ME is
component FSM_1_MEALY is
port(CLK, RESET, ENABLE : in bit; -- sekundäre Eingangssignale
     E : in bit_vector(1 downto 0); -- Impulsfolge
     A : out bit); -- Ausgangssignal
end component;

signal CLK_I, RESET_I, ENABLE_I, A_I: bit; -- Interne Signale
signal E_I: bit_vector(1 downto 0);
begin
  CLOCK: process
    begin
      CLK_I <= '0'; wait for 25 ns;
      CLK_I <= '1'; wait for 25 ns;
    end process CLOCK;
  
```

CE - DS 2 FSM

19



Testbench instanziert den Mealy-Automaten

Hazard in E = 10,00,10 bei:

```

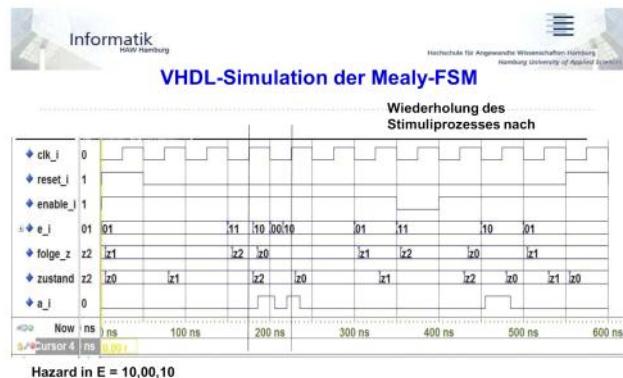
ABLAUF: process
begin
ENABLE_I <= '1'; RESET_I <= '1'; E_I <= "01"; wait for 50 ns;
RESET_I <= '0';
E_I <= "11"; wait for 100 ns;
E_I <= "10"; wait for 30 ns;
E_I <= "00"; wait for 20 ns;-- Hazard
E_I <= "01"; wait for 15 ns;-- A_I <= '1'
E_I <= "01"; wait for 85 ns;-- A_I <= '1'
E_I <= "01"; wait for 50 ns;
ENABLE_I <= '0';
E_I <= "11"; wait for 50 ns;-- E1 fest
ENABLE_I <= '1';
E_I <= "10"; wait for 50 ns;
E_I <= "01"; wait for 50 ns;
end process ABLAUF;
  
```

DUT: entity work.FSM_1_MEALY(SEQUENZ) -- Instanzierung der Mealy FSM
 port map(CLK => CLK_I, RESET => RESET_I, ENABLE => ENABLE_I,
 E => E_I, A => A_I); -- formal => actual
 end SEQUENZ;

Wiederholung der Stimulussequenz ab:

20

CE - DS 2 FSM



21 CE - DS 2 FSM

```

Informatik
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Synthese-Reportauszug

Synthesizing Unit: <FSM_1_MEALY>
Relative source file is "C:/Users/.../Documents/A_MG1_ISE_work/Modelsim_source/Lectures/cel...
    seq_mealy.vhd".
Found finite state machine <FSM_0> for signal <ZUSTAND>.
-----|-----
| States | 3
| Transitions | 3
| Inputs | 2
| Outputs | 1
| Clock | CLK
| Clock enable | ENABLE
| Reset | RST
| Reset type | asynchronous
| Reset State | #0
| Power Up State | #2
| Recovery State | #3
| Encoding | sequential
| Implementation | LUT
-----|-----
Summary) inferred 1 Finite State Machine(s).
Unit <FSM_1_MEALY> synthesized.
* Advanced HDL Synthesis
Optimizing FSM <ZUSTAND/FSM> on signal <ZUSTAND[1:2> with sequential encoding.
-----|-----
State | Encoding
-----|-----
#0 | 10
#1 | 01
#2 | 00
-----|-----

```

22 CE - DS 2 FSM

*Safe Recovery Verhalten in der Timingsim überprüfen
→ mit do U-File / Konsole angebe force ; no race*



Mealy-FSM: Implemented Equations

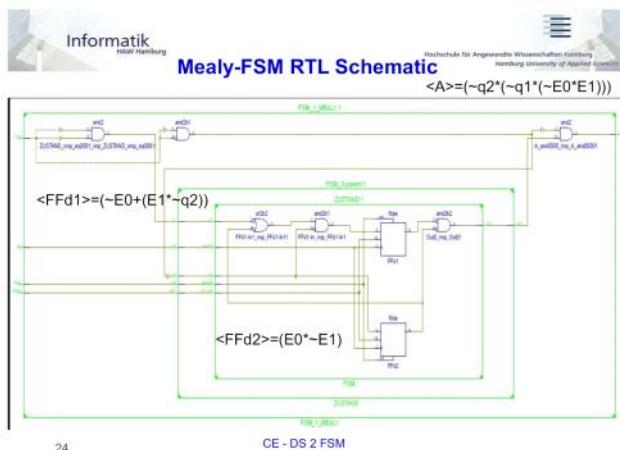
A <= (E(1) AND NOT E(0) AND NOT ZUSTAND_FSM_FFd2.LFBK AND
NOT ZUSTAND_FSM_FFd1.LFBK);

Z₁ FDCPE_ZUSTAND_FSM_FFd1: FDCPE
port map (ZUSTAND_FSM_FFd1,ZUSTAND_FSM_FFd1_D,CLK,'0',RESET);
ZUSTAND_FSM_FFd1_D <= ((NOT E(0) AND ENABLE) OR
(NOT ENABLE AND ZUSTAND_FSM_FFd1.LFBK) OR
(E(1) AND ENABLE AND NOT ZUSTAND_FSM_FFd2.LFBK));

Z₀ FDCPE_ZUSTAND_FSM_FFd2: FDCPE
port map (ZUSTAND_FSM_FFd2,ZUSTAND_FSM_FFd2_D,CLK,RESET,'0');
ZUSTAND_FSM_FFd2_D <= (NOT ENABLE AND ZUSTAND_FSM_FFd2.LFBK) OR
(NOT E(1) AND E(0) AND ENABLE);

23

CE - DS 2 FSM



24

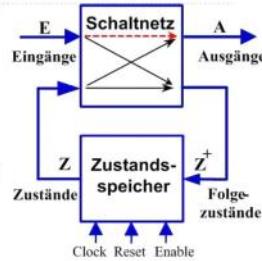
CE - DS 2 FSM



2.5 Zwei-Prozess VHDL Automatenbeschreibung

Ausgangspunkt vereinfachter Automatenmodelle ist die Huffmann-Normalform.

- Ein Zustandsspeicher
- Ein kombiniertes Schaltnetz, das das Übergangs- und das Ausgangsschaltnetz so zusammenfasst, dass die Zustandsabfrage nur einmal realisiert wird.
- Eine direkte Verbindung der Eingänge **E** auf die Ausgänge **A** (gestrichelt) existiert nur, falls ein Mealy-Verhalten modelliert werden soll.
- Entsprechend lassen sich VHDL-Modelle von Zustandsautomaten auch mit zwei Prozessen realisieren. Medvedev-Automaten können sogar mit einem einzigen Prozess aufgebaut werden.



CE - DS 2 FSM

25



Kombiniertes ÜSN und ASN

```
UE_A_SN: process(E, ZUSTAND)-- Folgezustände= u. Ausgangsberechnung
begin
    A <- '0' after 5 ns;
    FOLGE_Z<= Z0 after 5 ns; -- Defaultzuweisungen
    case ZUSTAND is
        when Z0 => if E = "01" then
                        FOLGE_Z<= Z1 after 5 ns;
                    end if;
        when Z1 => if E = "11" then
                        FOLGE_Z<= Z2 after 5 ns;
                    elsif E = "01" then
                        FOLGE_Z<= Z1 after 5 ns;
                    end if;
        when Z2 => if E = "10" then
                        FOLGE_Z<= Z3 after 5 ns;
                    elsif E = "01" then
                        FOLGE_Z<= Z1 after 5 ns;
                    end if;
        when Z3 => A <= '1' after 5 ns; -- Moore-Ausgang
                    if E = "01" then
                        FOLGE_Z<= Z1 after 5 ns;
                    end if;
    end case;
end process UE_A_SN;
end SEQUENZI;
```

26

*Bsp für Lohnaufgaben
berücksichtigen
nur 1 mal
decodieren*

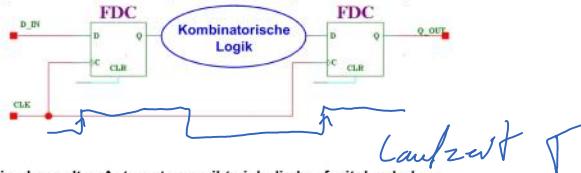
- kommunizierende FSMA im gekoppelten Sup-Automata
 → ASN → USM sind voneinander
 → entkoppelt, damit f_{max} nicht reduziert wird

21.09.2012



2.6 Entkopplung von Zustandsautomaten

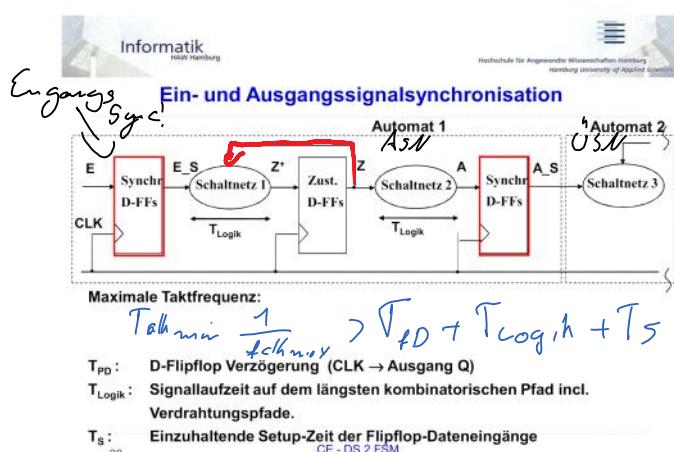
- Die maximale Taktfrequenz eines synchronen digitalen Systems wird durch die längste Laufzeit eines Signals durch die kombinatorische Logik zwischen je zwei Flipflops bestimmt (kritischer Pfad): RTL



- Bei gekoppelten Automaten ergibt sich die Laufzeit durch den kombinatorischen Pfad als Summe der Laufzeiten durch das Ausgangsschaltnetz des ersten Automaten und der durch das Übergangsschaltnetz des zweiten Automaten!

27

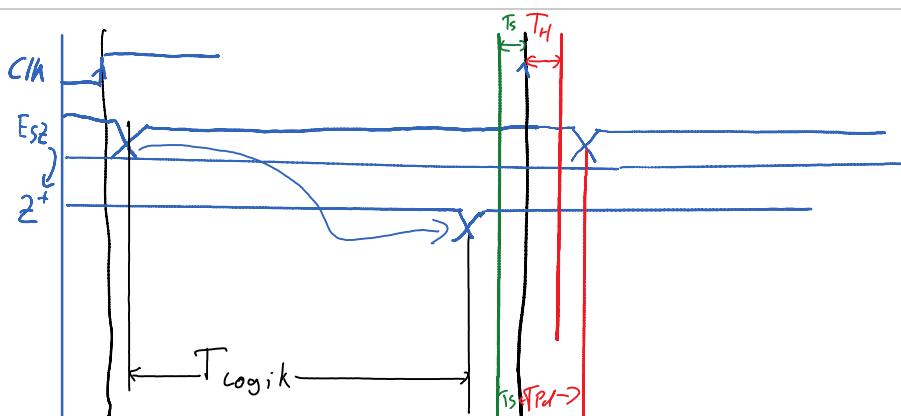
CE - DS 2 FSM



28

$$T_H \approx T_{hold} \quad T_S \approx T_{setup}$$

14



Einhalten des Setup-Intervalls, damit Clash-Zeit verhindert

von L - Transaktionen und Clock-Flanken \rightarrow keine metastabilen Zustände der DFF's entstehen.

MSZ: $0 < Q < V_{th}$; V_{DD}

Metastabile

$Q \xrightarrow{t_{crossover}} \text{High}$ Übergang nach unbekannter Zeit

WS12

21.09.2012



Moore-FSM mit synchronisierten Schnittstellen

```
entity FSM_sync is
  port( CLK, RESET : in bit;
        E: in bit_vector(1 downto 0);
        A_S: out bit ); -- Synchr. Ausgangssignal
end FSM_sync;

architecture SEQUENZ of FSM_sync is
type ZUSTAENDE is (Z0, Z1, Z2, Z3);
signal ZUSTAND, FOLGE_Z: ZUSTAENDE;
signal E_S: bit_vector(1 downto 0); -- Synchr. Eingangssignal
signal A: bit; -- Async. Ausgangssignal
begin
  SYNC: process(CLK, RESET) -- E/A-Synchronisation
  begin
    if RESET = '1' then
      E_S <= (others->'0') after 5 ns;
      A_S <= '0' after 5 ns;
    elsif CLK='1' and CLK'event then
      E_S <= E after 5 ns;
      A_S <= A after 5 ns;
    end if;
  end process SYNC;
  CE - DS 2 FSM
  29
```



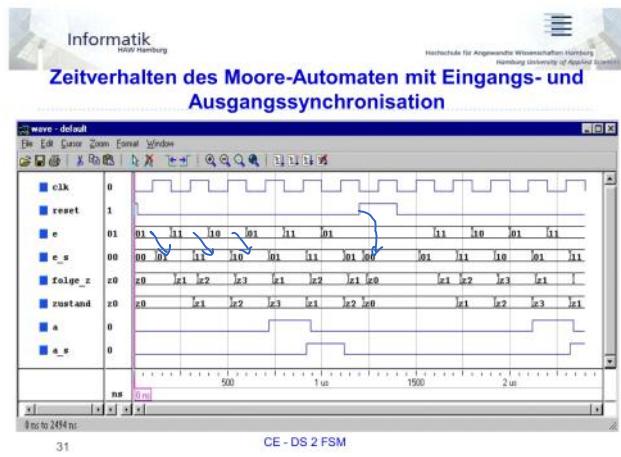
Huffman-Normalform mit synchronisierten Eingängen

```
Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
begin
  if RESET = '1' then
    ZUSTAND <= Z0 after 5 ns;
  elsif CLK = '1' and CLK'event then
    ZUSTAND <= FOLGE_Z after 5 ns;
  end if;
end process Z_SPEICHER;

UE_A_SN: process(E_S, ZUSTAND)-- Folgezustands- u. Ausgangsberechnung
begin
  ...
  ...
end process UE_A_SN;
end SEQUENZ;
```

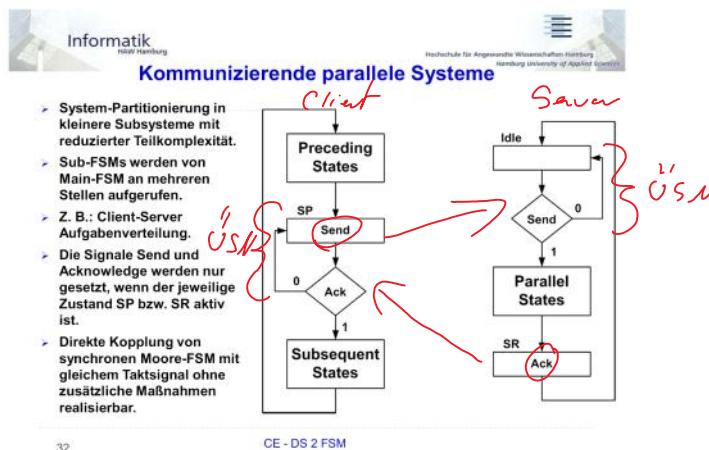
30

CE - DS 2 FSM



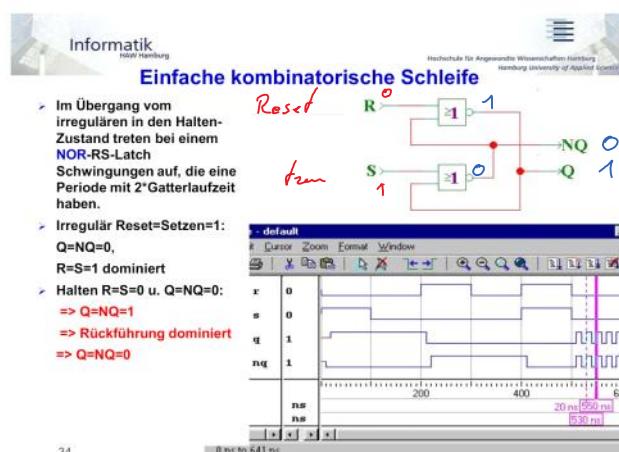
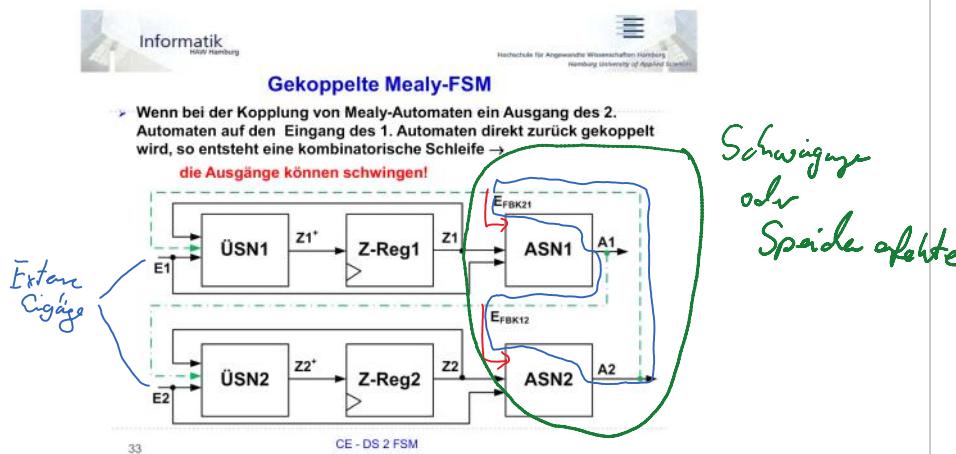
31

CE - DS 2 FSM



32

CE - DS 2 FSM



Prof. Dr. B. Schwarz

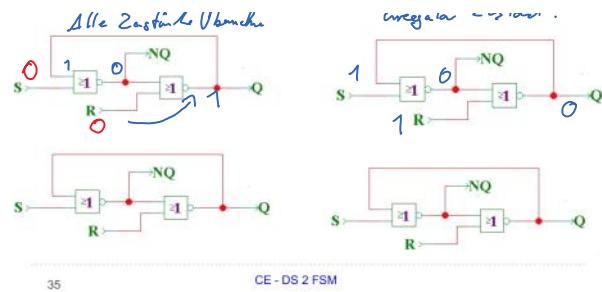
17

WS12

21.09.2012

R	S	Q	Q^+	$\bar{Q}^+ = NQ$
0	1	X	1	
1	0	X	0	
0	0	Q	Q	Solen Rücksetz
1	1	X	0	Halten (irregular)





35



2.7 Sequentieller Addierer

- Die erste Hauptaufgabe eines Automatenentwurfs liegt bei der Umsetzung einer **textuellen Spezifikation in ein Zustandsdiagramm**.
- Dazu ist zunächst zu prüfen:
 - Welche Eingangssignale sind synchron, welche asynchron?
 - Wie viele Zustände sind erforderlich, und welche Bedeutung haben diese?
 - Muss der Automat (aus Geschwindigkeitsgründen) als Mealy-Automaten realisiert werden oder reicht ein Moore-Automat mit einem Takt mehr Latenz?
 - Ist es erforderlich, die Anzahl der Zustände in einem zweiten Schritt systematisch zu minimieren?
 - Welche Zielhardware (FPGA oder (C)PLD) ist vorgesehen?
 - Ist für die Anwendung eine sichere Rückkehr aus möglicherweise vorhandenen Pseudozuständen sicher zustellen?
- Bei der Erstellung des Zustandsdiagramms werden zunächst die "normalen" Zustandsübergänge betrachtet und hinterher die Sonderfälle.
- Der sequentielle Addierer ist ein Beispiel für eine in den Automaten integrierte Arithmetikfunktion: schnelle **kombinierte Lösung** für kleine Aufgaben.

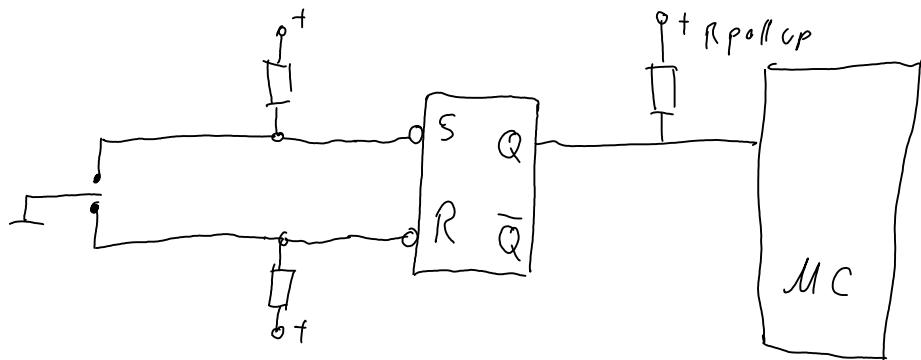
36

CE - DS 2 FSM

Prof. Dr. B. Schwarz

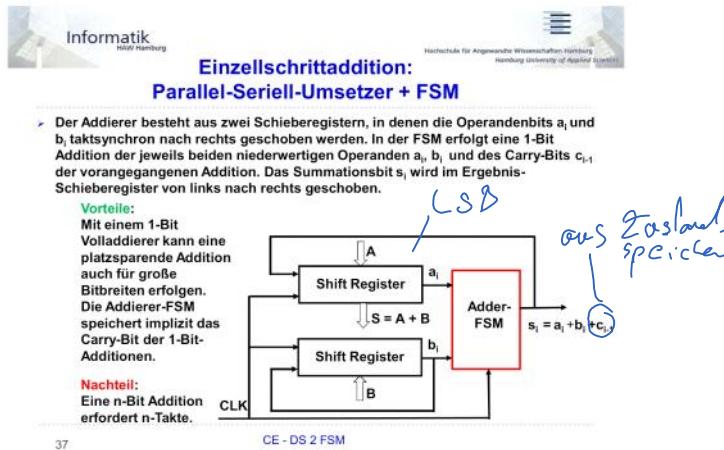
18

Entpfeilung von Mechanischen Schaltern!



WS12

21.09.2012



37



Abhängig vom Wert des Carry-Bits c_{i-1} der jeweils vorherigen 1-Bit Addition realisiert die FSM unterschiedliche Ergebnisse und Zustandsübergänge.

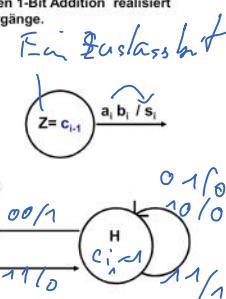
Bedeutung der Zustände:
G: Carry-In = '0'
H: Carry-In = '1'

Zustandsfolge- und Ausgangstabelle:

$Z = a_i b_i$	$Z' =$	s_i
G 0 0	G	0
G 0 1	G	1
G 1 0	G	1
G 1 1	H	0
H 0 0	G	1
H 0 1	H	0
H 1 0	H	0
H 1 1	H	1

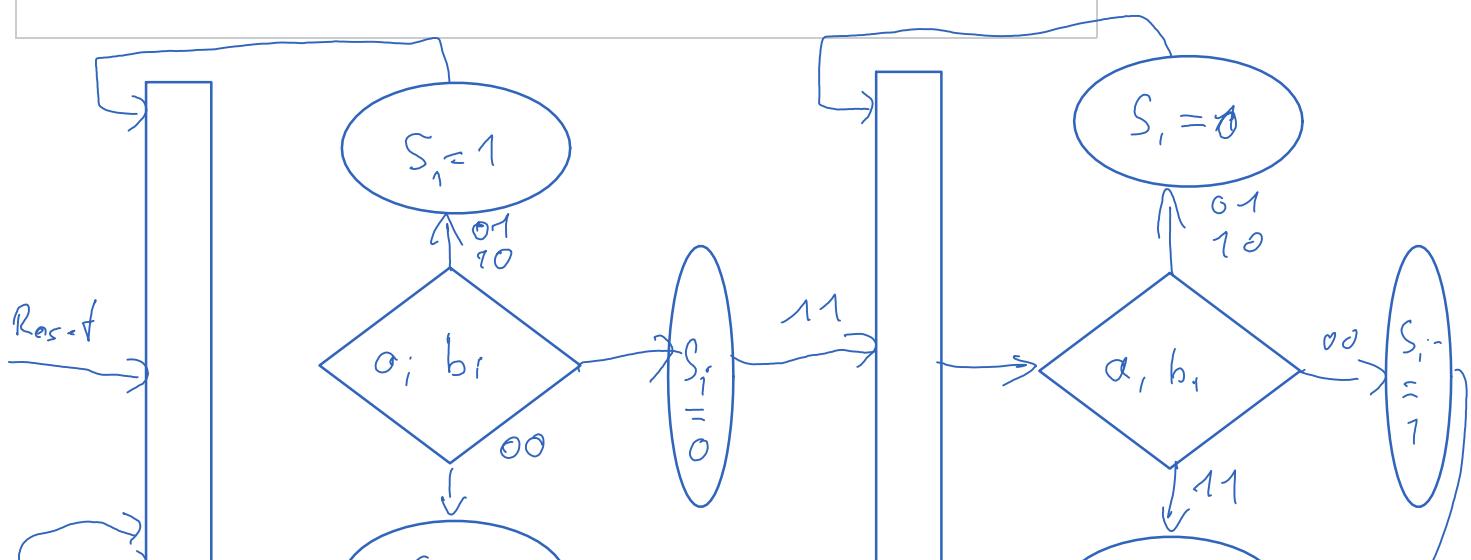
CE - DS 2 FSM

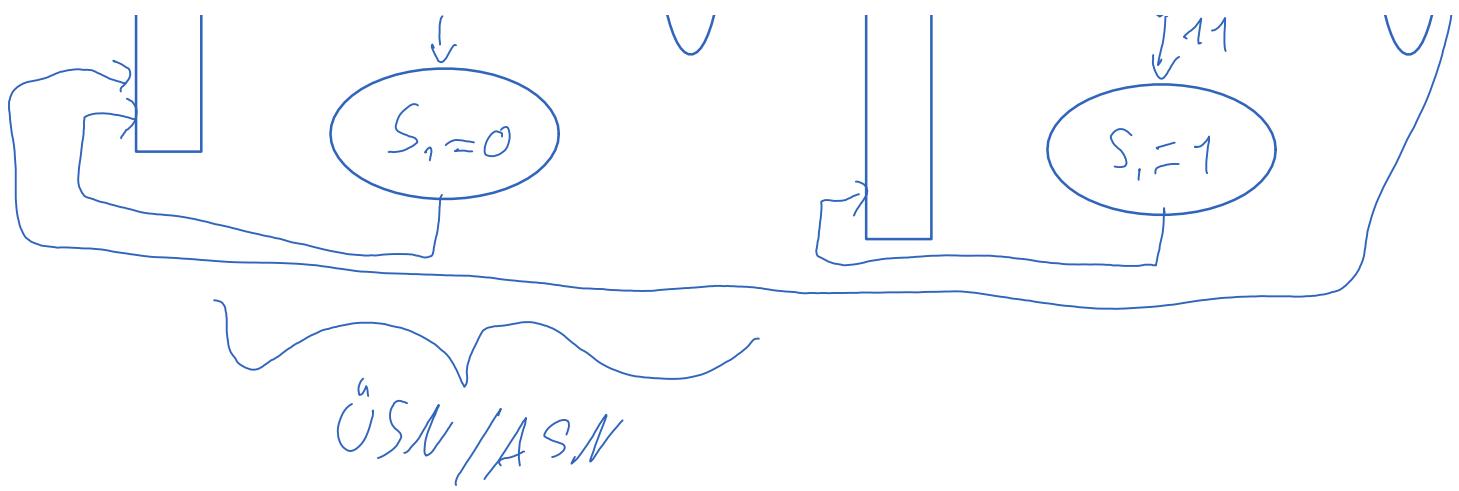
38



Prof. Dr. B. Schwarz

19





$\overset{\wedge}{\text{OSN}}/\text{ASN}$


Informatik
 Hochschule für Angewandte Wissenschaften Hamburg
 Hamburg University of Applied Sciences

Entwurf eines Moore-Automaten

Beim Moore-Automat darf das Ausgangssignal s_i nur vom Zustand $Z = c_{i,1}$ abhängen → Aufspaltung der Mealy-Zustände G und H in je zwei Zustände G_0, G_1 bzw. H_0, H_1 .

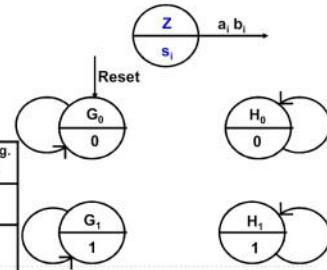
Gewählte Zustandscodierung:

Z	Q1	Q0
G_0	0	0
G_1	0	1
H_0	1	0
H_1	1	1

Zustand	Folgezustand Z'	Ausg.
Z	$a_i b_i = 00 \quad 01 \quad 10 \quad 11$	s_i
G_0		
G_1		
H_0		
H_1		

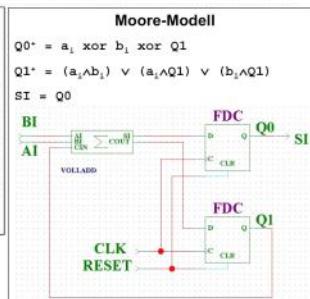
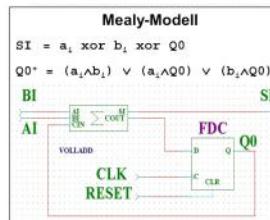
CE - DS 2 FSM

39



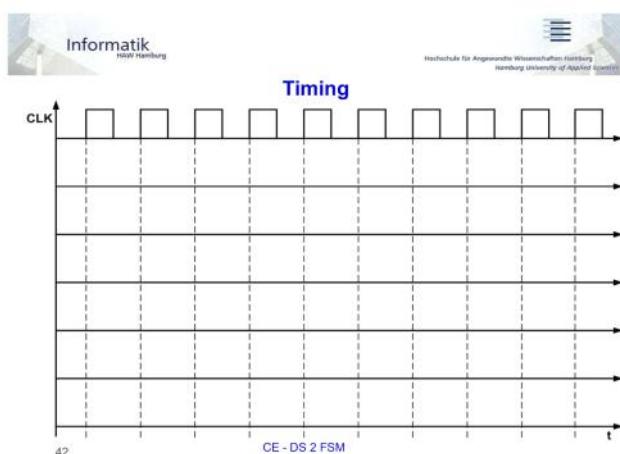
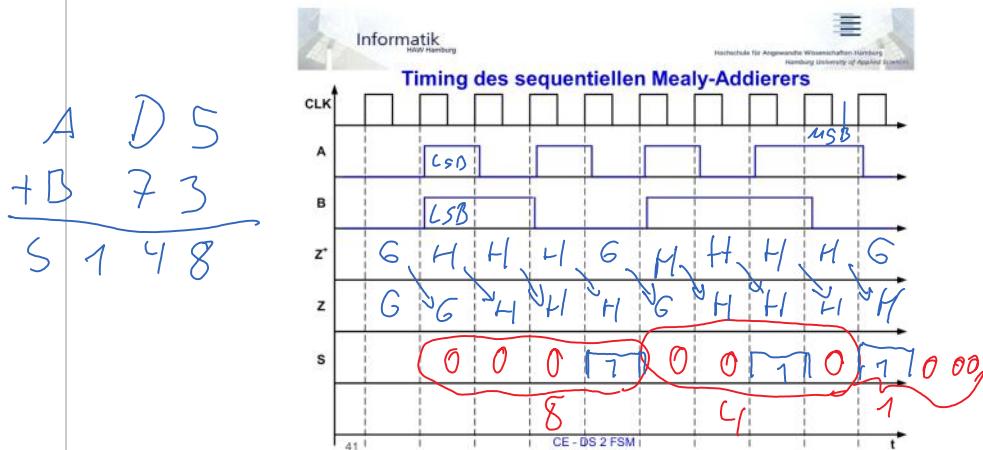

Informatik
 Hochschule für Angewandte Wissenschaften Hamburg
 Hamburg University of Applied Sciences

Syntheseergebnis der sequentiellen Addierer



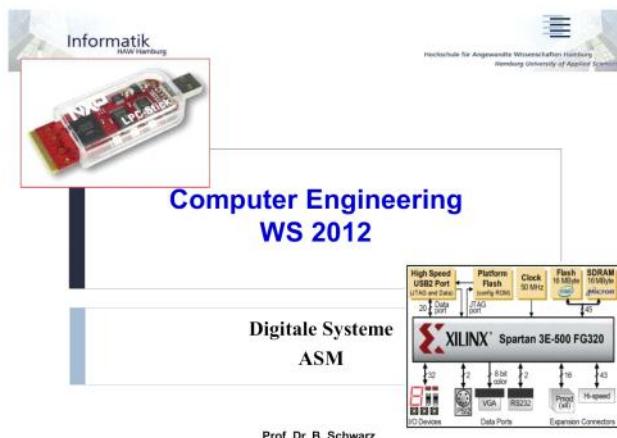
40

CE - DS 2 FSM



WS 12

21.09.2012

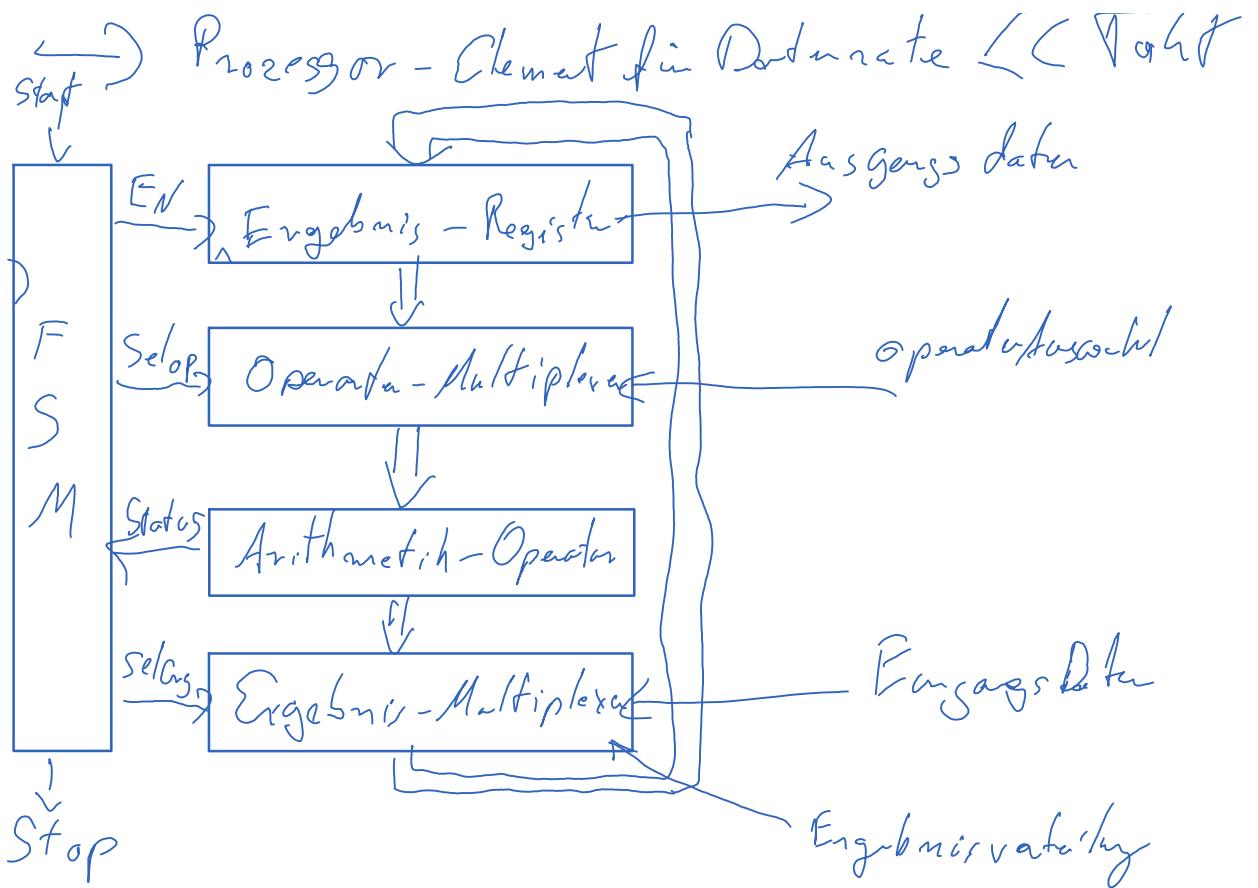


3. Architekturentwurf ASM-Diagramm für Multizyklus-Datenpfade

Prozessorelement für eine S-Kurvenapproximation
Gemeinsame Nutzung von Hardware Funktionseinheiten
(Ressource Sharing)
Gemeinsame Register- / Speicher-Nutzung
(Register Sharing)
VHDL-Modell des PE mit Fixed-Point Arithmetik im Q-Format
Datenpfad in Pipelinestruktur

Dynamische Systeme mit Rückführung
Multizyklusdatenpfad für ein Filter 1. Ordnung
Ergebnisübergabe an Folgezyklen

ASM-Kapitel → Multizyklus-Datenpfad
Ziel: Algorithmus mit begrenzten Anforderungen an Ressourcen
Sequenzierbar





3.1 Entwurf digitaler Systeme mit ASM-Diagrammen (ASM = Algorithmic State Machine)

- ASMs dienen der Beschreibung von Zustandsautomaten auf einer höheren, algorithmischen Abstraktionsebene. Sie beschreiben die sequentiellen Operationen eines digitalen Systems, das einen Algorithmus implementiert. Ein Algorithmus ist eine gesteuerte Sequenz von Aktionen und/oder Berechnungsschritten mit zyklischer Aktualisierung der Eingangsvariablen.
- ASMs eignen sich insbesondere für Multizyklus-Datenpfade mit einer Sequenz von Rechenschritten, die jeweils bei Aktualisierung der Eingangsgrößen neu gestartet wird.
- Es wird also vorausgesetzt, dass gilt Datenrate $\ll f_{clk}$

3

CE - DS 3.1 ASM

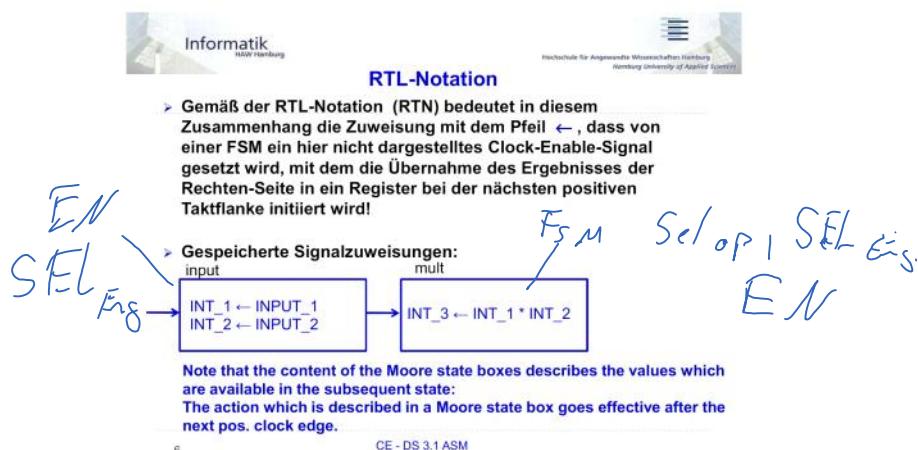
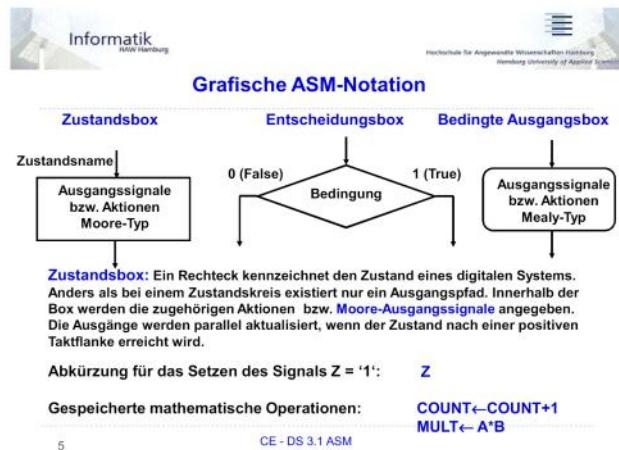


ASMs

- ASM Diagramme ähneln den Flussdiagrammen, die in den Anfängen der Computerprogrammierung genutzt wurden. Im Gegensatz zu den Flussdiagrammen beschreibt eine ASM ein zeitlich getriggertes Verhalten, da die Übergänge zwischen den Zuständen durch die Flanken eines Taktsignals ausgelöst werden.
- ASM-Diagramme werden mit drei Elementen aufgebaut.

4

CE - DS 3.1 ASM





Bedingte Verzweigungen – Mealy Ausgänge

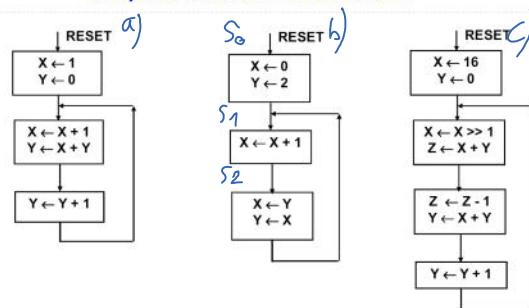
- **Entscheidungsbox:** Kennzeichnet die Bedingung für einen Zustandsübergang. Innerhalb der Box wird die zu testende Bedingung von Eingangssignalen angegeben. Z.B.:
 - W bedeutet, dass das Eingangssignal W zu testen ist
 - $W_1 \vee W_2$ W_1 oder W_2 müssen wahr sein
- **Bedingte Ausgangsbox:** Ein abgerundetes Rechteck beschreibt ein Mealy-Ausgangssignal. Bedingte Ausgangsboxen sind Ausgänge von Entscheidungsboxen, in denen die zu testenden Eingangssignale stehen. Der Ausgangspfad kann zu einer Zustandsbox oder zu einer weiteren Entscheidungsbox führen.

7

CE - DS 3.1 ASM



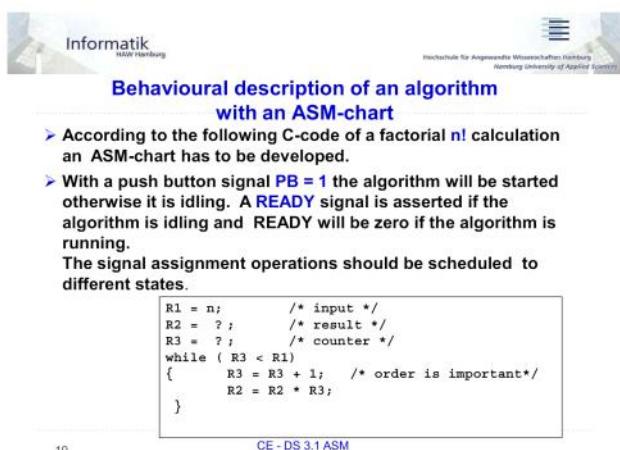
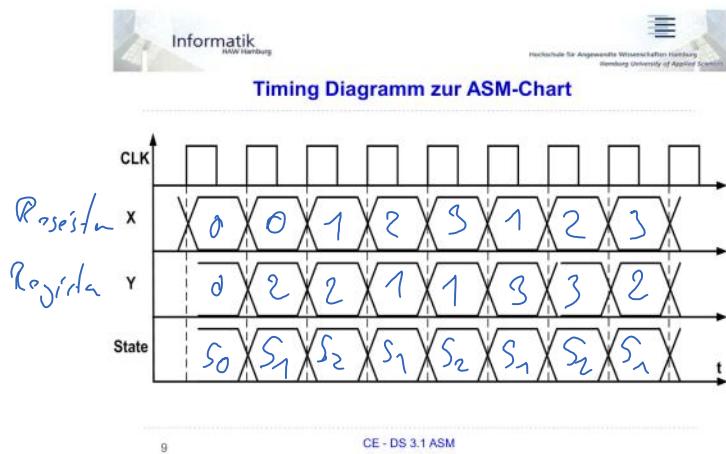
Beispiele zu ASM-Charts mit RTN



8

CE - DS 3.1 ASM

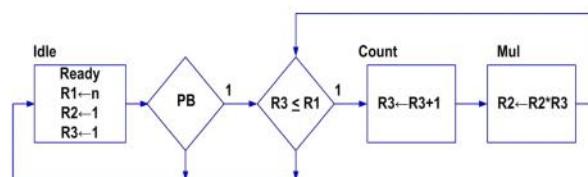
b)



10

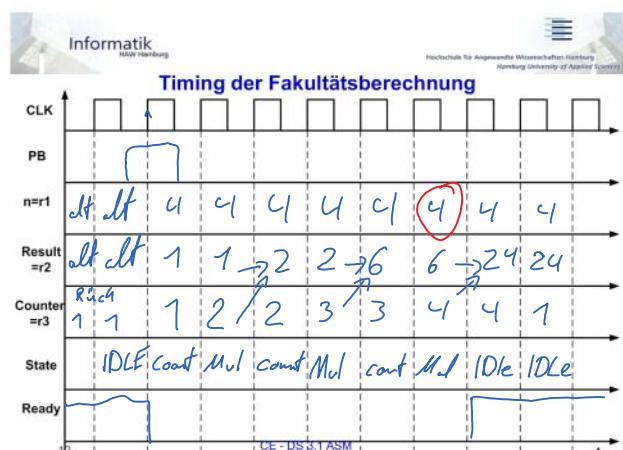


ASM Chart der Fakultätsberechnung



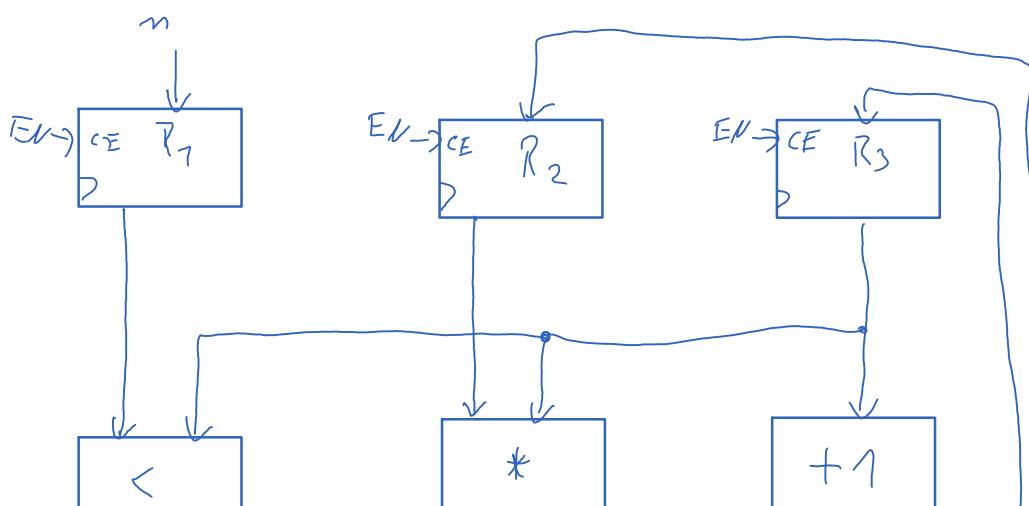
11

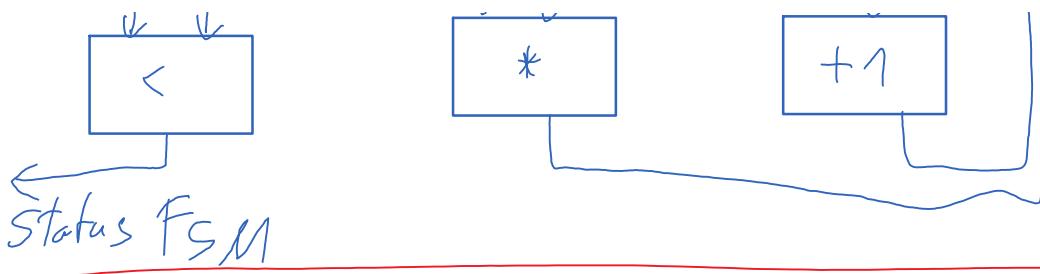
CE - DS 3.1 ASM



Prof. Dr. B. Schwarz

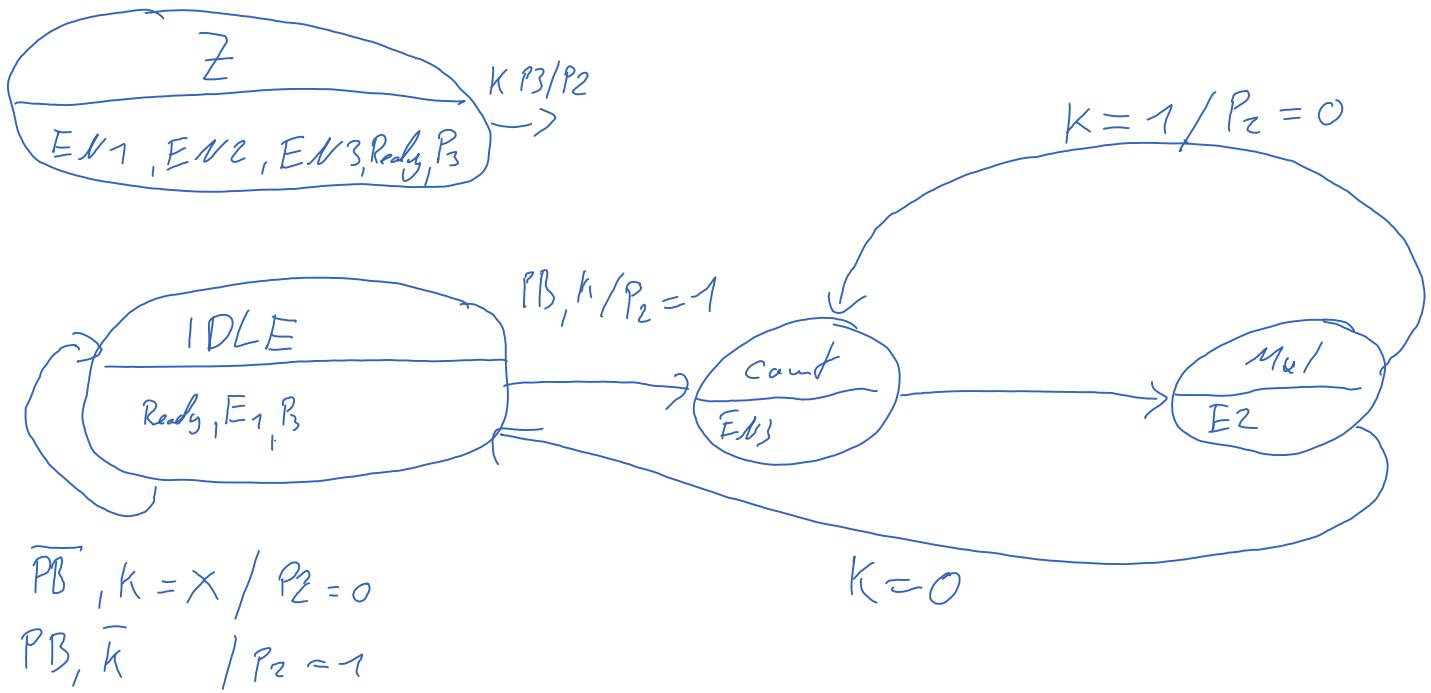
6





y) FSM , d.h. Zustandsdiagramm

Mealy- FSM



Registers - Prozess R2

Process ((1k))
begin

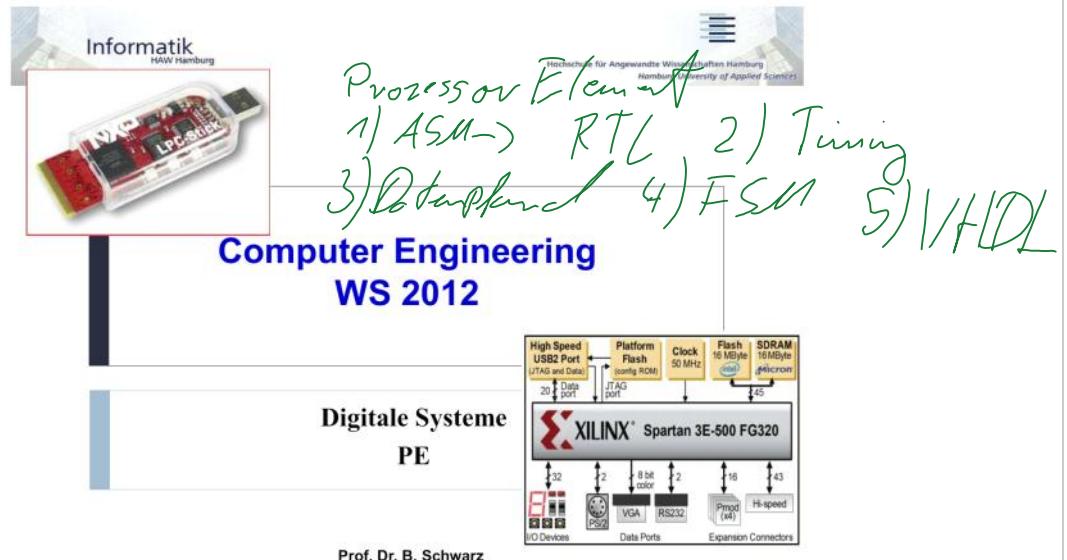
```

if c/h' event and c/h = '1' then
  if R2 = '1' then R2 <- others => '0', 1);
  else if EN = '1' then R2 <-

```

WS 12

21.09.2012



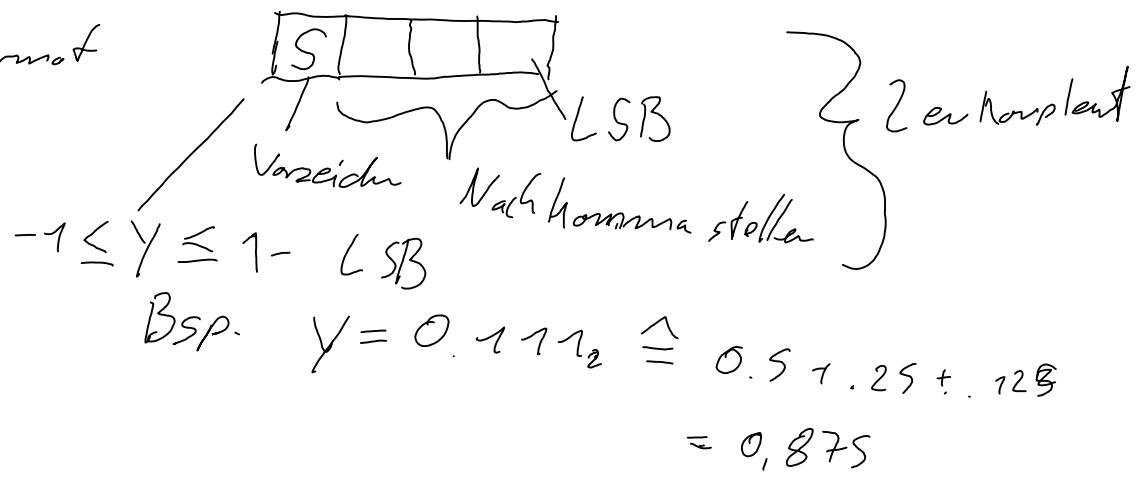
3. Architektsynthese ASM-Diagramme

Prozessorelement für eine S-Kurvenapproximation
Gemeinsame Nutzung von Arithmetik-Funktionseinheiten
(Resource Sharing)
Gemeinsame Register- / Speicher-Nutzung
(Register Sharing)
VHDL-Modell des PE mit Fixed-Point Arithmetik im Q-Format

Datenpfad in Pipelinestruktur

Dynamische Systeme mit Rückführung
Multizyklusdatenpfad für ein Filter 1. Ordnung
Ergebnisübergabe an Folgezyklen

C23-Format



3.2 Entwurf eines Prozessorelementes

- In diesem Abschnitt werden die Entwurfsschritte der Architektursynthese am Beispiel eines Prozessorelementes vorgestellt:
Spezialprozessor zur S-Kurvenapproximation für eine Motoransteuerung.
- Am Beispiel dieses Spezialprozessors werden die folgenden Techniken dargestellt:
Multizyklus-Datenpfad für Datenraten << Systemfrequenz
Entwurf von ASM-Charts
Gemeinsame Nutzung von Hardware Funktionseinheiten (Ressource-Sharing)
Gemeinsame Register- / Speicher-Nutzung (Register-Sharing)

3

CE - DS 2

Entwurf eines Prozessorelementes

- Spezialprozessor
Operanden- und Ergebnispfad-Steering mit Multiplexer
Zustandsdiagramm für den Steuerpfad
- VHDL-Modellierung mit Integer-Arithmetik im Q-Format

4

CE - DS 2

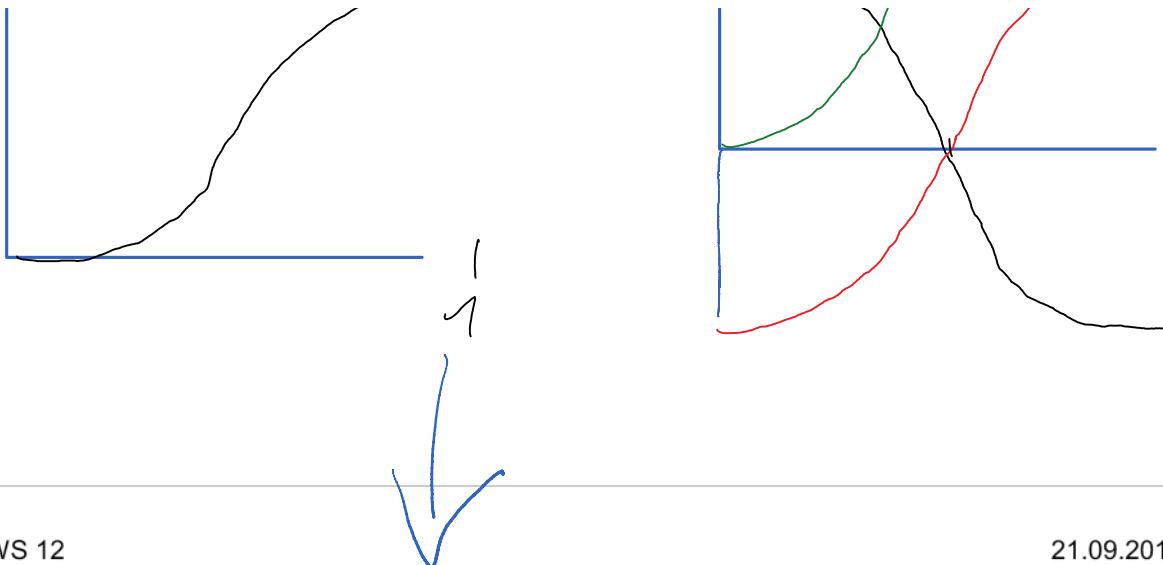
Prof. Dr. B. Schwarz

$$1 - \cos(x)$$

$$\cos(x)$$

2

$$-\cos(x)$$



WS 12

21.09.2012



Anwendungshintergrund (1)



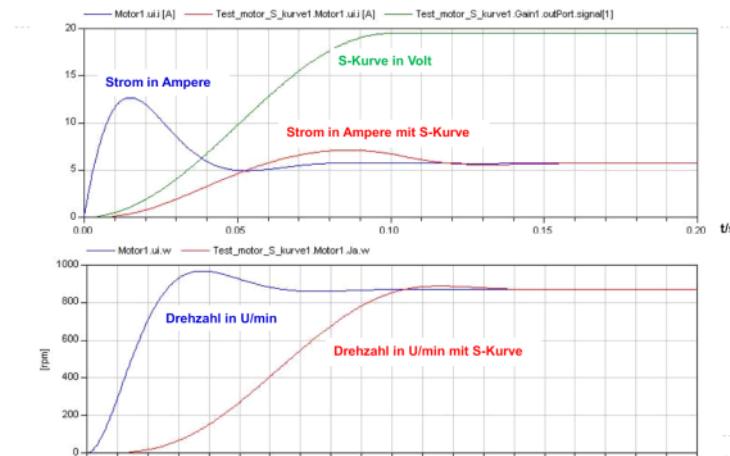
- Zur Begrenzung der Anlaufströme für Antriebsmotore wird die Motorspannung nicht sprungförmig, sondern mit einer S-Funktion eingeschaltet.
- Diese S-Funktion kann mit einer cos-Funktion gebildet werden:
 $S(x) = (1 - \cos(x))/2$, wobei das Argument x mit einer Rampenfunktion zu erzeugen ist.
- Die in Antriebseinheiten zum Einsatz kommenden µController können für zeitkritische Realtime-Anwendungen ggf. keine rechenintensiven Floatingpoint-Bibliotheken verwenden, sodass nur Integer-Arithmetik in Frage kommt.
- Eine S-Funktion ist also durch ein Polynom höherer Ordnung in x zu approximieren.

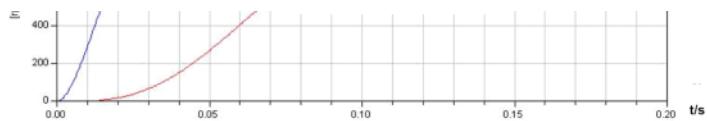
5

CE - DS 2



Anwendungshintergrund (2)





Entwurfsvorgaben (1)

- Die gleiche Vorgehensweise ist auch bei einer Realisierung von Antriebsteuerungen mit rekonfigurierbaren FPGA-ICs erforderlich.
- Da die Abtastrate für solche Antriebssteuerungen um Größenordnungen geringer ist als die verfügbare Taktfrequenz der ICs, kann mit einem **Multizyklus-Datenpfad** und **Resource-Sharing** gearbeitet werden.
- Für die HW-Realisierung der Arithmetik sind deshalb nur **ein Multiplizierer** und **ein Addierer** einzusetzen.
- Die Berechnung der S-Funktion erfolgt pro Wert $S(x)$ in einer Sequenz von Rechenschritten, wobei Zwischenwerte in Registern gespeichert werden (vgl. D.D. Gajski S. 336 – 353).

7

CE - DS 2

Entwurfsvorgaben (2)

- **Taylor-Reihenentwicklung** zur Approximation der Funktion $S(x) = (1-\cos(x))/2$ um den Punkt $x = 0$.
- Nur die ersten beiden Terme mit geradem Exponenten werden berücksichtigt.
- Die approximierte Funktion kann für $x > 0$ nur im Bereich bis zum ersten positiven Maximum verwendet werden.
- Bestimmung der Koordinaten dieses Maximums: $S(x_0) \leq 1$. Mit Kenntnis dieses Maximums wird die Funktion normiert, damit das neue Maximum 1 wird.
- Die unabhängige Variable x der S-Funktion wird im Q7-Format mit einem Vorzeichenbit und 7 Nachkommabits repräsentiert : $-1 \leq x_Q < 1$.
- Substitution von $x = x_0 x_Q$ in der S-Funktion, sodass sich eine Darstellung $S(x_Q)$ mit neuen Koeffizienten ergibt.

8

CE - DS 2

S-Kurven-Approximation

- Taylor-Reihenentwicklung

$$S(x) = 0.25 x^2 - x^4 / 48$$

- Maximum bei x_0 :

$$S(x_0 = \sqrt{6}) = \frac{3}{4} = 1/k$$

- Maximalwertnormierung $S(x_0 = \sqrt{6}) \Rightarrow 1!$

$$S(x)^*k = 1/3 x^2 - 1/36 x^4$$

- Skalierung der x-Achse $x = x_0 x_Q$ mit $-1 \leq x_Q \leq 1$

$$S(x_Q) = 2^*(x_Q)^2 - (x_Q)^4$$

$$\begin{aligned} \frac{dS(x)}{dx} &= 0 = 2 \cdot \frac{1}{4} x - \frac{1}{48} x^3 \Big| \frac{1}{x} \\ &= \frac{1}{2} - \frac{1}{48} x^2 = 0 \Rightarrow x_0 = \sqrt{6} \end{aligned}$$

$$S(x_0) = \frac{1}{4} \cdot 6 - \frac{36}{48} = \frac{3}{2} - \frac{3}{4}$$

$$S(x_Q) = \frac{1}{3} x_0^2 x_Q^2 - \frac{1}{36} x_0^4 x_Q^4 = \frac{3}{4}$$

1 Mat, 1 Sub

CE - DS 2

Struktur- und Freigabekonzept

- Ein integrierte Rampenzähler ist ein 8 Bit-Zähler der die x_Q Eingangsgröße im Q7-Format bereitstellt.

- 128 Zählstufen sollen in 100 ms den Bereich 0 bis 1 liefern und danach angehalten werden.

- Ein 3 Bit-Zyklenzähler wird hier vorgeschlagen, der zusammen mit der FSM (Control Unit) als eine Art Frequenzteiler für den Rampenzähler wirkt.

- Die FSM startet zusammen mit diesem Zyklenzähler und wartet nach Rückkehr in den Idle-Zustand solange, bis der Zyklenzähler seinen wrap-around ausgeführt hat.

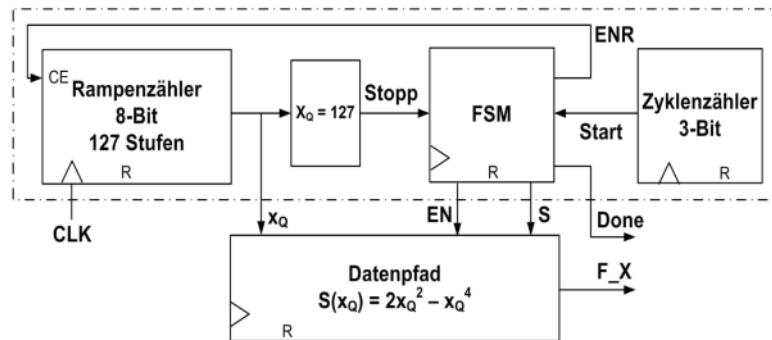
- Mit jedem wrap-around des Zyklenzählers erhält der Rampenzähler die Freigabe für das Inkrement einer Rampenstufe.

- Dieses Freigabekonzept für die Komponenten, die alle mit der gleichen Taktfrequenz betrieben werden, ist ein typischer Ansatz in so genannten Multiraten-Systemen.

10

CE - DS 2

Systemstruktur mit unterschiedlichen Taktbereichen



11

CE - DS 2

Zeitskalierung

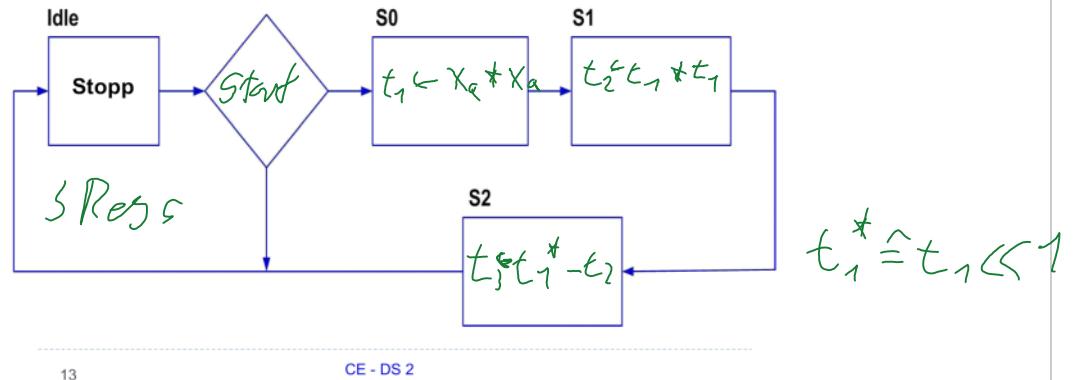
- 8-Bit Rampenzähler mit 128 Stufen: $x_{Q_{\max}} = 127$
- Anlaufintervall 100ms
- Effektive Taktfrequenz des Rampenzählers:
 $f_{R_{\text{clk}}} = 1/(100\text{ms}/128) = 1280 \text{ Hz}$
- 3-Bit Zyklenzähler:
 8 Zählzustände müssen das Intervall $1/f_{R_{\text{clk}}}$ abdecken
 $f_{\text{clk}} = 8 * 1280 \text{ Hz} = 10,24 \text{ kHz}$
 Taktfrequenz der Systemkomponenten

12

CE - DS 2

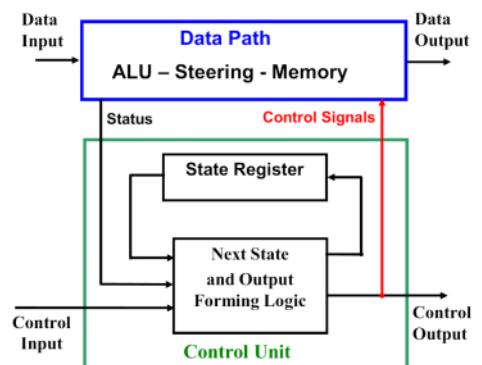
ASM des PEs zur Approximation der S-Kurve

➤ Zu realisierende Funktion: $S(x_Q) = 2 \cdot (x_Q)^2 - (x_Q)^4$



13

Strukturierung eines digitalen Systems als Prozessorelement in einen Datenpfad und einen Steuerpfad



14

CE - DS 2

Gemeinsame Register-Nutzung (Register Sharing)

- Nicht alle Zwischensignale müssen in einem eigenen Register abgelegt werden. Abhängig von der Nutzungsdauer können verschiedene Zwischensignale in einem Register abgelegt werden.
- Vom Zustandsautomaten wird festgelegt, welche Operationen in den verschiedenen Taktphasen mit den Registerwerten durchgeführt werden sollen.
- In einer Tabelle (signal lifetime table) wird die Lebensdauer der Zwischensignalwerte dargestellt. Dabei werden alle die Zustände gekennzeichnet, in denen das Zwischensignal noch benötigt wird. Begonnen wird mit dem Zustand, der auf die Abspeicherung im Zwischenregister folgt.

15

CE - DS 2

Signal-Lebensdauertabelle

	Idle	S ₀	S ₁	S ₂	Reg
t ₁	—	—	X	X	R ₁
t ₂	—	—	—	X	R ₂
y	X	X	X	X	R ₃

immer vorliegen
da Ausgang



Es ist zu berücksichtigen, dass das Ergebnis $y = S(xq)$ während eines kompletten Zyklus (8 Takte) am Ausgang zur Verfügung steht.

16

CE - DS 2

Datenpfad-Struktur

➤ Der Datenpfad ist in mehrere Funktionsebenen gegliedert:

1. Die **parallelen Register** nehmen die Zwischenergebnisse auf, sodass alte Werte ohne weitere Verwendung überschrieben werden können. Ggf. ist ein Multiplexer vorzuschalten, falls zu Beginn eines Zyklus die Eingangsgrößen eingespeichert werden müssen.
2. Die **Operanden** werden den Arithmetikfunktionen über **Multiplexer** zugeführt. Diese werden erforderliche, wenn eine Operation mehrmals verwendet wird und die Operanden aus unterschiedlichen Registern bzw. von extern zugeführt werden.
3. Die **AU-Elemente** stehen parallel zur Verfügung und verarbeiten die Operanden in einer Sequenz. Nur wenn der zu implementierende Algorithmus es zulässt, können die AU-Elemente auch parallel genutzt werden.
4. Eine weitere Multiplexer-Ebene verteilt die Rechenergebnisse auf die Register. Wenn bei der Signal-Register-Zuordnung auch die Operationen berücksicht werden, lassen sich hier Multiplexer einsparen.

17

CE - DS 2

Datenpfad der S-Kurvenapproximation

18

CE - DS 2



Integerarithmetik im Q-Format mit Signaltyp signed

- Signale und Variablen in digitalen Systemen auf Basis von FPGAs werden mit VHDL auf Bitvektorebene modelliert.
 - Für die unsigned and signed Vektordatentypen sind die arithmetischen Operatoren +, - und * sowie Vergleichsoperatoren in den Bibliotheken ieee.numeric_std und ieee.std_logic_arith definiert.
 - Damit stehen Integerzahlen und rein gebrochene Zahlendarstellungen, d.h. die Fractional-Zahlen, sowie deren Kombination für die 2er-Komplementarithmetik zur Verfügung.
 - Für Fractional-Zahlen im Q-Format gilt ein Bitstring mit der Form:
- $x_{\text{Bin}} = b_B \cdot b_{B-1} \dots b_1 b_0$ mit $b \in (0,1)$ und
einem Vorzeichenbit $b_B = 1$ für $x_{\text{Dez}} < 0$.
Dieses QB-Format repräsentiert Zahlen im Intervall $-1 \leq x_{\text{Dez}} < 1$

19

CE - DS 2



Q-Format

- Bewährt hat sich die **Fractional-Darstellung** im Q-Format für die Eingangs- und Ausgangssignale, Koeffizienten sowie Rechengrößen in digitalen Systemen, da Multiplikationsergebnisse im Bereich $-1 \leq x_{\text{Dez}} < 1$ begrenzt bleiben und betragsmäßig in Richtung des LSBs 2^{-B} für $b_0 = 1$ streben.
- Die QB-Notation angewandt auf Vektordatentypen sagt aus, dass die Binärdarstellung B Bits rechts des impliziten Binärpunktes enthält.
- Dies ist hier in einer beispielhaften Schreibweise dargestellt:

QB: sign bit + B significant bits = sign + msb ... lsb
z.B. für das Signal $x[\text{sign}, B-1 : 0]$

- Ein $B+1=5$ -Bit Koeffizient im Q4-Format hat z.B. folgende Deklaration:
`constant C0: signed(4 downto 0) := 01011; -- 0.6875 = 11/16`

20

CE - DS 2



Addition mit vorzeichenrichtiger Erweiterung der Operanden

- Eine Addition zweier QB-Größen A und C, die jeweils den positiven Maximalwert $1 \cdot 2^{-B}$ annehmen können, liefert maximal den Wert $2 \cdot 2^{-B+1}$, der aufgrund des Integeranteils nicht im QB-Format allein darstellbar ist.
- Der Ergebnisvektor SUM ist also um eine Bitstelle (Guard-Bit) breiter zu wählen, die den Integeranteil (Übertrag) aufnimmt.
- Da Signal- und Variablenzuweisungen auf beiden Seiten der Zuweisung den gleichen Datentyp und die gleiche Vektorbreite aufweisen müssen, sind die Summanden A und C zusätzlich mit einer vorzeichenrichtigen Erweiterung an die Breite des Summenvektors SUM anzupassen.

```
signal A, C: signed(7 downto 0); -- inputs
signal SUM: signed(8 downto 0); -- result
begin
  SUM <= (A(A'left) & A) + (C(C'left) & C) after 5 ns;
```

21

CE - DS 2



Binäre Multiplikation (1)

- Auch bei der Multiplikation ist die Vektorbreite des Ergebnisses auf die der Faktoren anzupassen.
- Eine Multiplikation mit einem QB_1 -Multiplikanden und einem QB_2 -Multiplikator ergibt ein QB_3 -Ergebnis.
Darin stehen $B_3 = B_2 + B_1$ Bits rechts vom impliziten Binärpunkt und der gesamte Vektor enthält $B_3 + 2$ Bits.
- Eine Q-Format-Multiplikation liefert nämlich **zwei Vorzeichenbits**, von denen das linke Bit ein „echtes“ Vorzeichen ist und das rechte Bit vor dem Binärpunkt als **Guard-Bit** genutzt werden kann.

22

CE - DS 2

Binäre Multiplikation (2)

```
constant COEFF: signed(7 downto 0) := x"7f";
                           -- 127/128 multiplier Q7
signal STAGE:   signed(11 downto 0); -- multiplicand Q11
signal MUL:     signed(19 downto 0); -- result Q18
begin
  MUL <= STAGE * COEFF after 6 ns;
```

Symbolische Schreibweise:

MUL[sign,sign. 17 : 0] = MUL[sign,guard. 17 : 0] =

STAGE[sign. 10 : 0] * COEF[sign. 6 : 0]

Binäre Multiplikation als Summe von Teilprodukten

Multiplikand * Multiplizierer

$$\begin{array}{r}
 \underline{-0.875 * (-0.6875)} \\
 04375 \\
 61250 \\
 065625 \\
 700000 \\
 0765625 \\
 5250000 \\
 0.6015625
 \end{array}
 \quad
 \begin{array}{r}
 \underline{1.0010} \\
 * \quad 1.0101
 \end{array}$$

Entwurf von Integer-Arithmetik mit dem Q-Format (1)

- Strategieübersicht
- 1. Ansatz: Auslegung des Datenpfades auf z.B. 10 Bit breite Signalpfade. Als Ausgangspunkt der Dimensionierung werden die Register als Quellen der Operanden mit 10 Bit Vektoren deklariert.
- Die für die Arithmetik-Operationen erforderlichen Modifikationen der Operanden und der Ergebnisse sind für die Vektorauslegung zu planen.
- Die Simulationsergebnisse des VHDL-Modells zeigen auf, ob die Vektorbreiten zur Genauigkeitserhöhung (kleineres LSB 2^{-B} , $B \uparrow$) anzupassen sind.
- Zyklus jeweils mit Wiederholung der Simulationsauswertung.

Entwurf von Integer-Arithmetik mit dem Q-Format (2)

- 2. Ansatz: Die Eingangssignale bzw. Operanden eines Matlab Floating-Point-Modells werden mit den Funktionen zur Fixed-Point-Arithmetik auf einen begrenzten Darstellungsbereich reduziert.
- Aus einer Ergebnisbewertung heraus wird auf die geeignete Vektordimensionierung des Q-Formats mit Nachkommastellen und Guard-Bits geschlossen.
- Ein Beispiel hierzu liefert die Vorabanalyse der Wurzel-Approximation mit dem Matlab-Code in der Laborbeschreibung CEP 3.

Entwurf von Integer-Arithmetik mit dem Q-Format (3)

- **3. Ansatz:** MDA (Modell getriebene Architektur) mit dem VHDL-Codegenerator **Systemgenerator**. RTL-Modellierung von PEs mit einer Matlab-Simulink basierten grafischen Oberfläche.
- Dimensionierung der Datenpfad-Funktionselemente mit Full-Precision (32 Bit Q-Format-Vektoren).
- Sukzessive Verfeinerung durch Vektorbreitenreduzierung und Überprüfung der Ergebnisgenauigkeit abgestimmt mit dem FPGA-Ressourcen-Bedarf.
- Ausgehend vom Simulationsendergebnis Übergang auf die VHDL-Simulation und die FPGA-Implementierung.
- Beispiele:
 D. Mellert: Modellierung einer Video-basierten Fahrspurerkennung mit dem Xilinx Systemgenerator für eine SOC-Plattform. BA HAW Hamburg, Department I; 04.2010
 D. Brandmeier: FPGA-Implementierung einer feldorientierten Regelung zur Leistungsfaktorkorrektur. Diplomarbeit HAW Hamburg, Department I/E 2006; ESW Wedel

Entwurf von Integer-Arithmetik mit dem Q-Format (4)

- **4. Ansatz:** Mathematische Analyse von linearen Systemen mit Rückkopplungen (IIR-Filter, Differentialgleichungen) durch komplexe Teilübertragungsfunktionen $G(j\omega)$ für innere Summationspunkte.
- Ausgangsskalierung und zusätzliche Dimensionierung des Q-Formats für Rückkopplungsaddierer mit Guard-Bit Erweiterung, die die durch Verstärkungsüberhöhungen verursachten Überschwinger überlauffrei aufnehmen.
- [5] Kapitel 9.2 IIR-Filter

Entwurf von Integer-Arithmetik (5)

5. Ansatz: Mobile-Kommunikation

Modellierung der Sprachcodierung in drei Schritten:

Floating-Point Realisierung

Reduzierung des C/C++ Codes für Festkomma-DSPs

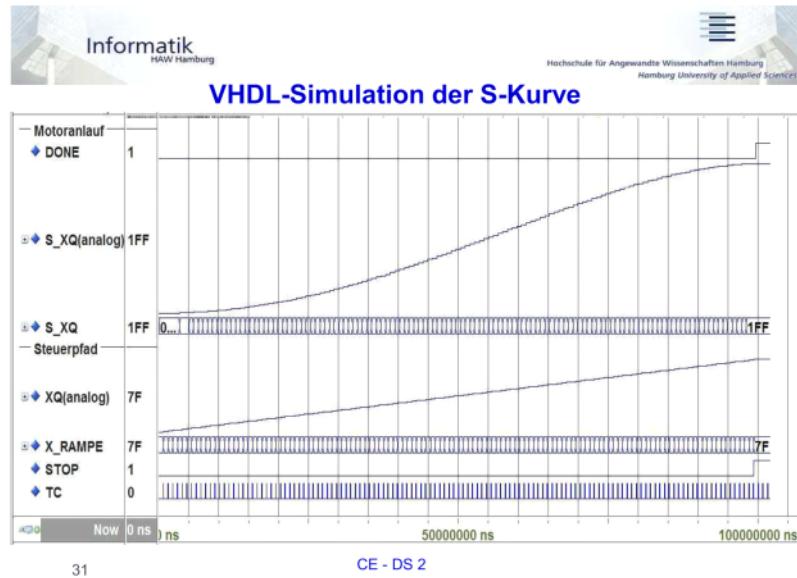
Akustischer Vergleich der Sprachqualität und Entscheidung für Fixed-Point Genauigkeit.

D. Bonkoungou: Bewertungskonzept und Berechnung von Festkomma-codebüchern für eine Festkommaimplementierung eines Sprachcodieralgorithmus. Diplomarbeit FH-Hamburg FB E/I 2002; Ericson Nürnberg

Auslegung des Q-Formats für die S-Kurve

➤ Verfahren nach Ansatz 1.:

- Rampenvektor x_Q : Erweiterung auf der LSB-Seite mit 2 Bit "00" und Kompensation des SHL durch Übergang von Q7 auf Q9.
- Abgriff des Multiplikationsergebnisses Q18 mit 2 VZ:
RESULT(18 : 9); abschneiden der unteren 9 Bit ohne Rundung.
- Erweiterung der Subtraktionsoperanden, sodass Faktor 2 für x^2 durch SHL erreicht wird und Sign-Extension bei x^4 , damit die Vektorlängen auf WIDTH+1 angeglichen werden.
- Hier Reduzierung des Subtraktionsergebnisses auf 10 Bit gezeigt, da $S(x_Q) < 1$ angenommen wird.
Mit einem 11 Bit Register R_3 für $S(x_Q)$ wäre mehr Sicherheit gegen einen Überlauf im Fall $S(x_Q) > 1$ gegeben.



31

CE - DS 2



```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
```

```
entity DATENPFAD is    -- Multizyklus-DP
generic ( WIDTH : natural := 13);    -- Vektorbreite im Datenpfad
port (CLK, RESET_N      : in bit;    -- LOW ACTIVE für XC3S400-Board 8. Stock
      SEL           : in bit;    -- Multiplexer Select S aus FSM
      R1_EN, R2_EN, R3_EN : in bit;    -- Registerfreigabe EN aus FSM
      X             : in bit_vector(7 downto 0);    -- Rampengenerator Xq
      F_X           : out std_logic_vector(9 downto 0) -- S-Kurve
      );
end DATENPFAD;
-----INTERNE SIGNALE-----
architecture VERHALTEN of DATENPFAD is
begin
  REG1, REG2, REG3 : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
  MUL            : std_logic_vector(WIDTH-1 downto 0);-- Arithmetic-Prozess
  SUB            : std_logic_vector(WIDTH-1 downto 0);
  X_CALC         : std_logic_vector(WIDTH-1 downto 0);-- shift left u. Q7 -> Q(Width-1)
  ZEROS          : std_logic_vector(WIDTH-1 -7 - 1 downto 0);-- XQ Ergänzung
```

32

CE - DS 2



Modell des Datenpfades (2)

```

begin
-----REGISTER-----
REGISTER1:process(CLK)
begin
  if (CLK = '1' and CLK'event) then
    if (RESET_N = '0') then
      REG1 <= (others => '0') after 5 ns;
    elsif (R1_EN = '1') then
      REG1 <= MUL after 5 ns; -- t1 = xx
    end if;
  end if;
end process REGISTER1;
REGISTER2:process(CLK)
begin
  if (CLK = '1' and CLK'event) then
    if (RESET_N = '0') then
      REG2 <= (others => '0') after 5 ns;
    elsif (R2_EN = '1') then
      REG2 <= MUL after 5 ns; -- t2 = xx*xx
    end if;
  end if;
end process REGISTER2;
REGISTER3:process(CLK)
begin
  if (CLK = '1' and CLK'event) then
    if (RESET_N = '0') then
      REG3 <= (others => '0') after 5 ns;
    elsif (R3_EN = '1') then
      REG3 <= SUB after 5 ns; -- t3 = y = 2xx - xxxx
    end if;
  end if;
end process REGISTER3;

```

CE - DS 2

33



Modell des Datenpfades (3)

```

ZEROS <= (others => '0');
X_CALC <= TO_stdlogicvector(X) & ZEROS; -- Rampe shift left u. Q7 -> Q(width -1)
-----ALU's-----
MULTIPLIER : process (SEL, X_CALC, REG1)
variable TMP : std_logic_vector(WIDTH-1 downto 0); -- selektierte Operanden
variable RESULT : std_logic_vector(2*WIDTH - 1 downto 0);-- doppelte Vektorbreite
begin
  if (SEL = '0') then
    TMP := X_CALC;
  else
    TMP := REG1;
  end if;
  RESULT := TMP * TMP; -- Q(width-1)*Q(width-1) = sign,sign. msb ... lsb; 2*width bit
  MUL <= RESULT ((2*WIDTH - 2) downto WIDTH-1); -- t1 ; t2 width bit: sign.msb ... lsb
end process MULTIPLIER;

SUBTRACTOR : process (REG1, REG2)
variable RESULT : std_logic_vector(WIDTH downto 0); -- Width +1 bit
begin
  -- shift left sign extension ; bleibt Q(WIDTH-1)
  RESULT := (REG1 & '0') - (REG2(REG2'left) & REG2);-- sign.guard.msb .. lsb
  SUB <= RESULT(WIDTH - 1 downto 0); -- y
end process SUBTRACTOR;
F_X <= REG3(WIDTH-1 downto WIDTH -10); -- S-Kurve y immer 10 Bit Ausgang
end VERHALTEN;

```

CE - DS 2

34



Modell des Steuerpfades (1)

```
-- FSM-Timer-Konzept
entity STEUERPFAD is -- FSM erweitert mit Rampenzähler und Zyklenzähler
port(
    PB: in bit;                                -- externer Startschalter
    RESET_N: in bit;
    CLK: in bit;
    DONE: out bit;                             -- Endwert erreicht
    R1_EN, R2_EN, R3_EN: out bit; -- EN Register Freigaben
    SEL: out bit;                            -- S MUX Select
    X: out bit_vector(7 downto 0) -- XQ Rampe
);
end STEUERPFAD;

architecture VERHALTEN of STEUERPFAD is
-- interne Signale -- Zyklenzähler- und Rampen-Power-Up
signal COUNTER: std_logic_vector(2 downto 0) := (others => '0');
signal X_RAMPE: std_logic_vector(7 downto 0) := (others => '0');

type STATES is (IDLE, S0, S1, S2);
signal ZUSTAND, FOLGE_Z: STATES := IDLE;
signal TC: bit;                         -- Zyklenzähler Terminal Count
signal STOP: bit;                        -- Rampenendwert erreicht
CE - DS 2
```

35



Modell des Steuerpfades (2)

```
ZYKLENSZAehler: process(CLK) -- Periodische wrap-arounds: Sägezahn
begin
    if CLK'event and CLK = '1' then
        if RESET_N = '0' then
            COUNTER <= (others => '0') after 5 ns;
        elsif PB = '1' then -- externe Freigabe wie bei FSM
            COUNTER <= COUNTER + 1 after 5 ns;
        end if;
    end if;
end process ZYKLENSZAehler;
TC <= '1' after 5 ns when COUNTER = 7 else '0' after 5 ns;-- Einzelpuls
RAMPENGENERATOR: process(CLK)
begin
    if CLK'event and CLK = '1' then
        if RESET_N = '0' then
            X_RAMPE <= (others => '0') after 5 ns;
        elsif TC = '1' and STOP = '0' then -- Selbsthaltung des Rampenendwertes
            X_RAMPE <= X_RAMPE + 1 after 5 ns;
        end if;
    end if;
end process RAMPENGENERATOR;
STOP <= '1' after 5 ns when X_RAMPE = 127 else '0' after 5 ns;-- 1 = Dauerpegel
Z_SPEICHER: process(CLK)
begin
    if (CLK = '1' and CLK'event) then
        if RESET_N = '0' then
            ZUSTAND <= IDLE after 5 ns;
        elsif PB = '1' then -- externer Start (CE)
            ZUSTAND <= FOLGE_Z after 5 ns;
        end if;
    end if;
end process Z_SPEICHER;
CE - DS 2
```

36



Modell des Steuerpfades (3)

```

UE_A_SN: process(ZUSTAND, TC, STOP)
begin
    SEL <= '0' after 5 ns; -- Defaults
    R1_EN <= '0' after 5 ns;-- Rampe an MUL
    R2_EN <= '0' after 5 ns;
    R3_EN <= '0' after 5 ns;
    FOLGE_Z <= IDLE after 5 ns; -- keine Transition
    DONE <= '0' after 5 ns; -- kein Endergebnis S(xq=127)
case ZUSTAND is
when IDLE =>
    DONE <= STOP after 5 ns; -- Anzeige des S(xq=127) Endwertes
    if TC='1' and STOP = '0' then -- 1. Rechnung mit erster Rampenstufe
        FOLGE_Z <= S0 after 5 ns; -- Ende mit Rampenstop
    end if;
when S0 =>
    R1_EN<='1' after 5 ns; -- t1 = xx speichern
    FOLGE_Z <= S1 after 5 ns;
when S1 =>
    SEL <= '1' after 5 ns; -- R1 an MUL
    R2_EN <= '1' after 5 ns; -- t2 = xx*xx speichern
    FOLGE_Z <= S2 after 5 ns;
when S2 =>
    R3_EN <= '1' after 5 ns; -- Subtraktion speichern
end case;
end process UE_A_SN;
X <= To_BitVector(X_RAMPE) after 5 ns;
end VERHALTEN;

```

37



FSM-Timer Konzept

- Dieses Konzept zur Kopplung eines Datenpfades mit dem Steuerpfad ist aufgrund folgender Aspekte zu empfehlen:
- Sofern in Anwendungen ein Timer-Grenzwert in mehreren Zuständen geprüft werden muss, ist es sinnvoll, den Komparator nicht mehrfach in der FSM zu realisieren. Der in der Timer-Entity platzierte Komparator spart damit Logik-Ressourcen ein.
- Zähler, die große Speicherbereiche adressieren, wie z.B. Bildzwischen-speicher mit >1 MB in Farbbildverarbeitungsanwendungen, sind ggf. mehr als 20 Bit breit. Ein Austausch von einzelnen Statusleitungen mit der FSM anstelle der Einkopplung aller Zählerleitungen in das kombinierte Übergangs- und Ausgangsschaltnetz kann die Verdrahtungsressourcen reduzieren. Da das FPGA-Routing in den Signallaufzeitpfaden je nach Anwendung bis zu 50 % ausmacht, kann diese vorgeschlagene Funktionspartitionierung auch die Timingkennwerte verbessern.

38

CE - DS 2



Top-Entity (1)

```
-- CE DS 2 im SS 10 SWR
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;

entity MOTORANLAUF is -- S-Kurven-Approximation Top-Entity
port(
    PB      : in bit;
    RESET_N : in bit;
    CLK     : in bit;
    DONE    : out bit; -- Maximalwert S(XQ=1) = 1 erreicht
    S_XQ   : out std_logic_vector(9 downto 0)-- S-Kurve
);
end MOTORANLAUF;
architecture VERHALTEN of MOTORANLAUF is

signal R1_CE, R2_CE, R3_CE, S: bit;-- portmap Kopplung
signal XQ: bit_vector(7 downto 0); -- Rampe

component STEUERPFAD  -- FSM erweitert mit Rampenzähler und Zyklenzähler
port(
    PB      : in bit; -- Externer Start
    RESET_N : in bit;
    CLK     : in bit;
    DONE    : out bit; -- Endwert erreicht
    R1_EN, R2_EN, R3_EN: out bit; -- Register-Freigabe
    SEL    : out bit;           -- MUX Select für MUL Operanden
    X      : out bit_vector(7 downto 0)-- Rampe
);
end component;
-----
```

CE - DS 2

39



Top-Entity (2)

```
component DATENPFAD  -- Multizyklus DP
port (CLK, RESET_N : in bit;          -- RESET LOW ACTIVE
      SEL       : in bit;           -- MUX Select für MUL Operanden
      R1_EN, R2_EN, R3_EN : in bit;-- Register-Freigabe
      X         : in bit_vector(7 downto 0); -- Rampe
      F_X       : out std_logic_vector(9 downto 0) -- S-Kurve
);
end component;
-----
```

begin

```
ST_I: STEUERPFAD  -- Instanziierung
port map(PB => PB, RESET_N => RESET_N, CLK => CLK, DONE => DONE,
         R1_EN => R1_CE, R2_EN => R2_CE, R3_EN => R3_CE,
         SEL => S, X => XQ);-- formal => actual
```

```
DP_I: DATENPFAD
port map(
    CLK => CLK, RESET_N => RESET_N, SEL => S,
    R1_EN => R1_CE, R2_EN => R2_CE, R3_EN => R3_CE,
    X => XQ, F_X => S_XQ);
end VERHALTEN;
```

CE - DS 2

40

Informatik
HAW Hamburg

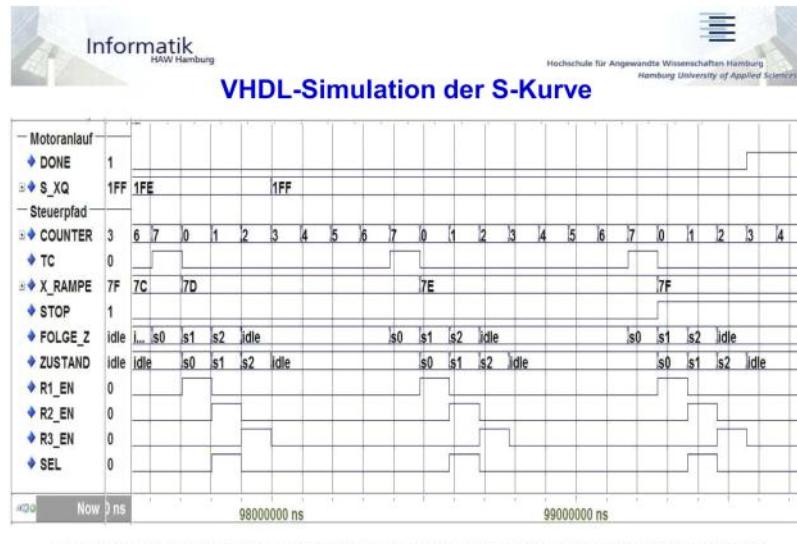
Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Testbench (1)

```

entity TB is
generic (PERIODE10_24KHZ_HALBE: time := 48828125 ps);
-- Frequenz fuer Anlaufdauer 100 ms
-- 100 ms / 8 / 128 = Periodendauer -> f = 1/T = 10,24 kHz
end TB;
architecture VERHALTEN of TB is
signal PB, CLK, RESET_N : bit;
component MOTORANLAUF
port( PB, RESET_N, CLK : in bit;
      DONE : out bit;
      S_XQ : out std_logic_vector(9 downto 0) );
end component;
begin
DUT: MOTORANLAUF
port map(PB -> PB, CLK -> CLK, RESET_N -> RESET_N, S_XQ -> open, DONE -> open);
TESTBENCH_CLK: process
begin
  CLK <= '0'; wait for PERIODE10_24KHZ_HALBE;
  CLK <= '1'; wait for PERIODE10_24KHZ_HALBE;
end process TESTBENCH_CLK;
TESTBENCH_RESET: process
begin
  RESET_N <= '0'; wait for (2*PERIODE10_24KHZ_HALBE);
  RESET_N <= '1'; wait;
end process TESTBENCH_RESET;
TESTBENCH_PB: process
begin
  PB <= '0'; wait for (2*PERIODE10_24KHZ_HALBE);
  PB <= '1'; wait;
end process TESTBENCH_PB;
end VERHALTEN;

```



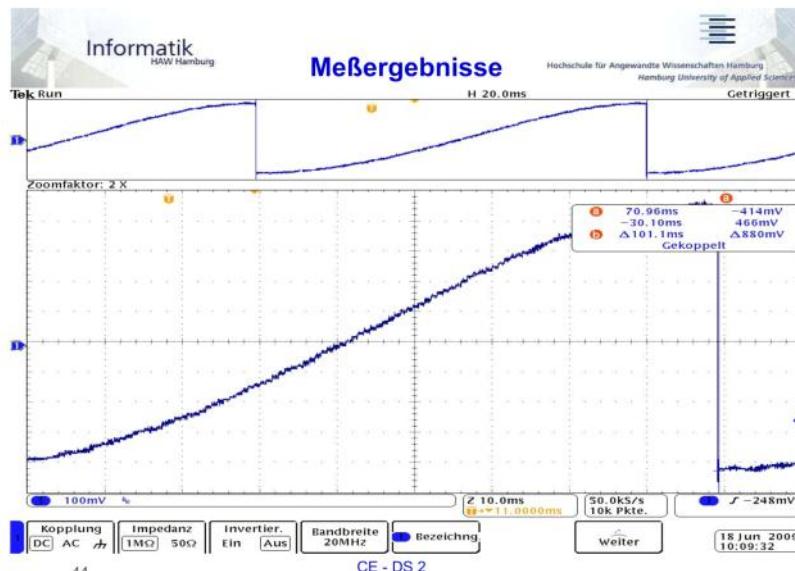


Auszug aus .do File für die Signalauswahl

```
# Fuer S-Kurvenapproximation
restart
view wave
radix hex
# Displays all signals in waves window:
#add wave -noupdate -divider PushButton
#add wave -height 30 -label PB sim:/tb/dut/pb
#add wave -noupdate -divider "Clock & Reset(LowActive)"
#add wave -height 30 -label CLK sim:/tb/dut/clk
#add wave -noupdate -divider "Motoranlauf"
add wave -noupdate -label DONE sim:/tb/dut/done
add wave -height 200 -analog-step -min 0 -max 512 -label "S_XQ(analog)" sim:/tb/dut/s_xq
add wave -height 30 -label COUNTER sim:/tb/dut/st_i/counter
add wave -noupdate -divider "Datenpfad"
add wave -height 30 -label REG3 sim:/tb/dut/dp_i/reg3
add wave -height 30 -label MUL sim:/tb/dut/dp_i/mul
# geforderte Anlaufzeit
run 102 ms
```

43

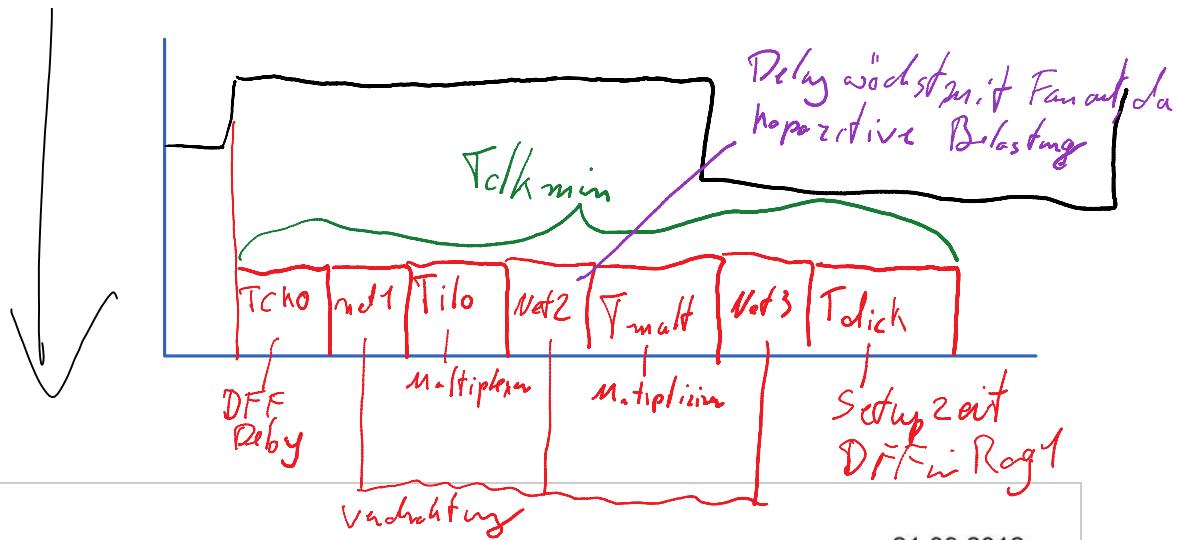
CE - DS 2



44

Timing analyse mit Post Phase & Route Timing analyse

Delay wächst mit Fanout



WS 12

21.09.2012



Längster Laufzeitpfad (1)

Delay (setup path): 10.496 ns (data path - clock path skew + uncertainty)				
Source:	ST_I/X_RAMPE_7 (FF)	Destination:	DP_I/REG1_5 (FF)	
Data Path Delay:	10.496ns (Levels of Logic = 2)	Clock Path Skew:	0.000ns	Source Clock: CLK_BUFGP rising
Destination Clock:	CLK_BUFGP rising	Clock Uncertainty:	0.000ns	
Maximum Data Path:	ST_I/X_RAMPE_7 to DP_I/REG1_5			
Location	Delay type	Delay(ns)	Physical Resource	Logical Resource(s)
SLICE_X13Y107.YQ	Tck0	0.511	ST_I/X_RAMPE->	
SLICE_X13Y96.G4	net (fanout=3)	0.772	ST_I/X_RAMPE_7	
SLICE_X13Y96.Y	Tl0	0.612	ST_I/X_RAMPE->7>	
MULT18X18_X0Y12.B15	net (fanout=12)	2.499	DP_UTMP_mux0000<0>	
MULT18X18_X0Y12.P17	Tmult	4.331	DP_UTMP_mux0000<12>	
SLICE_X2Y97.BX	net (fanout=2)	1.458	DP_IMult_RESULT_mult0000	
SLICE_X2Y97.CLK	Tdick	0.313	DP_IMUL<5>	
			DP_I/REG1_5	
Total		10.496ns (5.767ns logic, 4.729ns route) (54.9% logic, 45.1% route)		

CE - DS 2

45



Spartan 3E Switching Characteristics (1)

Configurable Logic Block (CLB) Timing

Table 98: CLB (SLICEM) Timing

Symbol	Description	Speed Grade				Units
		-5	-4	Min	Max	
Clock-to-Output Times						
T_{CK0}	When reading from the FFX (FFY) Flip-Flop, the time from the active transition at the CLK input to data appearing at the XQ (YQ) output	-	0.52	-	0.60	ns
Setup Times						
T _{AS}	Time from the setup of data at the F or G input to the active transition at the CLK input of the CLB	0.46	-	0.52	-	ns
T _{DICK}	Time from the setup of data at the BX or BY input to the active transition at the CLK input of the CLB	1.58	-	1.81	-	ns
Hold Times						
T _{AH}	Time from the active transition at the CLK input to the point where data is last held at the F or G input	0	-	0	-	ns
T_{CKDI}	Time from the active transition at the CLK input to the point where data is last held at the BX or BY input	0	-	0	-	ns

AC

CE - DS 2

T_{CKD}	G input				
	Time from the active transition at the CLK input to the point where data is last held at the BX or BY input.	0	-	0	-
46		CE - DS 2			



Spartan 3E Switching Characteristics (2)

Propagation Times						
T _{IO}	The time it takes for data to travel from the CLB's F (G) Input to the X (Y) output	-	0.66	-	0.76	ns

18 x 18 Embedded Multiplier Timing

Table 102: 18 x 18 Embedded Multiplier Timing

Symbol	Description	Speed Grade				Units
		-5	-4	Min	Max	
Combinatorial Delay						
T _{MULT}	Combinatorial multiplier propagation delay from the A and B inputs to the P outputs, assuming 18-bit inputs and a 36-bit product (AREG, BREG, and PREG registers unused)	-	4.34(1)	-	4.88(1)	ns

47

CE - DS 2



Längster Laufzeitpfad (2)

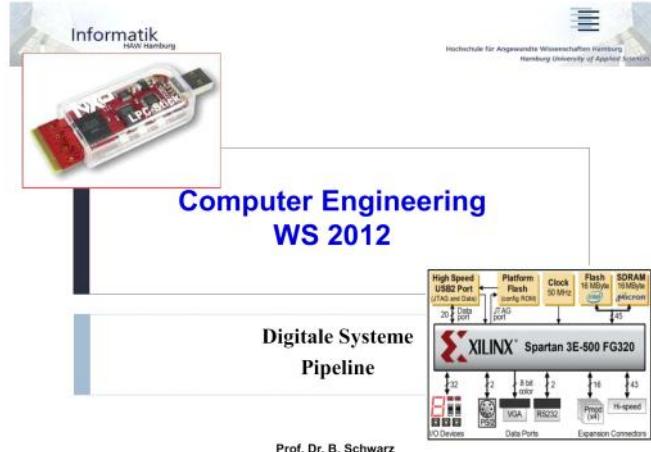
Delay (setup path):	10.332ns (data path - clock path skew+uncertainty)		
Source:	ST_I/ZUSTAND_FSM_FFd2_1(FF)		
Destination:	DP_I/REG1_5 (FF)		
Data Path Delay:	10.332ns (Levels of Logic = 2)		
Clock Path Skew:	0.000ns		
Source Clock:	CLK_BUFGP rising		
Destination Clock:	CLK_BUFGP rising		
Clock Uncertainty:	0.000ns		
Maximum Data Path:	ST_I/ZUSTAND_FSM_FFd2_1 to DP_I/REG1_5		
Location	Delay type	Delay(ns)	Physical Resource
			Logical Resource(s)
SLICE_X13Y103.YQ	T _{cko}	0.511	ST_I/ZUSTAND_FSM_FFd2_1
SLICE_X13Y96.G3	net (fanout=13)	0.608	ST_I/ZUSTAND_FSM_FFd2_1
SLICE_X13Y96.Y	T _{ilo}	0.612	ST_I/ZUSTAND_FSM_FFd2_1
MULT18X18_X0Y12.A17	net (fanout=12)	2.499	DP_I/TMP_mux000<0>
MULT18X18_X0Y12.P17	T _{mult}	4.331	DP_I/TMP_mux000<12>
SLICE_X2Y97.BX	net (fanout=2)	1.458	DP_I/Mmult_RESULT_mult0000
SLICE_X2Y97.CLK	T _{dclk}	0.313	DP_I/Mmult_RESULT_mult0000
			DP_I/MUL<5>
			DP_I/REG1<5>
			DP_I/REG1_5
Total	10.332ns (5.767ns logic, 4.565ns route) (55.8% logic, 44.2% route)		

48

CE - DS 2

WS 12

21.09.2012

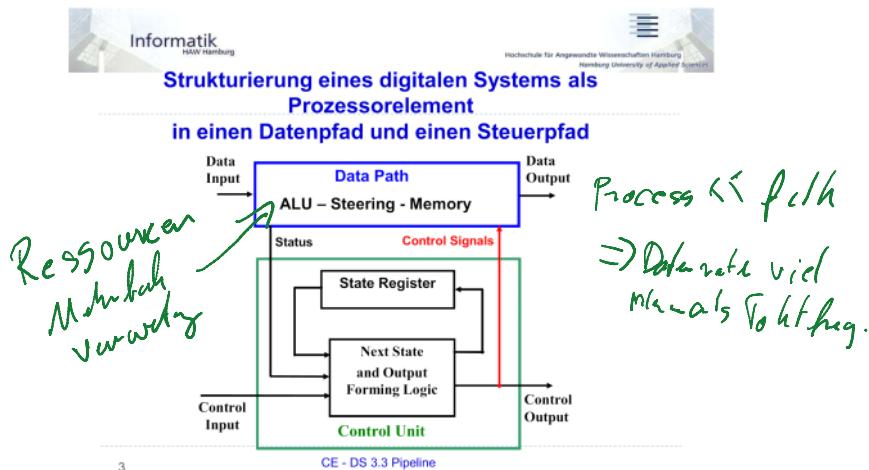


3. Architekturentwurf ASM-Diagramme

Prozessorelement für eine S-Kurvenapproximation
Gemeinsame Nutzung von Hardware Funktionseinheiten
(Ressource Sharing)
Gemeinsame Register- / Speicher-Nutzung
(Register Sharing)
VHDL-Modell des PE mit Fixed-Point Arithmetik im Q-Format

Datenpfad in Pipelinestruktur

Dynamische Systeme mit Rückführung
Multizyklusdatenpfad für ein Filter 1. Ordnung
Ergebnisübergabe an Folgezyklen



Informatik
HAW Hamburg

Datenrate ≈ Rechtfreq
wegen Ressourcen Mehrfachverwendung
Pipelining

- Pipelining ist das **Aufbrechen langer Signalpfade** in etwa gleich lange Schaltnetzstufen.
- Eine Separierung der **N Logik-Teilabschnitte** im Daten- (und ggf. im Steuerpfad) erfolgt mit **Pipeline-Registern**.
- Jeder einzelne Schaltnetzabschnitt wird takt synchron gespeichert. Dadurch werden **N unterschiedliche Eingangsdaten (ein Datensatz)** **parallel** in aufeinander folgenden Stufen verarbeitet
- Wenn die Länge der verkürzten N Signalpfadabschnitte etwa gleich groß ist und die **Laufzeit** jedes einzelnen T_{CLK}/N der ursprünglichen Takelperiode T_{CLK} entspricht, so ist die Bearbeitungsdauer (Latenz) für ein Eingangsdatum nahezu genauso lang wie ohne Pipelining.
- Allerdings ist der **Datendurchsatz mit Pipelining größer**, da pro Intervall T_{CLK}/N ein aktualisierter Ausgang verfügbar wird!
- Bis ein Ergebnis am Ausgang liegt vergeht eine **Latenz von N-Takten ($N \cdot T_{CLK}/N$)**.

4

CE - DS 3.3 Pipeline

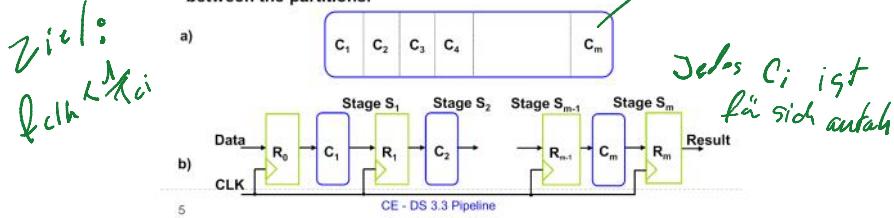
 $N \geq m$

Informatik
HAW Hamburg

Pipelining of Data Paths

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

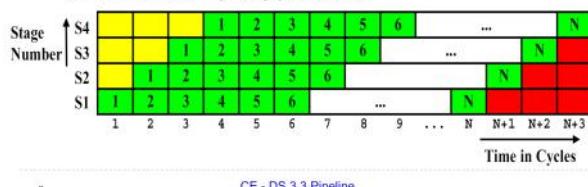
- Pipelining increases performance by restructuring long data paths with several **levels of logic** (C_i) and breaking it up over multiple clock cycles. Pipelining adds registers in the **combinational logic**.
- Shorter clock cycle supports an increased data throughput at the expense of added data latency.
- A computational circuit has to be partitioned into several approximately equal delay parts (C_i) and then inserting registers in between the partitions.



Letzter wic $m \cdot T_{C_i}$! Erhöhter Durchsatz von 1 Ergebnis pro Schritt

Phasendiagramm (1)

- For visualizing the operation of a pipeline a **space-time diagram** is used. The horizontal axis corresponds to time and the vertical axis corresponds to the stage number. The entries in the green boxes correspond to the symbolic IDs of the data currently worked on.
- The shown space-time diagram describes an example of a $m=4$ stage pipeline, with **each stage clocked four times** the frequency of the system without pipelining. After $N+3$ rising clock edges N data items are completely processed.





Phasendiagramm (2)

- Looking at one time instant, i.e. time unit 4, we can see that all stages of the pipeline are busy working on different data items. A data item is completely processed when it leaves the last stage. The first data item leaves the pipeline when all stages are filled. After that has accomplished **each single cycle yields a new data item result**.
- Disregarding the first $m-1=3$ cycles the pipeline works with a speedup which is determined by the number m of the stages: four times.
- In general, if there are **m stages** in the pipeline, the time taken to generate results can be **reduced to $1/m$ times** of the non-pipelined execution time, with the exception of the first result.

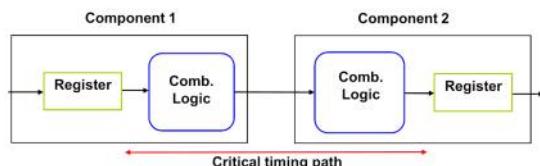
7

CE - DS 3.3 Pipeline



Control Pipelining (1)

- A critical timing path may arise in control and status signal lines between data path and control unit.
- The figure depicts the direct interfacing of an **FSM output forming logic** to an **input decoding logic of a data path**. Coupled combinational circuits of connected components have to be broken up with inserted registers in order to shorten data paths.



8

CE - DS 3.3 Pipeline

Control Pipelining (2)

- Each design cycle has to be finished with a post layout timing simulation which will yield an analysis of frequency limiting timing paths.
- After each step of inserting pipelining registers into data path and/or component connections special considerations must be applied to the number of states in control FSM and their transitions. The overall timing has to fit to the added path latency.

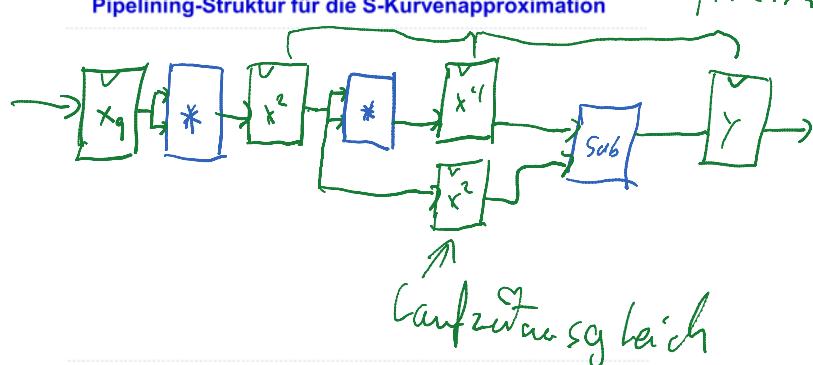


9

CE - DS 3.3 Pipeline

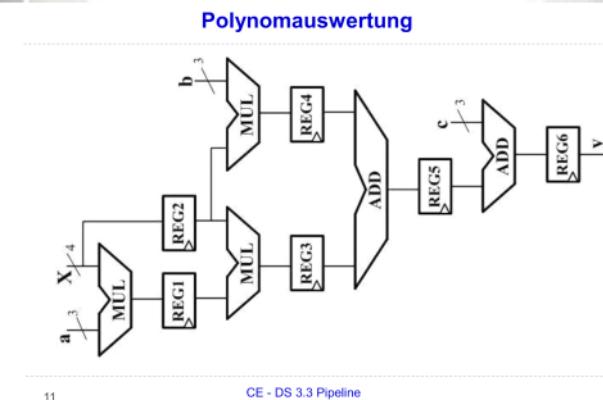
Pipelining-Struktur für die S-Kurvenapproximation

Pipeline register



10

CE - DS 3.3 Pipeline



11

1 Mal Mult * Add

⇒ $a, b \in \mathbb{K}$

Koeffizienten
an gleichzeitig

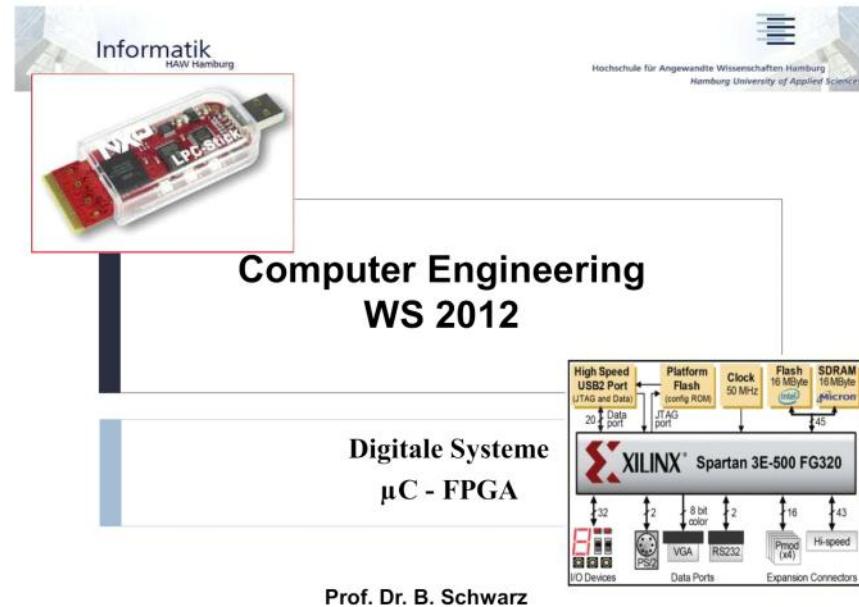
Polynomauswertung mit zeitveränderlichen Parametern

12

CE - DS 3.3 Pipeline

CE_SWR_F4

Dienstag, 4. Dezember 2012
09:48



Einleitung: μC – FPGA Kommunikation

Asynchrone Eingangssignale

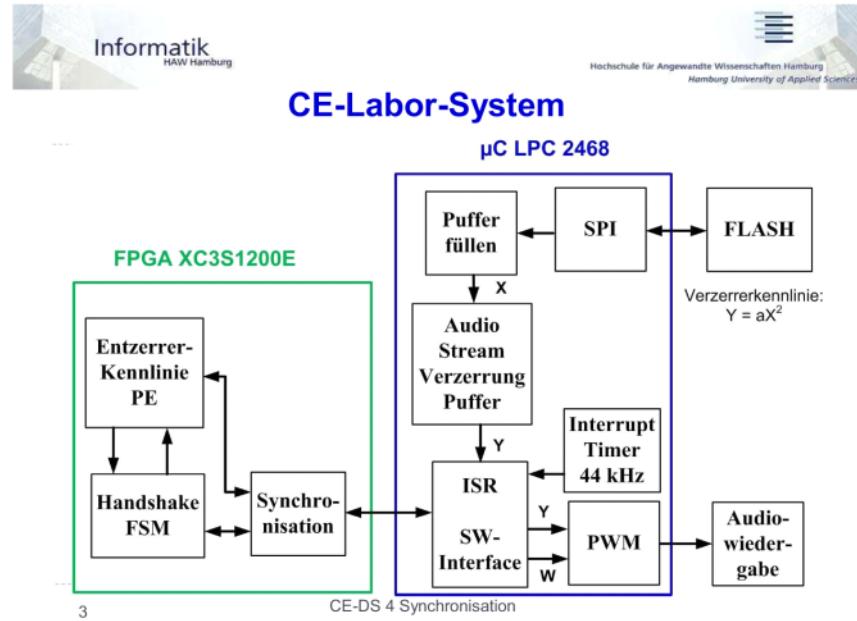
MTBF bei Flipflops mit metastabilen Zuständen

Synchronisationsschaltung für lange Impulse

Synchronisationsschaltung für kurze Impulse

Kommunikation zwischen asynchronen Clock-Bereichen

Vier-Phasen Handshake



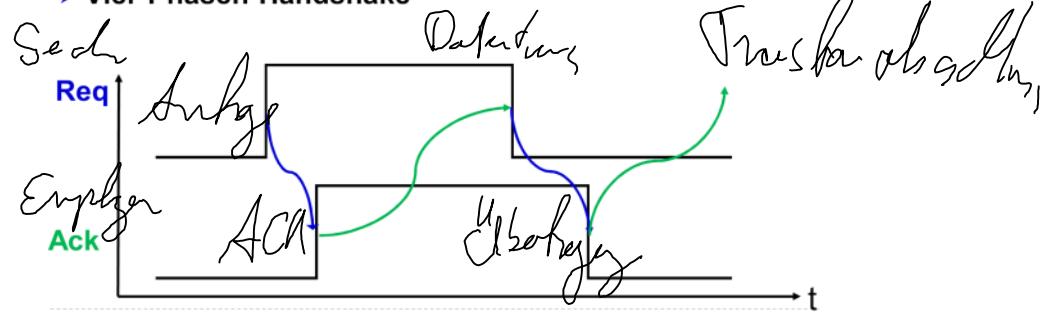
Asynchrone Kommunikation

Ereignisgesteuerte Effekte an den FPGA-Eingängen
erfordern Ansatz ohne Abhängigkeit von einem gemeinsamen oder übertragenen Clock-Signal.

- **Sender-Empfänger-Kommunikation** mit einem Protokoll, das Signalisierungskonventionen nutzt.
- Jede Komponente arbeitet mit der **eigenen Taktrate**.
- Nur für Interaktionen findet eine Kommunikation mit **synchronisierten Abläufen** statt.

Handshake-Kommunikation

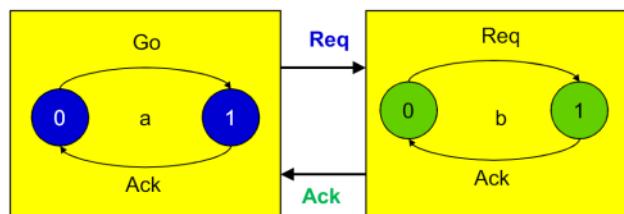
- Locally clocked – globally delay-insensitive
- Requester/Client/Master – Provider/Server/Slave
- Vier-Phasen-Handshake





Wechselseitige Abstimmung

- Beide Seiten betreiben die Abstimmung mit einem Automaten.
- Zustandstransitionen zeigen den Fortschritt der abgestimmten Kommunikation an.

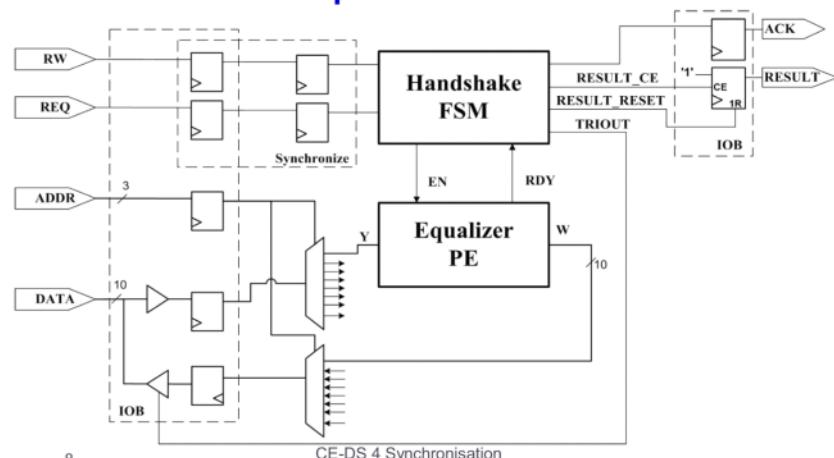


7

CE-DS 4 Synchronisation



Asynchrone Kommunikation μC - FPGA



8

CE-DS 4 Synchronisation



FPGA-Schnittstellen

- Eingangssignale werden durch **D-FFs** auf den FPGA-Takt **synchronisiert**:
Pegel ändern sich gleichzeitig.
Weniger, kürzere Hazards in der Eingangslogik.
- Ausgangsregister liefern eine parallele Pegelaktualisierung zu den GPIOs des µC.
- D-FFs in den **Input-Output-Blocks (IOBs)** stehen für beide Richtungen zur Verfügung.
- Handshake-Signale (**REQ, RW**), die die getaktete Zustandssequenz in der FSM beeinflussen, sind einer speziellen Synchronisation zu unterziehen.

*Synthese - Implementierung
Prozessier*

9

CE-DS 4 Synchronisation



Asynchrone Eingangssignale

- T_S, Setup time*
- Asynchron sind alle Eingangssignale eines synchronen RTL-Entwurfs, die nicht mit dem Systemtakt synchronisiert sind, deren Pegeländerung also irgendwann während des Taktzyklus erfolgen kann, also auch während des Entscheidungsintervalls
- Holdtime*
- $t_E = t_S + t_h$ der abtastenden Clock.

Dazu gehören z.B.:

- Anforderungen externer Geräte: Tastatureingaben, Interrupts, serielle Schnittstellen, ...
- Der Datenaustausch zwischen zwei jeweils synchronen Systemen, die jedoch mit unterschiedlicher Frequenz betrieben werden: Rechnerkopplungen, USB-Interfaces zum FPGA-Prozessor.

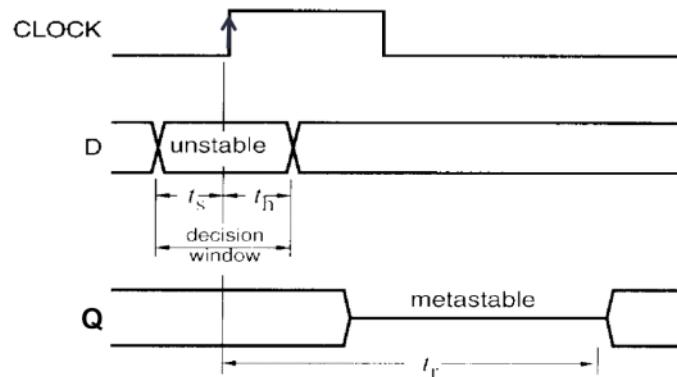
10

CE-DS 4 Synchronisation

- f_2 : Rechteck Intervall; Auflösungszeit
 \Rightarrow ist das Maximum des Zeitintervalls, in dem das Signal Q metastabil simuliert, ohne dass ein Synchronisationsfehler auftritt.
- Ein Synchronisationsfehler tritt auf, wenn ein System einen Sync-Ausgang Q nutzt, solange dieser noch metastabil ist

→ Konsequenz: Aufwände müssen ein "große" Fenster für f_2 zur Verfügung stellen!

Metastabiler Zustand des D-FF Ausgangs



11

CE-DS 4 Synchronisation

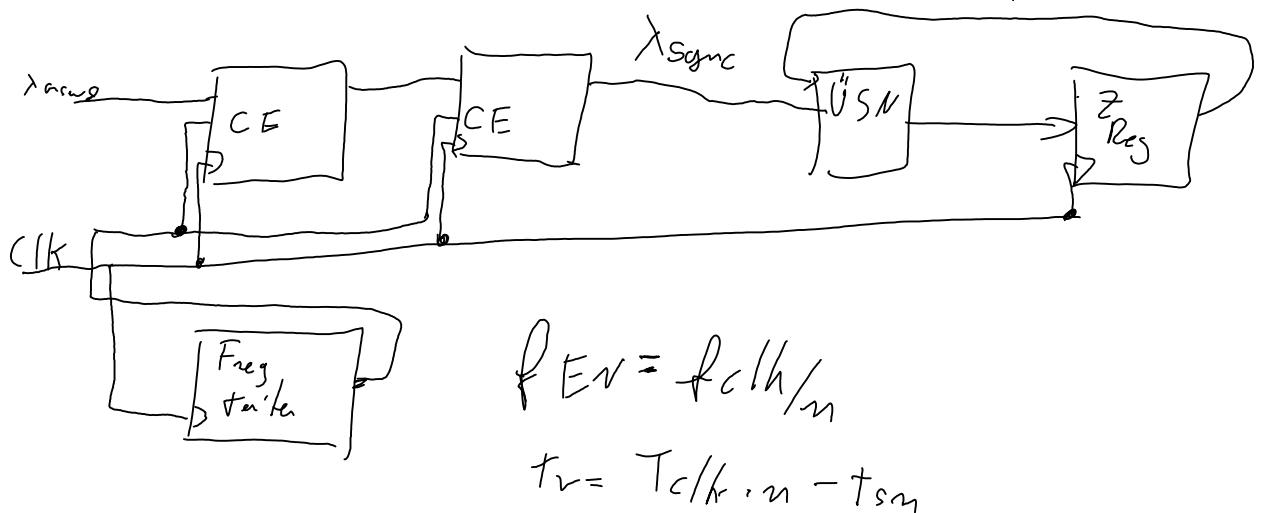
Metastabiler Zustand

- Der metastabile Zustand liegt im Bereich zwischen dem High- und Low-Pegel bei „midsupply“.
- Es ist nicht vorhersehbar, welcher Logikpegel sich im Anschluss an die Auflösungszeit t_r (resolution time) am D-FF Ausgang einstellt.
- Alle asynchronen Eingangssignale, die auf Flipflop-, Zähler- FSM- oder Schieberegister-Eingänge geführt werden, sind speziell zu synchronisieren.
- Dies soll sicherstellen, dass sich deren Eingangssignale **nicht während des Entscheidungsintervalls $t_E = t_s + t_h$ verändert**. Andernfalls können die Register- bzw. Zählerausgänge in den metastabilen Zustand gehen, der dann über das gesamte System verteilt wird.

12

CE-DS 4 Synchronisation

Vorläufig bars für Vergrößerung durch einzige Taktfrequenz



t_c sei leicht verletzt, dann -asym synchronisiert
jedoch zeit einhalten kann

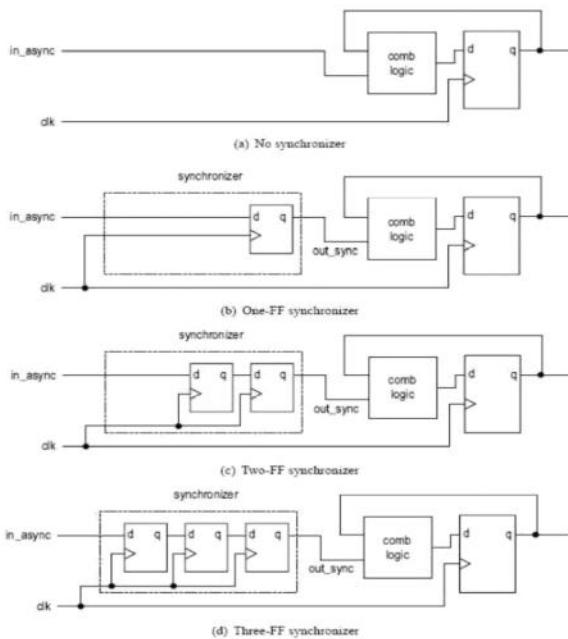


Synchronisierer

Unterschiedliche
lange Zeitfenster
für die
Auflösungszeit t_r
verfügbar.

aus P. P. Chu:
RTL Hardware Design Using VHDL, Wiley 2006

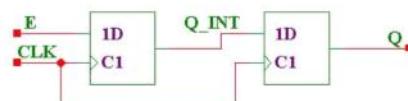
13



Synchronisationsschaltung für lange Impulse

- An einem vorgeschalteten Synchronisations-FF selbst können die Setup- und Hold-Zeiten verletzt werden.
- Sofern das 1. Synchronisations-D-FF in den metastabilen Zustand übergegangen ist, kann dieser Zustand sich jedoch im Laufe der Taktperiode auflösen ($t_r < T_{CLK}$) und damit das 2. D-FF einen korrekten Logikpegel $Q_{_INT}$ übernehmen.
- Damit wäre die Wahrscheinlichkeit reduziert, dass angeschlossene FSMs und Datenpfade mit undefinierten Pegeln angesteuert werden.

Synchronisations Flipflop

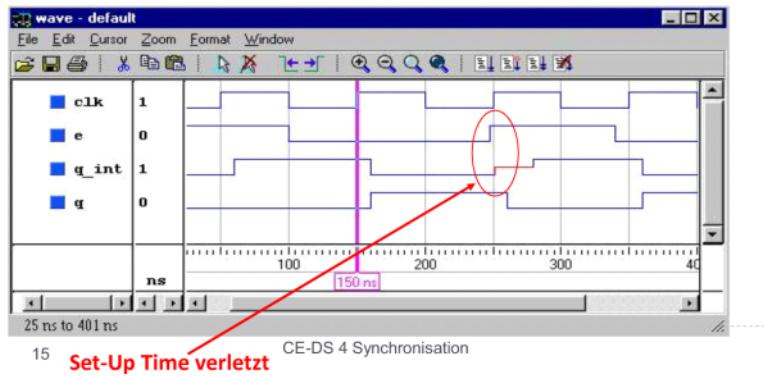


14



Auflösungszeit

- Das Signal E wird durch die Synchronisation um bis zu zwei Takte verzögert!
- Die Wahrscheinlichkeit für das Auftreten eines Fehlers wird durch eine mittlere Zeit zwischen auftretenden Fehlern **MTBF** (**mean time between failure**) beschrieben.



Abschätzung der Fehlerhäufigkeit von D-FFs

- Synchronisationsfehler treten auf, wenn der metastabile Zustand länger andauert, als die vom Hersteller angegebene Auflösungszeit **tr**.
- Die mittlere Zeit **MTBF** zwischen zwei Synchronisationsfehlern ist abhängig von:
 - **f**: Taktfrequenz mit der die Flipflops betrieben werden.
 - **a**: Frequenz mit der sich das asynchrone Signal ändert.
 - **T0**: D-FF Konstante gibt das Zeitfenster an, in dem Pegelwechsel einen Fehler erzeugen.
 - **t**: D-FF Konstante des Übergangsverhaltens von metastabilen Zuständen

Tr füg gegeben Schaltungen
bestimmen



$$tr = \frac{1}{f \cdot t_h} - t_{SU}$$

MTBF

$$MTBF(tr) = \frac{\exp(tr/\tau)}{T_0 \cdot f \cdot a}$$

Familie	τ/ns	T_0/s	tr/ns	t_{SU}/ns
74LS74	1.5	$4.0 \cdot 10^{-1}$	77.71	20
74HCxx	1.82	$1.5 \cdot 10^{-6}$	71.55	25
XC95108-20	0.17	$9.6 \cdot 10^{-18}$	2.3	10

Vgl. J. F. Wakerly: Digital Design. Principles and Practices. Pearson 4th ed 2006



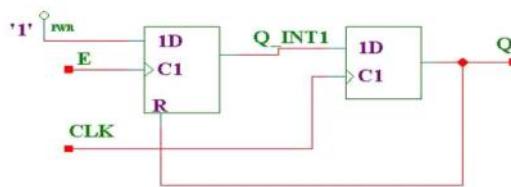
Beispiel: MTBF

- Das Interruptsignal eines Mikroprozessor, der mit $f = 10 \text{ MHz}$ Taktfrequenz arbeitet wird mit zwei 74LS74 D-FFs synchronisiert.
- Der asynchrone Interrupt tritt mit einer Rate von 10^5 1/s auf.
- Die verfügbare Zeit zum Abklingen des metastabilen Zustands ist $tr = 1/f - t_{SU} = 80\text{ns}$.
- Fehler treten auf mit $MTBF(80 \text{ ns}) = 3.6 \cdot 10^{11} \text{ s}$, sodass fehlerhafte Übergänge etwa alle 114 Jahrhunderte auftreten!
- Falls der Prozessor jedoch mit 16MHz getaktet werden soll, ergibt sich:
 $tr = 42.5 \text{ ns} \rightarrow MTBF(42.5 \text{ ns}) = 3.1 \text{ s. } \text{Nich takzeptabel!}$

- Berechnen Sie die MTBF für den Baustein XC95108-20 jeweils bei $f = 16\text{MHz}$ und $f = 50\text{MHz}$!

Synchronisationsschaltung für kurze Impulse

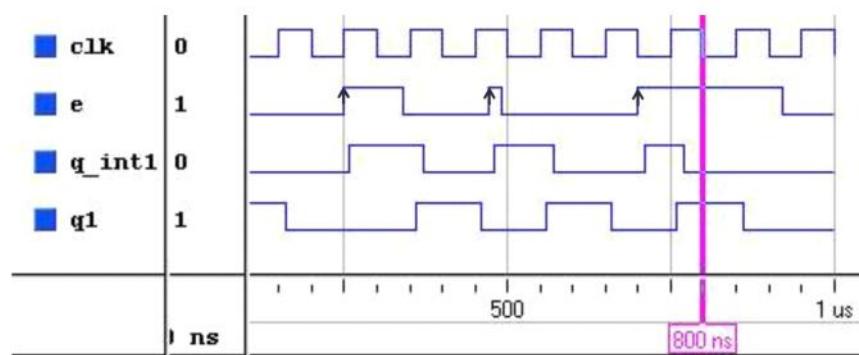
- Das asynchrone Eingangssignal E liegt am Takteingang des ersten Synchronisations-Flipflops.
- Das erste D-FF wird durch das zweite Flipflop asynchron zurückgesetzt.
- Unabhängig von der Impulsdauer des Signals E erzeugt die Schaltung immer einen Puls Q1 für die Länge eines Taktes (T_{CLK})!
- Der Ausgang Q1 ist nicht frei von metastabilen Zuständen, ggf. muss die Schaltung um ein weiteres Flipflop ergänzt werden.



19

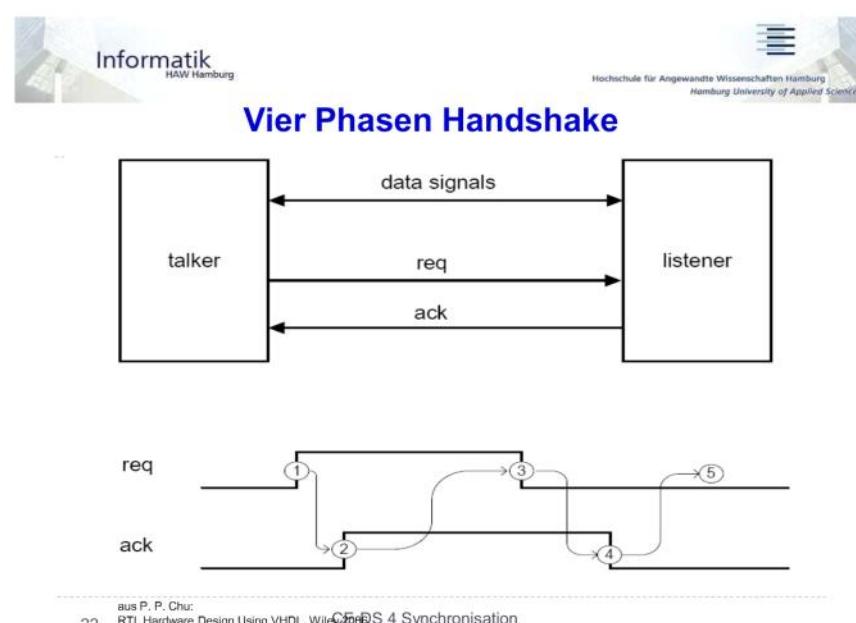
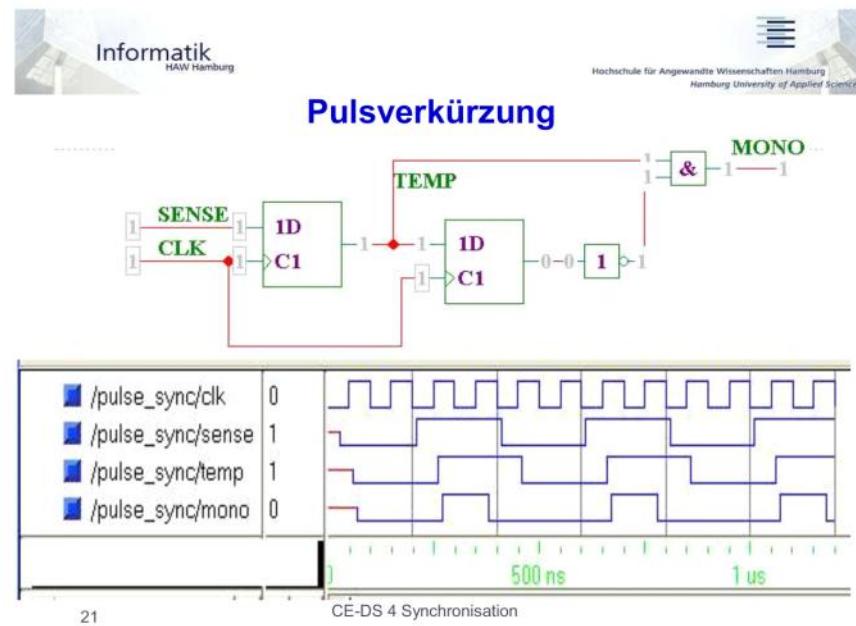
CE-DS 4 Synchronisation

Pulsverlängerung



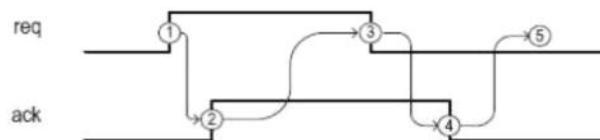
20

CE-DS 4 Synchronisation





Aufforderungs- und Bestätigungssequenz



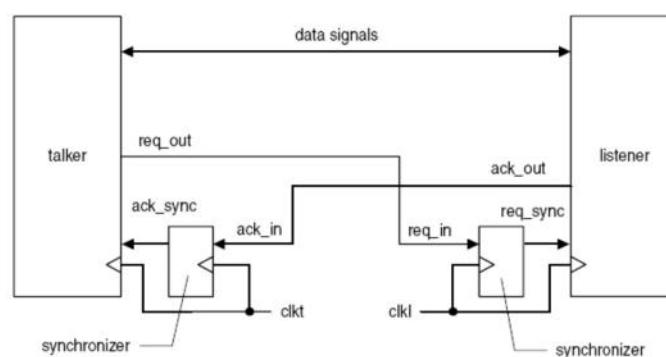
- Phase 1: Talker aktiviert $\text{req} = 1$ (A)
- Phase 2: Listener aktiviert $\text{ack} = 1$ (B, A)
- Phase 3: Talker deaktiviert $\text{req} = 0$ (B, A)
- Phase 4: Listener deaktiviert $\text{ack} = 0$ (B)
- Talker kann neuen Request starten

23

CE-DS 4 Synchronisation



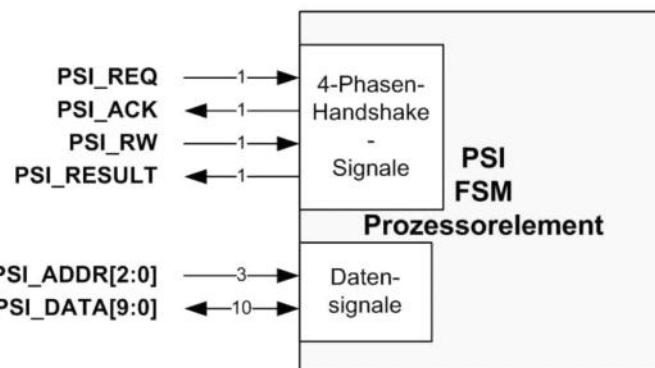
Synchronisation erforderlich für Talker und Listener in unterschiedlichen Clock Domains



aus P. P. Chu:
RTL Hardware Design Using VHDL, Wiley 2006
24

CE-DS 4 Synchronisation

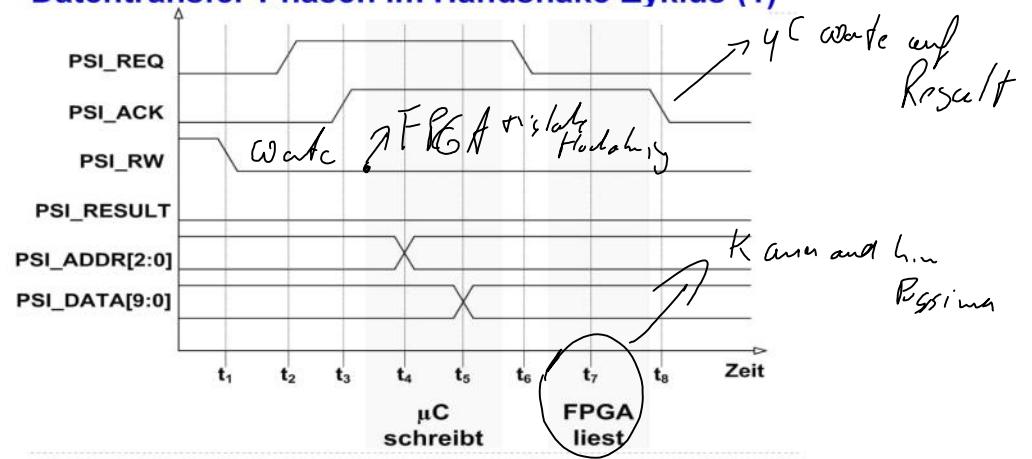
μ C – FPGA Interface



25

CE-DS 4 Synchronisation

Datentransfer-Phasen im Handshake-Zyklus (1)

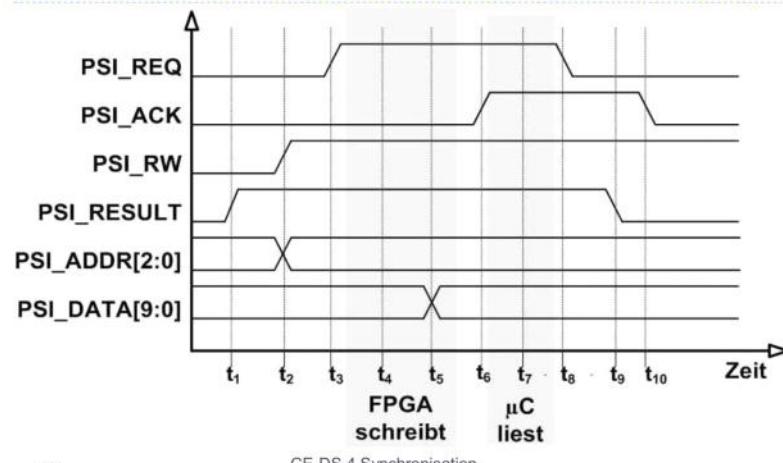


26

CE-DS 4 Synchronisation



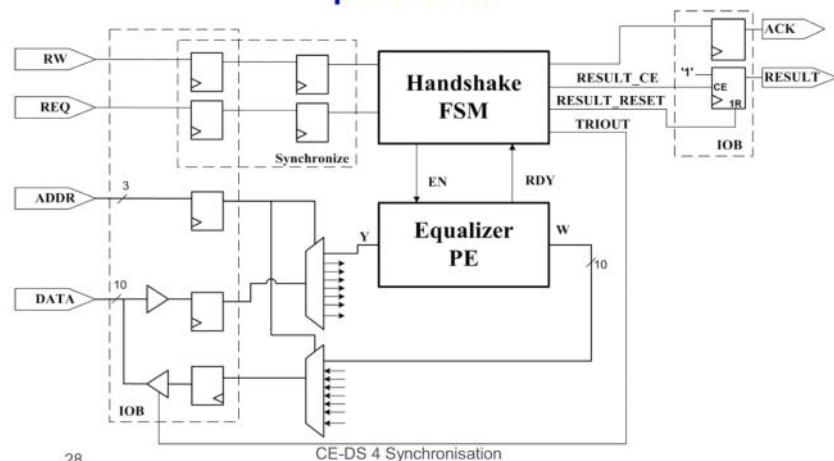
Datentransfer-Phasen im Handshake-Zyklus (2)



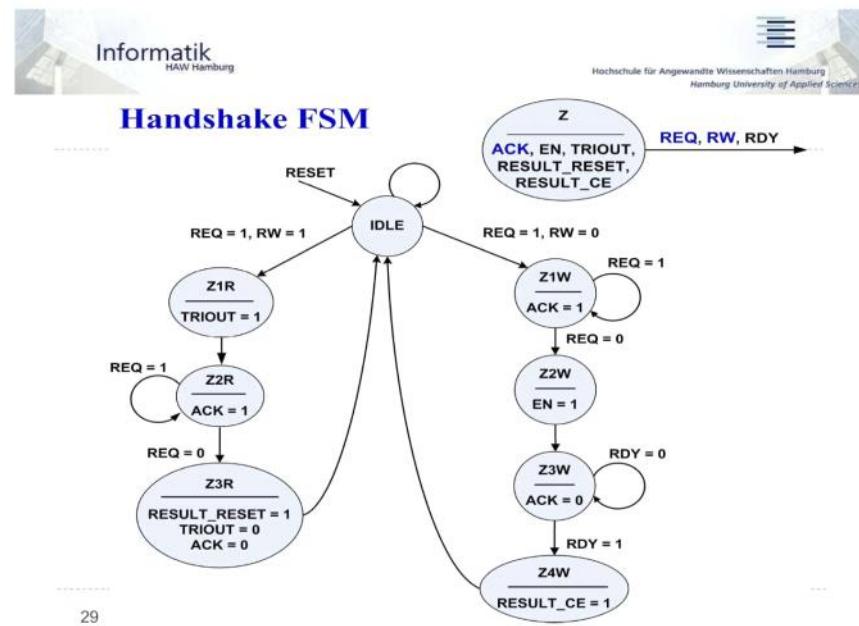
27



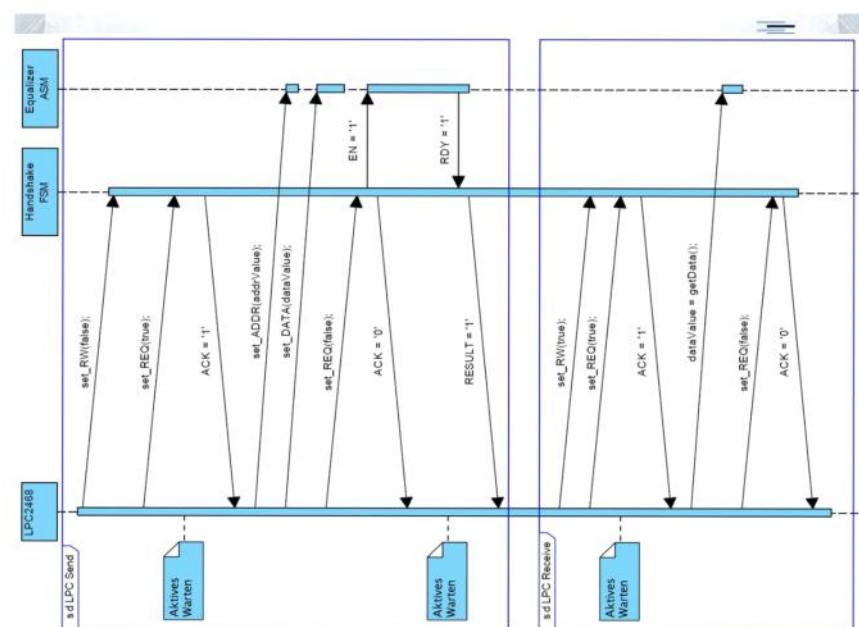
Asynchrone Kommunikation μC - FPGA



28

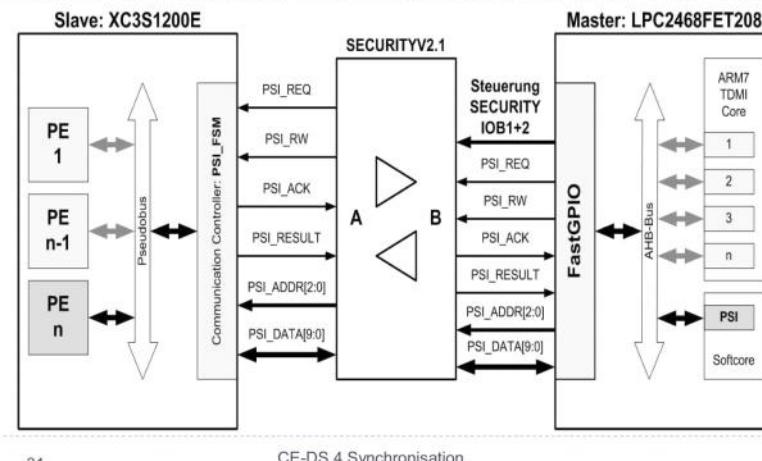


29





Security Board Interface

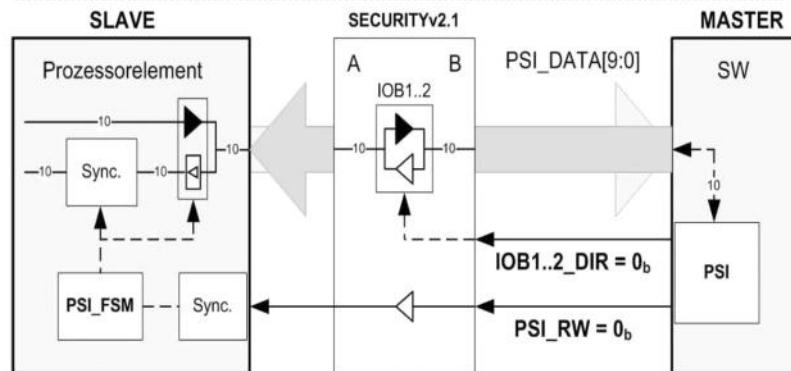


31

CE-DS 4 Synchronisation



Transfersteuerung µC - FPGA

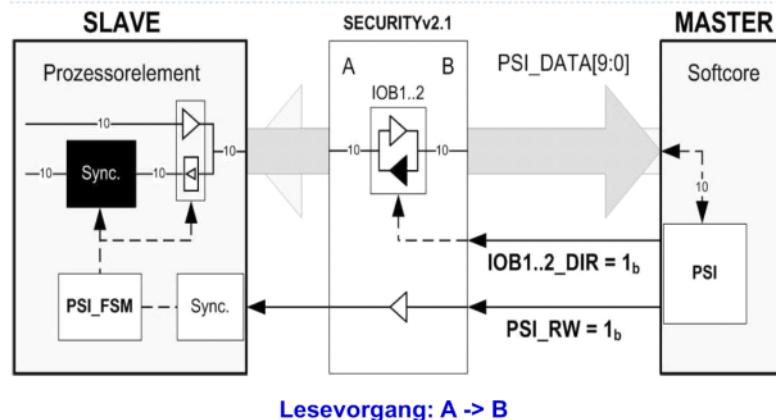


Schreibvorgang: B → A

32

CE-DS 4 Synchronisation

Transfersteuerung FPGA - µC



33

CE-DS 4 Synchronisation

Write – Sequenz in ISR

```
// Macro-Definitionen in hwconfig.h
...
/** LPC-2468 Write to FPGA */
#define PSI_ENABLE_WRITE /* Datenflussrichtung: B -> A */
#define PSI_SET_REQ; /* 1. Phase: Anforderung */
#define PSI_WAIT_ACK_SET; /* 2. Phase: Bestätigung */
#define PSI_WRITE_DATA( outdata ); /* 10 Bit Datum schreiben */
#define PSI_CLR_REQ; /* 3. Phase */
#define PSI_WAIT_ACK_CLR; /* 4. Phase; ACK = '0' Polling */
...
```

34

CE-DS 4 Synchronisation



Read – Sequenz in ISR

```
/** LPC-2468 Read from FPGA */
/* Ergebnisse verfügbar ?*/

PSI_ENABLE_READ;           /* Datenflussrichtung: A -> B */
PSI_SET_REQ;                /* 1. Phase: FPGA schreibt */
PSI_WAIT_ACK_SET;          /* 2. Phase */
PSI_READ_DATA( *indata );   /* 10 Bit Datum lesen */
PSI_CLR_REQ;                /* 3. Phase: Lesebestätigung */
PSI_WAIT_ACK_CLR;          /* 4. Phase: Lesesequenz beendet */

/* Datenausgabe an DAC und PWM */
```

35

CE-DS 4 Synchronisation

Daten-/Steuersignale		Anmerkung	Transfer	TI-LPC uC Port[Pin]	SECURITYv2.1			Development Boards		
Name	Typ				Bustreiber	Connector Typ	Pin	FPGA Signal	Pin	Connector FX2-100 Port-Pin
PSI_REQ	S	Auftrag	unidirektional	P1[12]	IOB4	X2	24	IOB4<3>	F11	J1A-34
PSI_ACK	S	Auftragsbestätigung	unidirektional	P1[8]	IOB5	X2	25	IOB5<2>	C14	J1A-41
PSI_RW	S	Schreiben/Lesen	unidirektional	P1[11]	IOB4	X2	23	IOB4<3>	F11	J1A-34
PSI_RESULT	S	Auftragsfertigstellung	unidirektional	P2[12]	IOB5	X3	9	IOB5<1>	A14	J1A-40
PSI_ADDR[0]	S	Adresse des Prozessorelements	unidirektional	P1[15]	IOB4	X2	33	IOB4<0>	B11	J1A-31
PSI_ADDR[1]				P1[14]			34	IOB4<1>	C11	J1A-32
PSI_ADDR[2]				P1[13]			29	IOB4<2>	E11	J1A-33
PSI_DATA[0]	D	Datum	bidirektional	P0[5]	IOB1	X3	36	IOB1<0>	A4	J1A-07
PSI_DATA[1]				P1[10]		X2	26	IOB1<1>	C3	J1A-08
PSI_DATA[2]				P0[13]		5	IOB1<2>	C4	J1A-09	
PSI_DATA[3]				P0[14]		7	IOB1<3>	B6	J1A-10	
PSI_DATA[4]				P0[19]		X3	15	IOB1<4>	D5	J1A-11
PSI_DATA[5]				P0[20]		14	IOB1<5>	C5	J1A-12	
PSI_DATA[6]				P0[21]		X2	19	IOB1<6>	F7	J1A-13
PSI_DATA[7]				P0[22]		18	IOB1<7>	E7	J1A-14	
PSI_DATA[8]				P0[29]		6	IOB2<0>	A6	J1A-15	
PSI_DATA[9]				P0[30]		15	IOB2<1>	C7	J1A-16	
IOB1_DIR	S	Transferrichtung IOB1- PSI_DATA[7:0]	unidirektional	P2[3]	IOB1	X3	32	X X X		
IOB2_DIR	S	Transferrichtung IOB2- PSI_DATA[9:8]	unidirektional	P1[4]	IOB2	X3	33	X X X		



```
PSI_WRITE_DATA( outdata );
/* 10 Bit Datum schreiben */

//Daten Ein- und Ausgabe
#define PSI_WRITE_DATA( d ) FIO0MASK = ~PSI_DATA_PINS_PORT0;
\

FIO0PIN = (d & (3<<8))<<(29-8) | (d & (0xf<<4))<<(19-4) | (d & 0xc)<<11 | (d &1)<<5;
\

FIO0MASK = 0;
\

FIO1MASK = ~PSI_DATA_PINS_PORT1;
\

FIO1PIN = (d & 2)<<9;
\

FIO1MASK = 0;
```

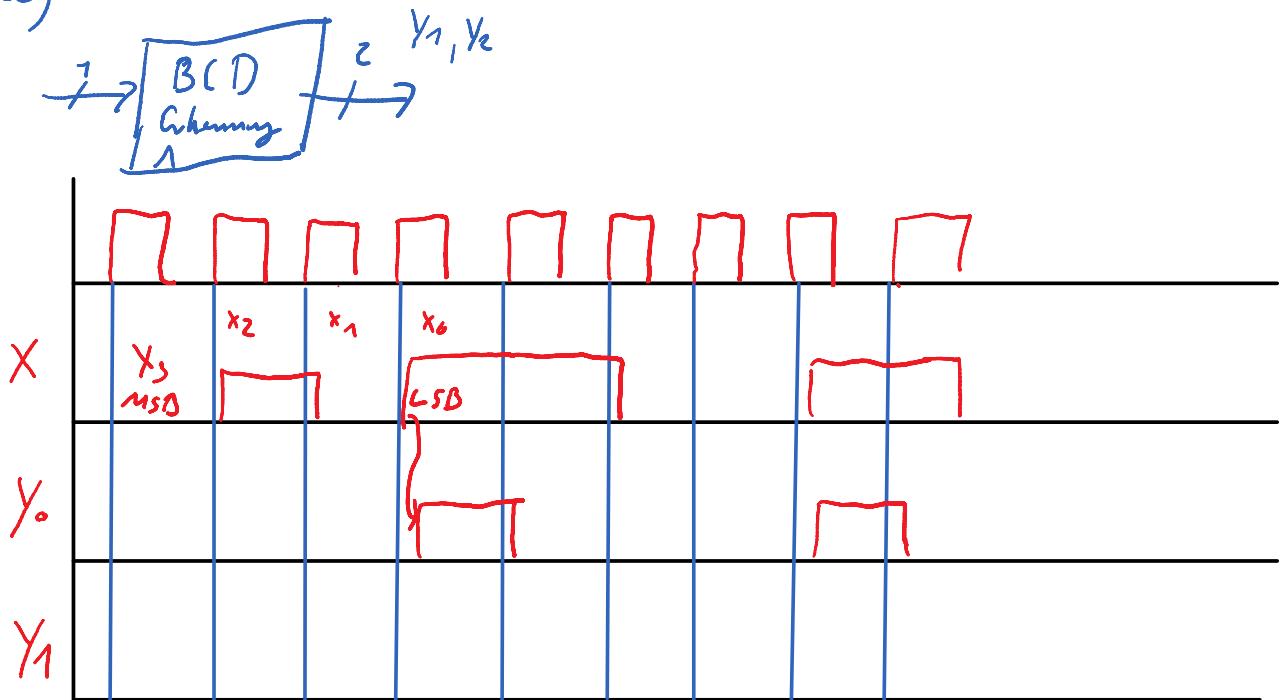
CE_Klausur

Mittwoch, 26. September 2012

15:30

13

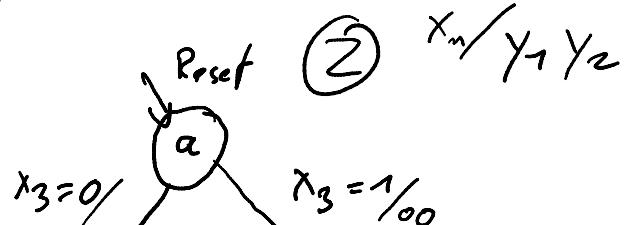
a)

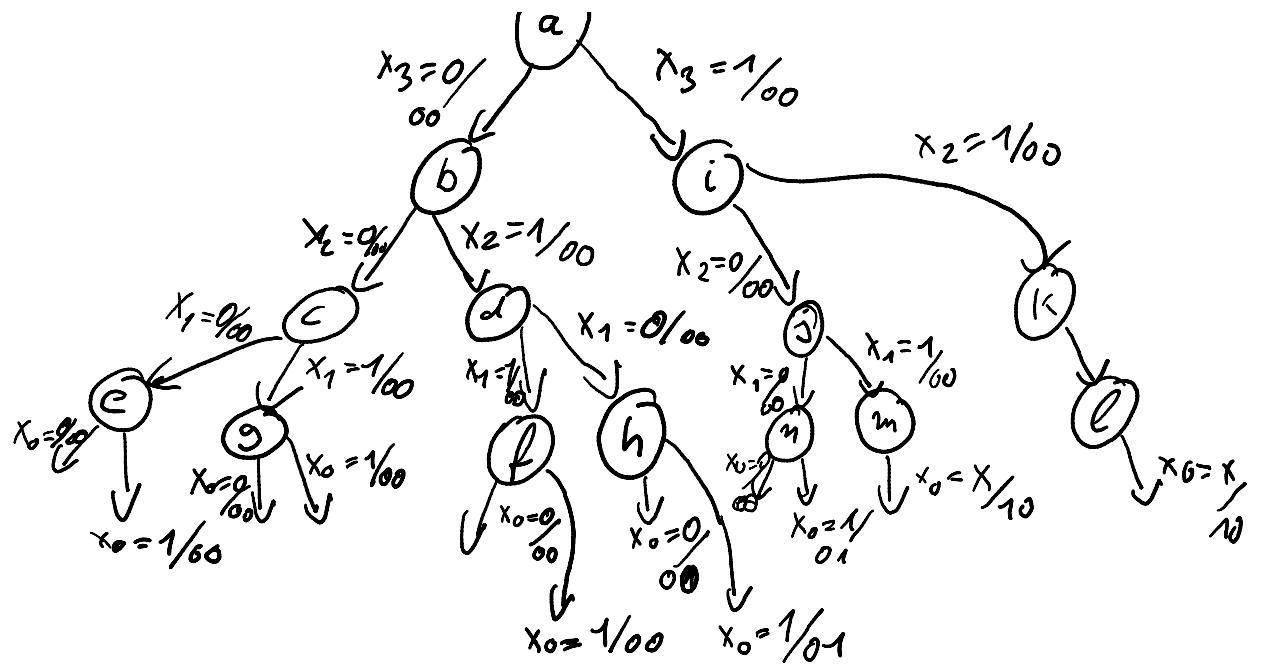


	x_3	x_2	x_1	x_0	y_1	y_0
0	0				0	1
1	1				0	1
2	0	1			0	0
3	1	1			0	0
4	0	0	1		1	0
5	1	0	1		1	0
6	0	0	0	1	0	1
7	1	0	0	1	0	1
8	0	1	0	0	0	0
9	1	1	0	0	0	0
10-15	x	x	x	x	1	0

A blue bracket below the table spans the columns for x_3, x_2, x_1, x_0 and is labeled "zu grob".

Mealy FSM





CE_SWR_KLAU

Dienstag, 4. Dezember 2012
08:44

Name:
Vorname:
Matrikelnr.:

Semestergruppe TI3

Digitaltechnik 2

08. Juli 2007

Gesamtdauer 120 Minuten
Fragenteil: 40 Minuten
Aufgabenteil: 80 Minuten

Zugelassene Hilfsmittel: Handout, eigene Mitschrift, Labormaterial, Lehrbücher.
Ergebnisse werden nur anerkannt, wenn die Herleitung vollständig ist und Aussagen
begründet sind.

	Erreichbare Punkte	Erreichte Punkte
Fragenteil	38	
A 1	16	
A 2	21	
A 3	25	
Summe	100	

Ergebnis

Datum:

1 Serieller 2er-Komplement-Umsetzer

Eine FSM soll einen seriellen Umsetzer realisieren, der auf vorzeichenbehaftete 3 Bit Worte das 2er-Komplement anwendet. Ein synchronisiertes Eingangssignal $x(t)$ liefert einen seriellen Bitstrom, der aus geschlossenen Sequenzen (keine Überlappung) von Codewörtern besteht: Drei Bits x_0 (LSB: Anfang), x_1 und x_2 (MSB, Vorzeichen) mit jeweils der Breite einer Taktperiode.

Die Mealy-FSM soll drei parallele Ausgangsbits $y(t) = (y_2(\text{Vorzeichen}), y_1, y_0)$ bestimmen, unmittelbar nachdem das Eingangsbit x_2 verfügbar ist. Während des Empfangszyklus können die Ausgangsbits (y_2, y_1, y_0) auf "000" gesetzt werden.

- Geben Sie ein Blackbox-Symbol an.
- Skizzieren Sie ein Timing der Ein- und Ausgangssignale sowie des Taktsignals clk für zwei aufeinander folgende Bitkombinationen $x(t)$.
- Stellen Sie eine Wahrheitstabelle für den Zusammenhang $y = f(x_2, x_1, x_0)$ auf und geben dabei für y auch die hexadezimale Darstellung an.
- Entwickeln Sie das Zustandsdiagramm, indem Sie die Zustände sukzessive mit a, b, c, ... bezeichnen.
- Argumentieren Sie qualitativ zur Zusammenfassbarkeit von Zuständen des Zustandsdiagramms.

2 Multizyklus-Datenpfadmodellierung eines Prozessorelementes

Betrachten Sie folgenden C-Codeausschnitt:

```
unsigned char A, B, C, D, E ;      /* inputs*/
special int    Y = 0;              /* type in question*/
Y = A*B*D + A*B*E + C*D + C*E;  /*result Y*/
```

Die Datenrate der synchronisierten Eingangsdaten A, B, C, D und E ist deutlich geringer als die Taktfrequenz. Aktualisierte Eingänge werden mit einem Enable-Impuls EN angezeigt und bleiben bis zum nächsten EN-Puls verfügbar: keine Speicherung erforderlich. Es sollen nur **ein Multiplizierer** und **ein Addierer** als arithmetische Funktionselemente zum Einsatz kommen.

- Entwickeln Sie eine ASM Chart mit einer minimalen Anzahl von Zuständen. Wenden Sie dazu auf den Ausdruck für Y das Distributivgesetz an. Der Ausgang Y ist mit einem Ready-Bit anzugeben.
- Geben Sie eine Signal-Lifetime Tabelle und ein Register-Sharing an.
- Skizzieren eine Multiplexerlösung des Datenpfades und tragen die gespeicherten Signale an die Register an. Kennzeichnen Sie die Arithmetikblöcke mit den jeweiligen Ergebnissen.
- Geben Sie die erforderlichen Vektorbreiten an. Legen Sie dazu die Register auf die maximal erforderliche Breite aus.

3 Pipelined Data Path

Betrachten Sie diesen C-Codeausschnitt:

```
signed char  B,C , D, E, F ;      /* inputs*/
special int   Y = 0, A, P ; /* type in question*/
if (B > D)
    A = B + C;
else
    A = D + E ;
P = F * A ;
Y = Y + P ;
```

- a) Skizzieren Sie einen äquivalenten Datenpfad mit vollständigem Pipelining. Es liegen keine Einschränkungen der verfügbaren HW-Ressourcen vor.
- b) Geben Sie für Eingangsgrößen im Q7-Format die erforderlichen Vektorbreiten der internen Signale an und kennzeichnen Sie die Vektorzusammensetzung mit Sign- und Guard-Bits.
- c) Die Breite der Ergebnisgröße Y soll von Reset zu Reset für die Verarbeitung von 256 Eingangsdatensätzen ausgelegt werden.

CE_SWR_Skurve

Dienstag, 13. November 2012
08:55

CE_HTM_F1

Dienstag, 2. Oktober 2012
11:10

Start up Code : Cpu-Takt

Linker : Speicher
Reset addr
Stack / Heap \rightarrow Malloc / Free



Computer Engineering WS 2012

Einleitung

HTM – SHF - SWR



CE WS12

Computer Engineering



Themen:

- Ein- und Ausgabe, Peripherie
 - Memory Mapped, IO Mapped
 - Einfache Peripherie, General Purpose IO (GPIO)
 - Timer
 - Interruptverarbeitung
 - Serielle Übertragung
 - Speicher: RAM, Flash, EEPROM
- Softwareerstellung für Embedded Systems
 - Erstellung, Download, Debuggen
 - Startup-Code

Literatur:

- ▶ LPC24XX User Manual
http://www.nxp.com/acrobat_download/usermanuals/UM10237.pdf
- ▶ LPC2468 Data Sheet
http://www.nxp.com/acrobat_download/datasheets/LPC2468.pdf
- ▶ The Insider's Guide To The NXP LPC2300/2400 Based Microcontrollers, Hitex
<http://www.hitex.com/index.php?id=download-insiders-guides>
- ▶ ARMv5 Architecture Reference Manual, gilt u. a. für ARM7
<http://www.arm.com/miscPDFs/14128.pdf>
- ▶ Sloss, Symes, Wright : ARM System Developer's Guide, Morgan Kaufmann Publishers, 2004.
- ▶ William Hohl: ARM Assembly Language, CRC Press, 2009.
- ▶ Helmut Bähring: Mikrorechner-Technik, Springer, 2002.
- ▶ Wayne Wolf: Computer as Components, Morgan Kaufmann, 2008.

3

Übersicht

- 
- ▶ Einleitung
 - ▶ Softwareerstellung

4

Was kennzeichnet ein eingebettetes System?

- › Gerät, das ein programmierbares Prozessorsystem enthält
- › Kein Allzweckrechner
- › **PC selbst ist kein eingebettetes System**
- › **PC kann aber zum Aufbau eines eingebetteten Systems verwendet werden**

5

Anwendungsbereiche

- › Automatisierung
 - › Steuern und Regeln von Prozessen, Fertigungs- und Produktionsanlagen, Überwachung
- › Medizintechnik
 - › Beatmungsgeräte, Narkosesysteme
 - › Computer-Tomograph, Ultraschallgeräte
- › Netzwerk
 - › Router, Switches, WLAN, VoIP
- › Haushalt
 - › Waschmaschine, Mikrowelle, ...
- › Unterhaltung
 - › Handys, Digitalkameras
 - › Digitales Fernsehen, Video-on-demand, Hi-Fi, DVD, Spiele, ...

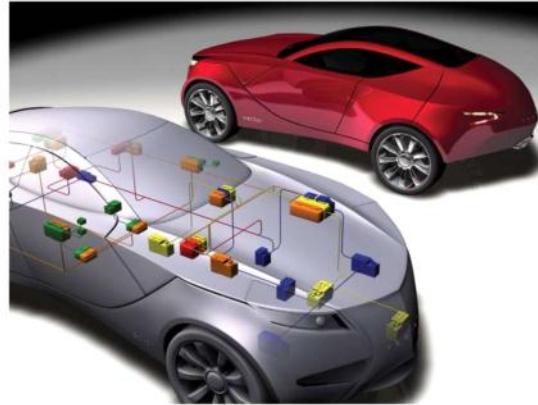


6



Anwendungsbereiche

- Automobiltechnik
 - Oberklasse:
Anzahl Steuergeräte >80
 - Diverse Bussysteme:
 - CAN
 - LIN
 - Flexray
 - MOST
 - Anwendungen:
 - Motorsteuerung
 - x-by-wire:
Bremsen,
Lenkung
 - Komfort
 - Unterhaltung



7



Typische Anforderungen an eingebettete Systeme

- Hohe Ausfallsicherheit
- kurze Reaktionszeiten
- vorgegebene technische Randbedingungen und Schnittstellen
- Geringer Leistungsverbrauch
- Niedrige Entwicklungs-/Produktionskosten

8

Eingebettetes Prozessorsystem: Typische Komponenten

- CPU
- Speicher
 - **Programmspeicher (Flash), Datenspeicher (RAM), Parameterspeicher (EEPROM)**
- Zeitgeber
 - Steuerung von periodischen Abläufen
 - PWM (Puls-Weiten-Modulation)
- Ein- und Ausgabe
 - **Digitale Schnittstellen**
 - parallel, seriell (RS232, USB, I2C, CAN, ...)
 - **Analoge Schnittstellen**
 - Analogie Ein- und Ausgabe
- Systemmanagement
 - Speicherverwaltung
 - Interruptsteuerung
 - DMA-Steuerung
 - Powermanagement, Ausfallsicherheit (Watchdog)

9

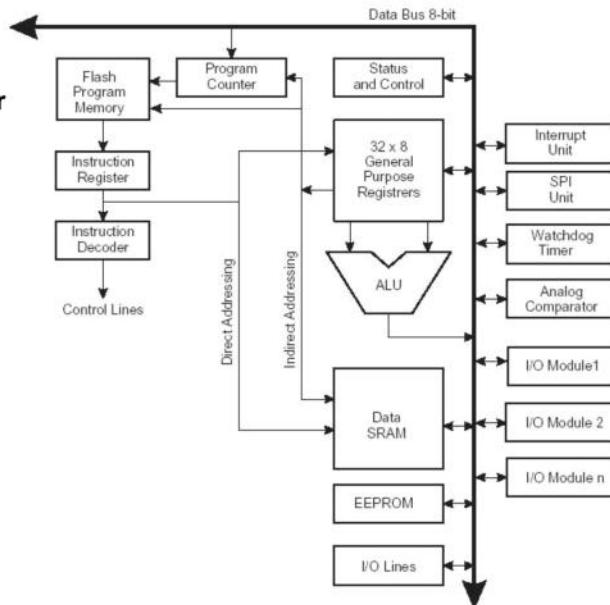
Mikrocontroller

- Enthält viele Komponenten auf einem Chip, z. B.:
 - CPU
 - Speicher
 - RAM
 - Flash
 - Zeitgeber
 - I/O
- Unterscheiden sich in
 - Geschwindigkeit (Taktfrequenz)
 - Breite Datenbus (z. B.: 4, 8, 16, 32 Bit)
 - Speichergrößen
 - I/O Komponenten

10

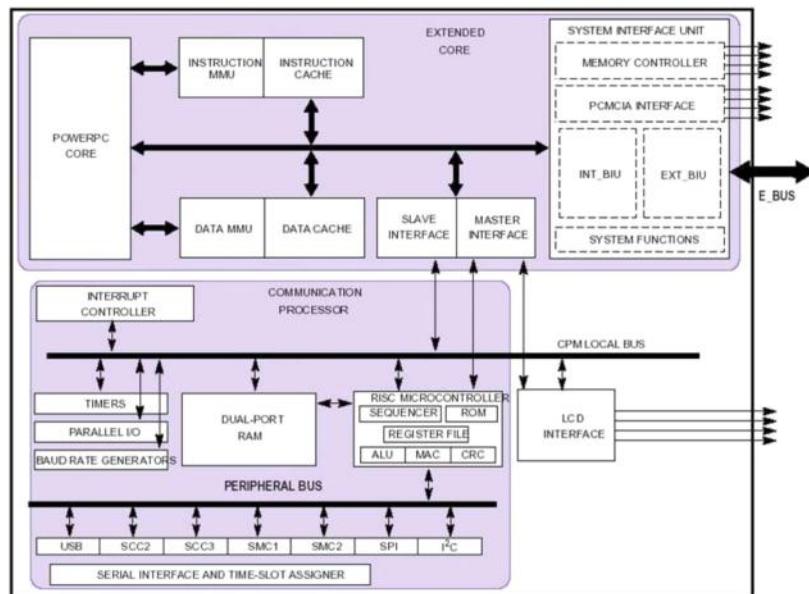
Mikrocontroller ATMEL, AVR

- ▶ Harvard-Architektur
- ▶ 8-Bit Datenbus
- ▶ I/O Module:
 - 53 I/O Lines
 - 4 Timer
 - A/D Converter
 - 2 RS232
 - 1 SPI
 - CAN Feldbus



11

Mikrocontroller Freescale, MPC823e (Power PC)



12

Übersicht

- ▶ Einleitung
- ▶ Softwareerstellung



13

Besonderheiten Softwareentwicklung für Eingebettete Systeme

- ▶ Aufteilung der Implementierung in
 - ▶ **Hardware-basiert**
 - ▶ **Software-basiert**
- ▶ Betriebssysteme
 - ▶ **häufig kein Betriebssystem**
 - ▶ **wenn Betriebssystem, dann meist**
 - kein Speicherschutz
 - echtzeitfähig
 - erfordern meist BSPs (Board Support Package)
- ▶ **Massenspeicher**
 - ▶ **häufig kein Massenspeicher**
 - Programme werden im ROM und
 - nichtflüchtige Daten im EEPROM gespeichert
- ▶ **Softwareerstellung**
 - ▶ **Verwendung von Crosscompilern**
 - erfordert „Download des erzeugten Codes“

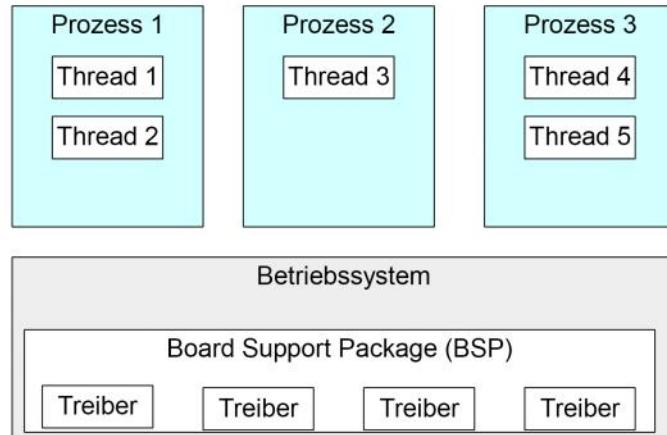
14

Besonderheiten Softwareentwicklung für Eingebettete Systeme

- Debugging, Fehlersuche
 - erfordert zusätzliche Werkzeuge
 - Simulator
 - Crossdebugger, Anbindung über
 - serielle Schnittstelle
 - Netzwerk
 - JTAG-Interface
 - Logikanalysator
 - Fehlersuche ohne Beeinflussung des Zeitverhaltens
 - Emulator
 - Testen
 - erfordert
 - Einbindung in die reale Umwelt oder
 - Verwendung von „Hardware in the Loop-Technik“
 - Softwarepflege
 - erfordert Möglichkeit zum Firmware-Update

15

Embedded System mit Betriebssystem



16



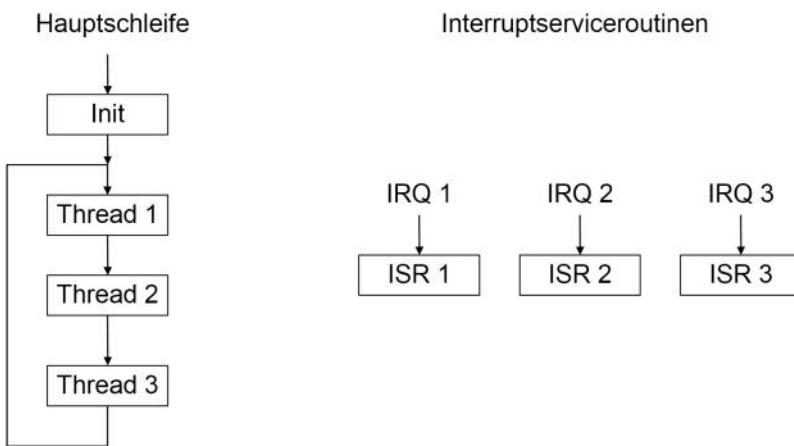
Typischer Aufbau einer Thread

```
int variable1;  
  
double variable2 = 3.14;  
  
void *PrintHello(void *threadarg){  
  
    int variable3;  
  
    initialisierung();  
  
    while( 1 ) {  
  
        warte_auf_Ereignisse();  
  
        bearbeitung();  
  
        ausgabe();  
    }  
}
```

17

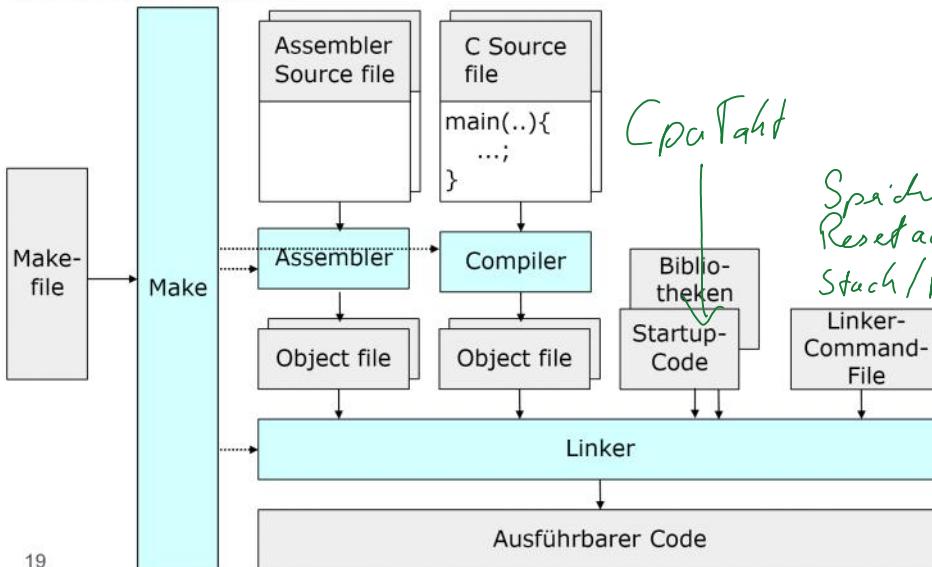


Embedded System ohne Betriebssystem



18

Softwareentwicklung



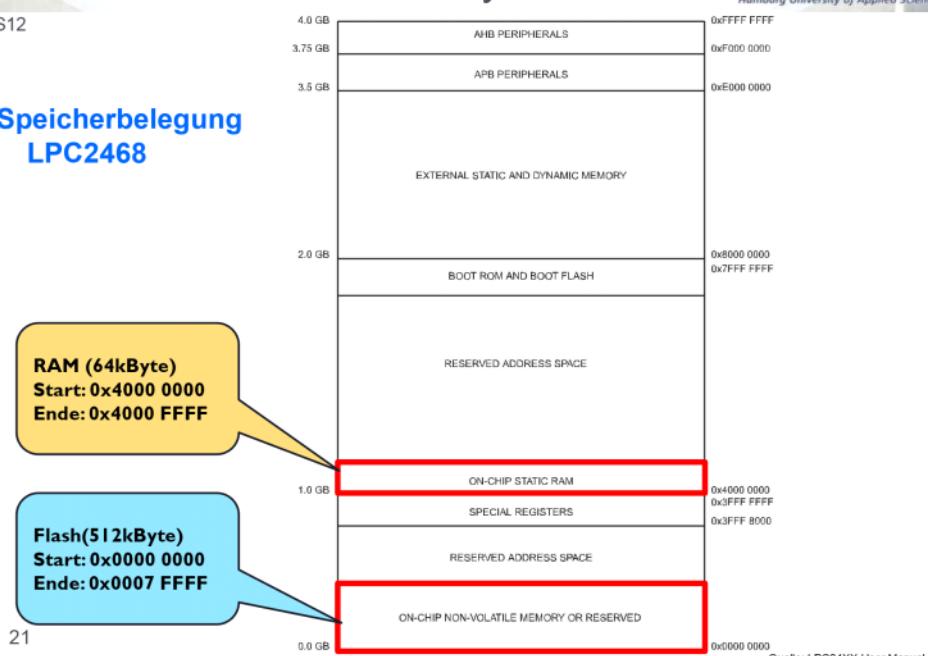
19

Speicherarten

ROM (Flash)	<ul style="list-style-type: none"> Persistente Speicherung Löschbar (blockweise) N-mal wiederholbar (typ. 1000) Langsam Preiswert 	Programmspeicher
RAM	<ul style="list-style-type: none"> Flüchtige Speicherung Schnell Teuer 	Arbeitsspeicher
EEPROM	<ul style="list-style-type: none"> Persistente Speicherung Bytewise löschbar N-mal wiederholbar (typ. 10⁶) Langsam Teuer 	Speicherung von Konfigurationsdaten

Embedded Systems

Speicherbelegung LPC2468



Embedded Systems

Sektionen

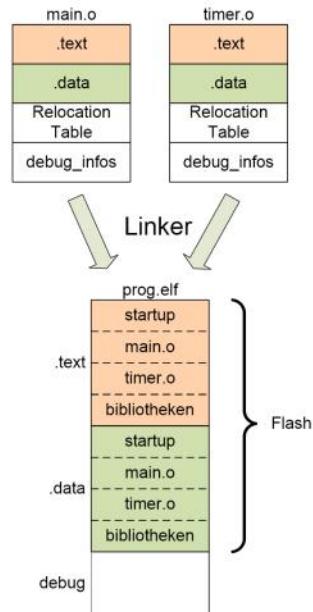
- Ausführbarer Code ist in Sektionen aufgeteilt
- Ermöglicht gezielte Platzierung des erzeugten Codes im Speicher
- Skript für Linker (Loader) gibt die Adressen der Sektionen vor

Wichtige Sektionen:

Name	Inhalt	Speicherort	Bemerkung
.text	Programmcode <i>Konstanten</i>	FLASH	
.data	Initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen vom FLASH in das RAM kopiert
.bss	Nicht initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen gelöscht

Linker (loader, ld)

- ▶ Zusammenfügen der Sektionen
- ▶ Zuweisung der Adressen
 - ▶ Jede Sektion hat zwei Adressen:
 - Virtual Memory Address (VMA): Adresse der Sektion, wenn das Programm ausgeführt wird.
 - Load Memory Address (LMA): Adresse, unter der die Daten abgelegt werden.
- ▶ Anwendung der Relocation Table
- ▶ Platzhalter für Adressen werden durch die tatsächlichen Adressen ersetzt.



23

Programmiersprache C: Speicheraufteilung

- ▶ Verwendung des Speichers:
- ▶ Konstante (Verwendung des Schlüsselworts **const**):
 - Spezielle Sektion **.rodata**.
- ▶ Globale und statische Variable:
 - **.data** oder **.bss** (Abhängig von Initialisierung)
- ▶ Lokale Variable:
 - Stack
- ▶ Dynamische Variable:
 - Heap *Malloc / free*

Dynamische Verwendung
Speicherbedarf schwer vorhersagbar.
Kritisch: Rekursive Funktionsaufrufe.

24

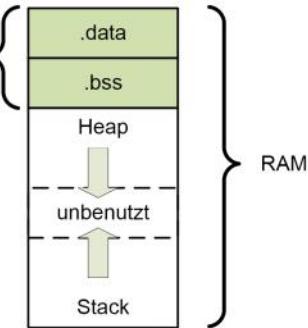
Programmiersprache C: Speicheraufteilung

- › Häufige Anordnung:

- › **Heap:**

- Beginnt am Ende der .bss-Sektion.
- Wächst von kleinen nach großen Adressen.

statisch



- › **Stack:**

- Beginnt an der höchsten RAM-Adresse.
- Wächst von großen nach kleinen Adressen.

- › **Vorteil:** Sehr flexibel.

- › **Nachteil:** Kein Schutz vor Stack- oder Heapüberlauf.

- › In Embedded Systems muss der Heap- und Stackbedarf vorab abgeschätzt werden.
Heap und Stack sollten statisch eingerichtet werden.

25

Programmiersprache C: Speicheraufteilung HiTOP

- › ARM 7 TDMI:

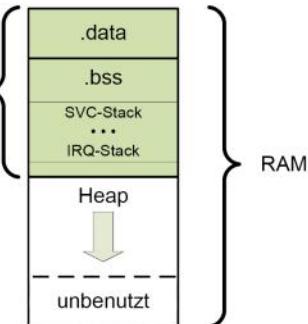
- › **6 Stacks für 7 Betriebsarten**

- USR/SYS haben gemeinsamen Stack

- › **Größen der Stacks werden statisch in der .bss-Sektion festgelegt**

- Stacks werden in `start.s` definiert

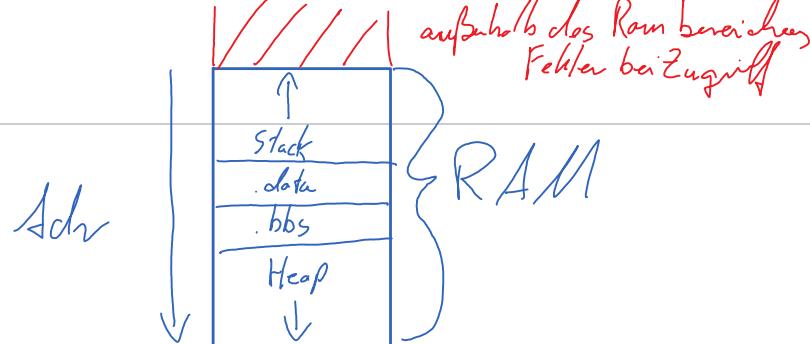
statisch



- › **Heap nutzt den gesamten restlichen Speicher ab Ende der .bss-Sektion**

- Definiert durch die Laufzeitbibliothek

26



Programmiersprache C: Laufzeitbibliothek

- Meist nicht Bestandteil des Compilers.
- Üblich:
Verbindung der Anwendung mit dem Betriebssystem.
 - Betriebssystem wird über Software-Interrupt aufgerufen:
 - Umschaltung in System-Betriebsmodus.
 - Höhere Privilegien.
 - Wesentliche Aufgaben:
 - Ein- und Ausgabe, Dateien und Geräte.
 - Prozessverwaltung, Starten und Stoppen von Prozessen.
 - Speicherverwaltung.

27

Programmiersprache C: Laufzeitbibliothek HiTOP

- Verwendung der **newlib**:
 - Open-source Projekt
 - Standard C Bibliothek für Embedded Systems
 - Wird gepflegt von Red Hat Entwicklern
- Kann einfach angepasst werden:
 - Unterstützung einer Vielzahl von CPU-Typen
 - Verwendung von insgesamt 17 Stub-Funktionen (System Calls).
 - Optimierung des Speicherbedarfs (printf versus iprintf).
 - Kann reentrant-fähig übersetzt werden.

28



Programmiersprache C: Laufzeitbibliothek HiTOP

- newlib: System Calls, Beispiel Speicherallokierung

```
extern unsigned char end[];
extern unsigned char _end_of_heap[];
static unsigned char *heap_ptr = NULL;

void* __sbrk_r( struct _reent *_s_r, ptrdiff_t nbytes ) {
    unsigned char *prev_heap_end;

    if ( heap_ptr == NULL)
        heap_ptr = end;

    prev_heap_end = heap_ptr;

    if ((heap_ptr + nbytes) > _end_of_heap_){
        return NULL;
    }

    heap_ptr += nbytes;

    return (void*)prev_heap_end;
}
```

29



Programmiersprache C: Laufzeitbibliothek HiTOP

- newlib: System Calls, Beispiel Ausgabe

```
_ssize_t _write_r ( struct _reent *r, int file, const void *ptr, size_t len) {
    int i;
    const unsigned char *p;
    int retval;

    p = (const unsigned char*) ptr;

    for (i = 0; i < len; i++) {
        if (*p == '\n' ){
            do{
                retval = uart0Putch('\r');
            }while(retval == -1);
        }

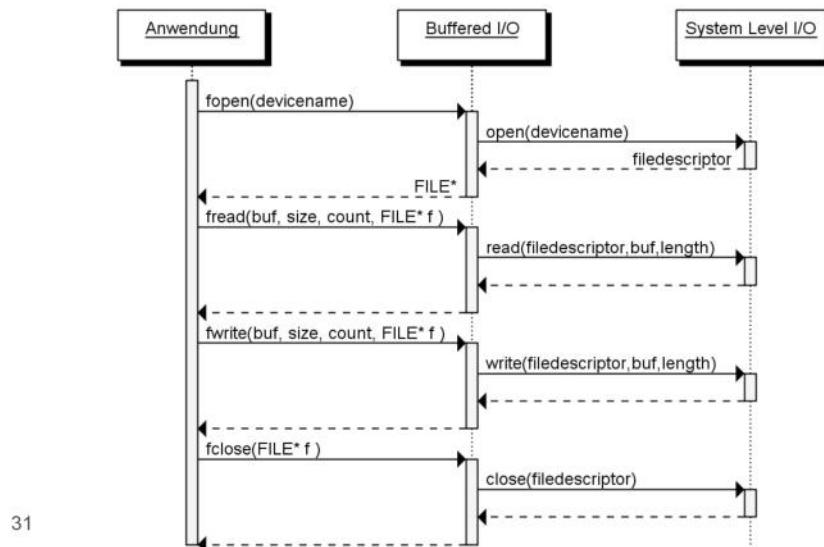
        do{
            retval = uart0Putch(*p);
        }while(retval == -1);
        p++;
    }

    return len;
}
```

30

Programmiersprache C: Laufzeitbibliothek HiTOP

newlib: System Calls, Beispiel Ausgabe



Computer Engineering

WS 2012

Interruptverarbeitung

HTM – SHF - SWR

Interrupt-verarbeitung

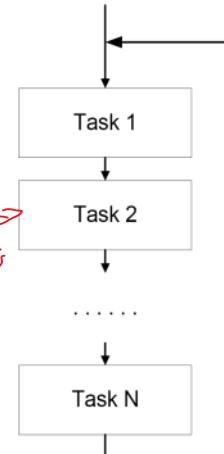
Programmaufbau ohne Betriebssystem

► Typisch:
Die einzelnen Aufgaben werden der Reihe nach in einer Hauptschleife abgearbeitet.

► Maximale Reaktionszeit:

Summe der maximalen Reaktionszeiten aller Tasks

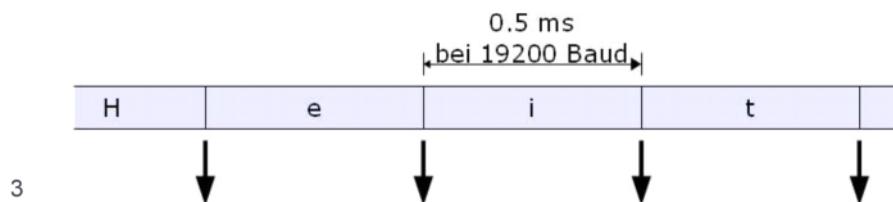
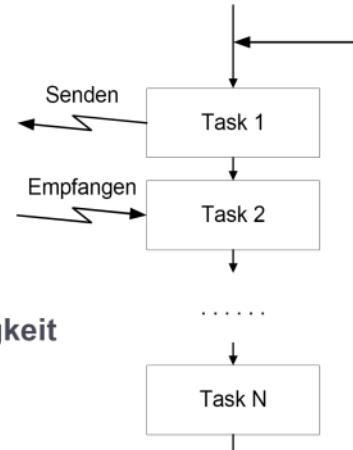
Ext. Ereignis



Interrupt-verarbeitung

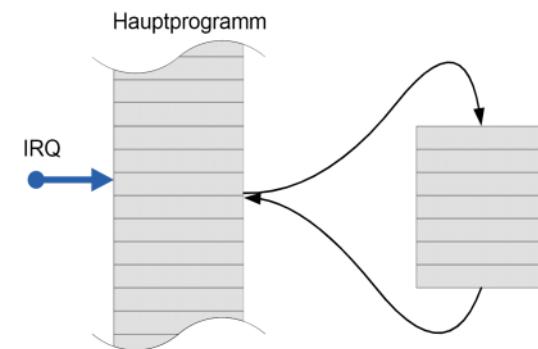
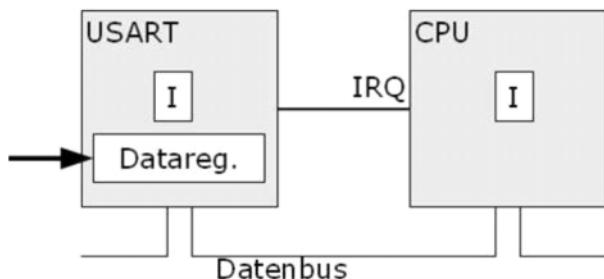
Reaktion auf externes Ereignis

- ▶ Regelmäßige Abfrage des Statusregisters z.B.: „Receiver Data Ready“ oder „Transmitter Empty“
- ▶ Falls nichts zu tun ist: Task wird beendet
- ▶ Andernfalls werden die Daten empfangen oder gesendet und verarbeitet.
- ▶ Typisch:
Empfangsdaten kommen mit fester Geschwindigkeit
 - ▶ Erfordert ausreichend kleine Reaktionszeit



Interrupt-verarbeitung

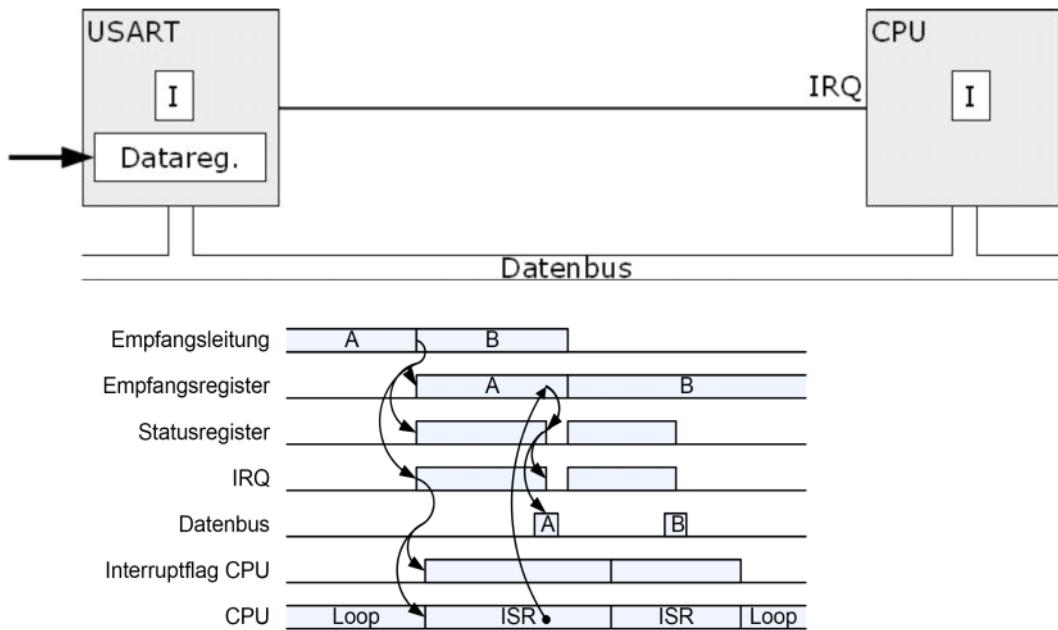
Interruptverarbeitung



- ▶ IRQ unterbricht laufendes Hauptprogramm.
 - ▶ Aktuelle Instruktion des Hauptprogramms wird vollständig abgearbeitet.
 - ▶ Zustand der CPU (Kontext) wird gerettet.
 - ▶ Abarbeitung der Interrupt Service Routine (ISR).
 - ▶ Alter Zustand der CPU wird wieder hergestellt.
 - ▶ Fortsetzung des Hauptprogramms an unterbrochener Stelle.

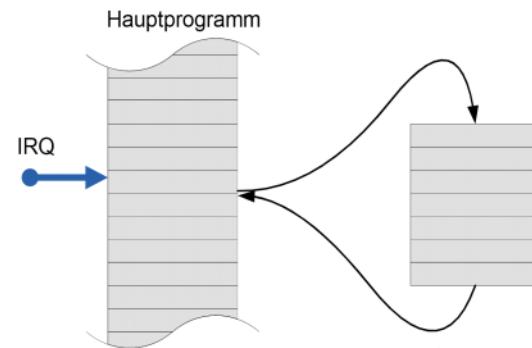
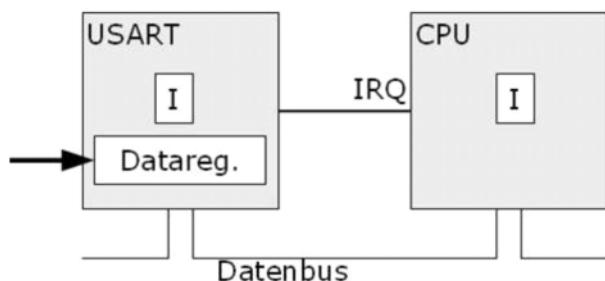
Interrupt-verarbeitung

Einzelne Interruptquelle



Interrupt-verarbeitung

Interruptverarbeitung



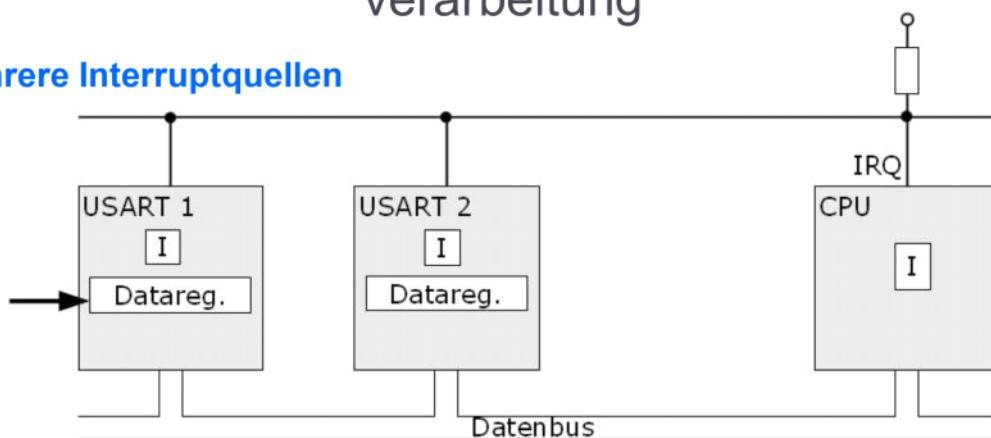
Interrupt
Service
Routine

► **Wichtig:**

- **ISR darf Zustand der CPU (Kontext) nicht verändern!**
- **Alle von der ISR verwendeten Register**
 - Statusregister, Datenregister**müssen zuvor gerettet und hinterher wieder hergestellt werden.**
- **Retten kann z.B. erfolgen:**
 - auf dem Stack
 - in zusätzlichen Registersätzen.

Interrupt-verarbeitung

Mehrere Interruptquellen

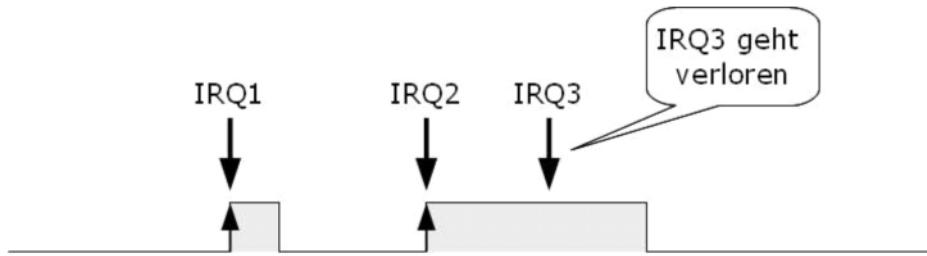


- ▶ Signalisierung über gemeinsame IRQ-Leitung (Open-Kollektor-Prinzip)
- ▶ Probleme:
 - ▶ Was passiert, wenn beide Bausteine gleichzeitig Interrupt auslösen?
 - ▶ Wie findet die CPU die aktive Interruptquelle?

7

Interrupt-verarbeitung

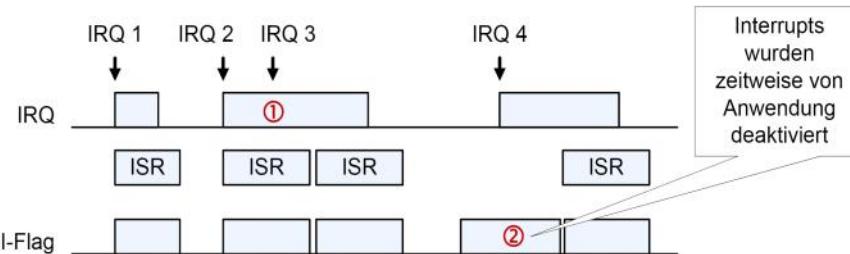
Flankensensitiver Interrupt



- ▶ CPU reagiert auf eine Flanke des Interruptsignals.
- ▶ Gut geeignet z.B. zum Zählen von Impulsen.
- ▶ Problematisch bei mehreren Interruptquellen, Interrupts können verlorengehen.
- ▶ Wird verwendet bei „Nicht maskierbaren Interrupts“ (NMI)

Interrupt-verarbeitung

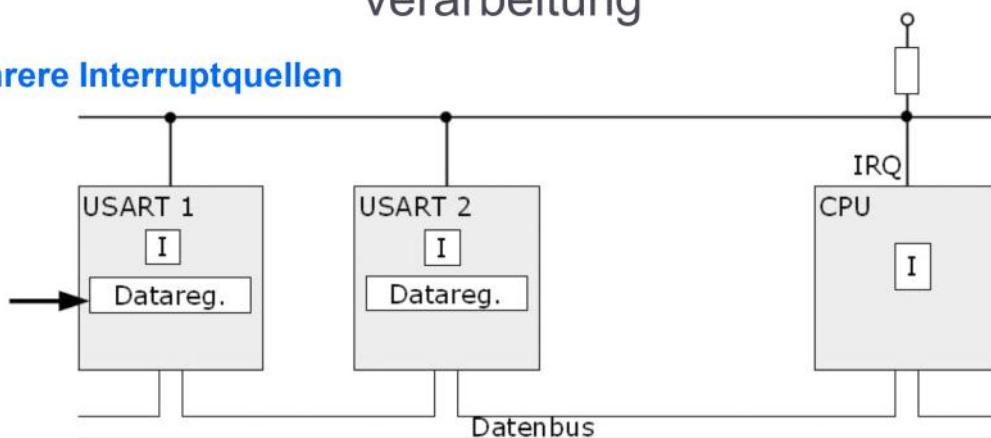
Pegelsensitiver Interrupt



- ▶ CPU reagiert auf den **Pegel** des Interruptsignals.
- ▶ Erfordert **Interruptflag** zum Deaktivieren der IRQ-Logik.
 - ▶ Ohne I-Flag würde nach gestarteter ISR sofort die nächste Unterbrechung ausgelöst!
 - ▶ Mehrere gleichzeitige Interruptanfragen gehen nicht mehr verloren. ①
 - ▶ I-Flag gesetzt:
 - 9 Interrupts bleiben erhalten, bis Interruptlogik wieder aktiv wird. ②

Interrupt-verarbeitung

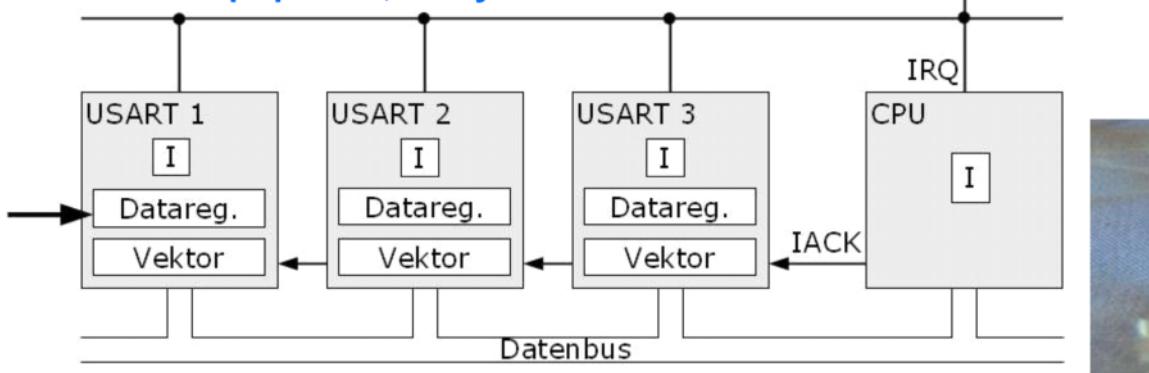
Mehrere Interruptquellen



- ▶ Auswahl der ISR per Software:
 - ▶ Polling der Statusregister in den Peripheriebausteinen.
 - ▶ Reihenfolge (Priorität) kann per Software festgelegt werden.
 - ▶ Problematisch: Zeitaufwand zum Auffinden der Interruptquelle.

Interrupt-verarbeitung

Mehrere Interruptquellen, Daisy-Chain



- ▶ Bausteine sind in Daisy-Chain angeordnet
- ▶ CPU-Hardware aktiviert IACK
- ▶ Erster Baustein in der Kette mit aktiviertem IRQ:
 - IACK wird nicht weitergereicht
 - IRQ-Vektor wird auf Datenbus gelegt
- ▶ Priorität wird durch Daisy-Chain festgelegt.

11

Interrupt-verarbeitung

Interruptvektor

- ▶ Häufig: Integerzahl, z.B. 8-Bit.
- ▶ Wird während der Initialisierung der Bausteine in das entsprechende Register geschrieben.
- ▶ Verweist auf einen Eintrag der **Vektortabelle**
- ▶ Vektortabelle kann enthalten:
 - ▶ Startadresse der ISR oder
 - ▶ ersten Befehl der ISR (meist Sprungbefehl zur eigentlichen Routine).
- ▶ Durchführung des IACK-Zyklusses und Auswertung der Vektortabelle meist durch Hardware der CPU

Interrupt-verarbeitung

Beispiel: Vektortabelle PC

- ▶ Vektortabelle enthält Startadressen der ISR (32-Bit).
- ▶ Hardwareinterrupts sind IRQ0 bis IRQ15
- ▶ Zusätzlich sind sogenannte Ausnahmen enthalten, z.B.:
 - ▶ Teilen durch 0
 - ▶ Unbekannter Befehl

Nr.	Adresse	Belegung
0	000-003	CPU: Division durch 0
1	004-007	CPU: Einzelschritt
2	008-00B	CPU: NMI (Fehler in RAM-Baustein)
3	00C-00F	CPU: Breakpoint erreicht
4	010-013	CPU: numerischer Überlauf
5	014-017	Hardcopy
6	018-01B	unbekannter Befehl (nur 80286)
7	01D-01F	reserviert
8	020-023	IRQ0: Timer (Aufruf 18,2 mal/s)
9	024-027	IRQ1: Tastatur
0A	028-02B	IRQ2: zweiter IR-Baustein 8259 (nur AT)
0B	02C-02F	IRQ3: serielle Schnittstelle 2
0C	030-033	IRQ4: serielle Schnittstelle 1
0D	034-037	IRQ5: Festplatte
0E	038-03B	IRQ6: Diskette
0F	03C-03F	IRQ7: Drucker
....
68-6F	1A0-1BF	frei für Anwendungsprogramme
70	1C0-1C3	IRQ08: Echtzeituhr (nur AT)

Interrupt-verarbeitung

Beispiel: Vektortabelle AVR-Familie (8-Bit CPU)

- ▶ Vektortabelle ersten Befehl der ISR (32-Bit).
- ▶ **meist Sprungbefehl zur eigentlichen ISR**

Vector No.	Program Address ⁽²⁾	Source	Interrupt Definition
1	0x0000 ⁽¹⁾	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	0x0002	INT0	External Interrupt Request 0
3	0x0004	INT1	External Interrupt Request 1
4	0x0006	INT2	External Interrupt Request 2
5	0x0008	INT3	External Interrupt Request 3
6	0x000A	INT4	External Interrupt Request 4
7	0x000C	INT5	External Interrupt Request 5
8	0x000E	INT6	External Interrupt Request 6
9	0x0010	INT7	External Interrupt Request 7
10	0x0012	TIMER2 COMP	Timer/Counter2 Compare Match
11	0x0014	TIMER2 OVF	Timer/Counter2 Overflow
12	0x0016	TIMER1 CAPT	Timer/Counter1 Capture Event
13	0x0018	TIMER1 COMPA	Timer/Counter1 Compare Match A
14	0x001A	TIMER1 COMPB	Timer/Counter1 Compare Match B
15	0x001C	TIMER1 COMPC	Timer/Counter1 Compare Match C
16	0x001E	TIMER1 OVF	Timer/Counter1 Overflow
17	0x0020	TIMER0 COMP	Timer/Counter0 Compare Match
18	0x0022	TIMER0 OVF	Timer/Counter0 Overflow
19	0x0024	CANIT	CAN Transfer Complete or Error
20	0x0026	OVRIT	CAN Timer Overrun
21	0x0028	SPI, STC	SPI Serial Transfer Complete

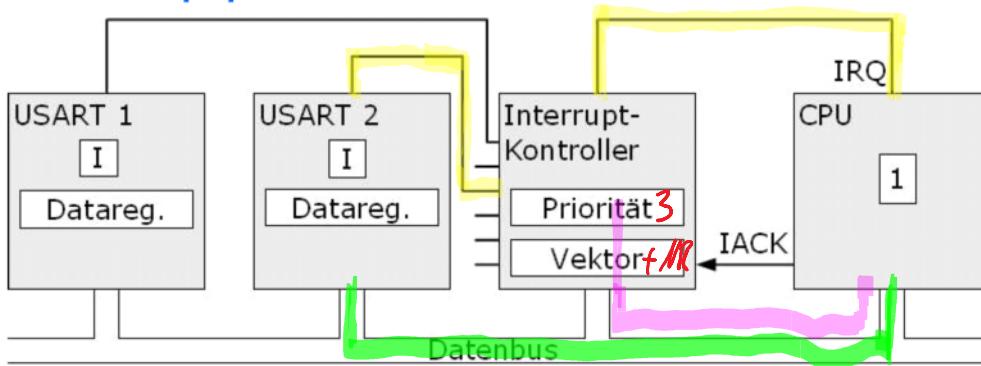
Mehrere Interruptquellen



15 Software Technisch
Schnell, da auch niedrige
IRQ interrupten können
⇒ keine proaktive
IRQ Verarbeitung

Interrupt-verarbeitung

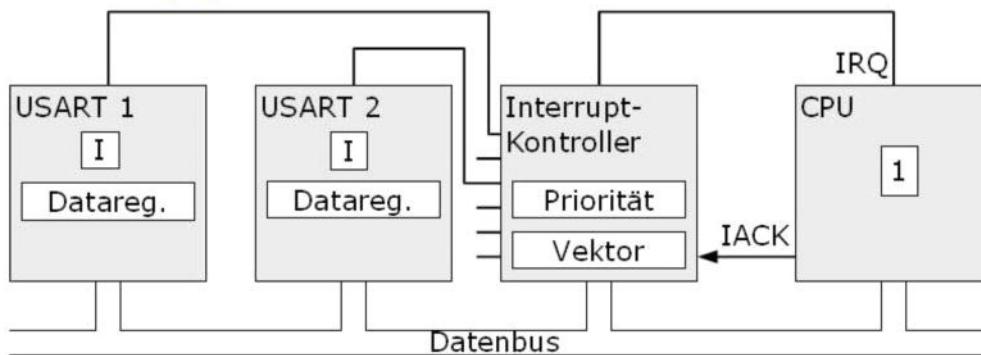
Mehrere Interruptquellen



- ▶ Auswahl mit Interrupt-Controller:
 - ▶ Ermöglicht Verwendung von Standard-Bausteinen
 - ▶ Priorität wird vom Controller festgelegt
 - Unterschiedliche Strategien: Feste Prioritäten, Round Robin
 - ▶ Controller ermöglicht preemptive Interruptbehandlung:
 - IRQ mit höherer Priorität können laufende ISR unterbrechen

Interrupt-verarbeitung

Mehrere Interruptquellen



- ▶ Preemptive Interruptbehandlung
 - ▶ Steuerung mit einem zusätzlichem Prioritätsregister
 - Register enthält Priorität des Interrupts, der gerade bearbeitet wird.
 - ▶ Controller blockiert alle Interrupts mit kleinerer oder gleicher Priorität.
 - ▶ Interrupts mit höherer Priorität werden weitergereicht.
 - ▶ Zum Zurücksetzen der Prio muss Controller Ende der ISR erkennen:
 - Meist mittels zusätzlichem Ausgabebefehls von CPU

Mehrere Interruptquellen



Volatile und globale Variable

- ▶ mit volatile werden Variable gekennzeichnet, deren Wert sich außerhalb des aktuellen Programmfpades ändern kann, z.B.:
 - ▶ in einer ISR.
 - ▶ Hardware-Register, z.B. Timer.
- ▶ Kennzeichnung verhindert eine übermäßige Optimierung, z.B.:
 - ▶ Entfernen einer solchen Variable aus einer Schleife.

Interrupt-Service-Routine:

```
volatile int counter;

void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    ....
    counter++;
    ....
}
```

Hauptschleife:

```
.....
while( 1 ){
    int temp = counter;
    printf( "Zähler ist: %d\n", counter );
    temp
}
.....
```

Wiedereintrittsfeste Bibliotheksfunktionen

Nicht wiedereintrittsfeste Bibliotheksfunktion:

```
char buf[100];
char* textUmkehr( char* str){
    int i;
    for( i=0,j=strlen(str)-1; str[i]!='\0'; i++,j-- ){
        buf[j] = str[i];
    }
    return buf;
}
```

Interrupt-Service-Routine:

```
volatile char* rev;
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    .....
    rev = textUmkehr("Heitmann");
    .....
}
```

Hauptschleife:

```
.....
while( 1 ){
    printf( "Reverse Text ist: %s\n", textUmkehr( "HAW" ) );
}
.....
```

Wiedereintrittsfeste Bibliotheksfunktionen

Wiedereintrittsfeste Bibliotheksfunktion:

```
void textUmkehr( char* str, char* ergebnis){  
    int i;  
    for( i=0,j=strlen(str)-1; str[i]!='\0'; i++,j-- ){  
        ergebnis[j] = str[i];  
    }  
}
```

Interrupt-Service-Routine:

```
volatile char rev[80];      //warum soll und darf dies keine lokale Variable sein?  
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {  
    ....  
    textUmkehr("Heitmann", rev);  
    ....  
}
```

Hauptschleife:

```
....  
while( 1 ){  
    char rev[80];  
    textUmkehr( "HAW", rev );  
    printf( "Reverse Text ist: %s\n", rev );  
}  
....
```

Wiedereintrittsfeste Bibliotheksfunktionen

- ▶ Bibliotheksfunktionen, die von der Hauptschleife und von der ISR aufgerufen werden, müssen wiedereintrittsfähig (reentrant) sein.
 - ▶ **printf** und **malloc** sind es nicht.
- ▶ **Kritisch:**
 - ▶ **Schreibzugriffe auf statische oder globale Variable.**
 - ▶ **Rückgabe von Adressen auf statische oder globale Variable.**
 - ▶ **Aufrufe von non-reentrant Funktionen.**
- ▶ **Bibliotheksfunktionen sollten mit**
 - ▶ **lokalen Variablen oder**
 - ▶ **Daten, die vom Aufrufer zur Verfügung gestellt werden****arbeiten.**

Synchronisation zwischen ISR und Hauptschleife

- ▶ Z.B. zu beachten bei „gleichzeitigen“ Schreibzugriffen von
 - ▶ ISR und
 - ▶ Hauptschleife
- auf eine globale Variable:

Interrupt-Service-Routine:

```
volatile int counter;  
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {  
    ....  
    counter++;  
    ....  
}
```

Hauptschleife:

```
....  
while( 1 ){  
    if( counter >= COUNTERMAX ){  
        <mach etwas>  
        counter = 0;  
    }  
    ....  
}
```

Synchronisation zwischen ISR und Hauptschleife

- ▶ Z.B. zu beachten bei „gleichzeitigen“ Schreibzugriffen von
 - ▶ ISR und
 - ▶ Hauptschleifeauf eine globale Variable.

- ▶ **Unkritisch, wenn der Zugriff „atomar“ erfolgt (er besteht aus einer einzigen Maschineninstruktion).**

```
//atomar:  
counter = 55;           //ok, wenn int und 32-Bit CPUs  
localvalue = counter;  
  
//nicht atomar:  
counter++;             //Bei RISC CPUs: Read-Modify-Write Zyklus  
counter |= (1<<bitnr);
```

Synchronisation zwischen ISR und Hauptschleife

- ▶ Wenn Zugriff nicht atomar, Synchronisationsmechanismen verwenden:
 - ▶ Bei Unterstützung durch ein Betriebssystem:
 - Semaphore
 - Mutex
 - Monitor
 - ▶ Ohne Betriebssystem:
 - Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren:
 - Nur notwendig in Hauptschleife.
 - ISR ist in der Regel nicht unterbrechbar.

Hauptschleife:

```
#include <armVIC.h>
.....
while( 1 ){
    disableIRQ();
    if( counter >= COUNTERMAX ){
        <mach etwas>
        counter = 0;
    }
    enableIRQ();
    .....
}
```

Synchronisation zwischen ISR und Hauptschleife

- ▶ Ohne Betriebssystem:
 - ▶ Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren.
 - ▶ Deaktivierung bedeutet Verlängerung der Latenzzeit, daher
 - Ausschaltzeit möglichst kurz halten!

Hauptschleife:

```
.....
while( 1 ){
    int merker = 0;
    disableIRQ();
    if( counter >= COUNTERMAX ){
        merker = 1;
        counter = 0;
    }
    enableIRQ();
    if( merker ){
        <mach etwas>
    }
    .....
}
```

Synchronisation zwischen ISR und Hauptschleife

- ▶ Ohne Betriebssystem:
 - ▶ Für die Dauer des Zugriffs Interruptverarbeitung deaktivieren.
 - ▶ Bei Verwendung in Bibliotheksfunktionen aufpassen, dass Interrupt nicht versehentlich eingeschaltet wird.

Hauptschleife:

```
.....
while( 1 ){
    int merker = 0;
    int oldirq = disableIRQ();
    if( counter >= COUNTERMAX ){
        merker = 1;
        counter = 0;
    }
    restoreIRQ(oldirq);
    if( merker ){
        <mach etwas>
    }
    ....
}
```

Interrupt- verarbeitung

Ein- und Ausschalten von Interrupts in der Hitex Laufzeitumgebung

- ▶ Funktionen in armVIC.h:

```
unsigned disableIRQ(void);  
  
unsigned enableIRQ(void);  
  
unsigned restoreIRQ(unsigned oldCPSR);  
  
unsigned disableFIQ(void);  
  
unsigned enableFIQ(void);  
  
unsigned restoreFIQ(unsigned oldCPSR);
```

System mit kurzen Reaktionszeiten

- ▶ **Latenzzeit:**

Zeit, die zwischen dem Auftreten eines Ereignisses und bis zur Bearbeitung des Ereignisses vergeht.

- ▶ Oft gefordert: Latenzzeit soll vorgegebene Grenze nicht überschreiten:
 - ▶ **Beispiel: Serielle Schnittstelle.**
 - ▶ Bei nicht unterbrechbaren ISRs ergibt sich die Latenzzeit des Systems aus der Bearbeitungsdauer der langsamsten ISR.
 - ▶ Interruptroutinen müssen daher möglichst schnell abgearbeitet werden.
 - ▶ **In ISRs sollte nicht gewartet werden, z.B.:**
 - Warten auf Timer.
 - Warten auf Fertig-Bit einer Hardware-Komponente.
 - Eingabeaufforderung an Benutzer.
 - ▶ **Besonders wichtig bei vielen Interruptquellen.**

Computer Engineering WS 2012

Interruptverarbeitung im LPC 2468

HTM – SHF - SWR

Ausnahme- und Interruptverarbeitung im LPC 2468

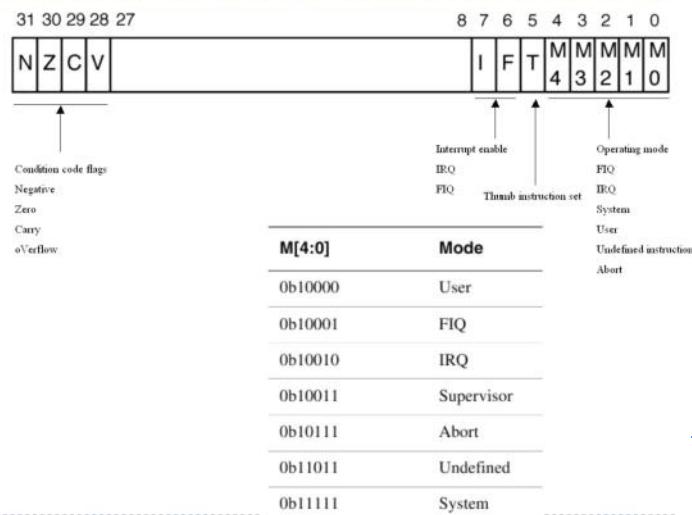
Betriebsarten, Statuswort

- ▶ ARM 7 TDMI unterscheidet 7 Betriebsarten.
- ▶ **User Mode** ist die normale Betriebsart.
- ▶ Wechsel in andere Betriebsarten durch **Exceptions (Ausnahmen)**.
- ▶ Momentane Betriebsart steht im „**Current Program Status Register**“ (**CPSR**)
- ▶ Betriebsarten haben:
 - ▶ **eigene Stacks**
 - ▶ **eigene Register** (z.B. **LR** und **SP**) (siehe nächste Folie)
 - ▶ „**Saved Program Status Register**“ (**SPSR**)
(beim Wechsel der Betriebsart wird hier das **CPSR** gespeichert)
 - ▶ **unterschiedliche Rechte bei Zugriff auf Systemregister**
(z.B. im User Mode kann nicht der SP oder das SPSR verändert werden)

Ausnahme- und Interruptverarbeitung im LPC 2468

Betriebsarten

Statuswort



3

Ausnahme- und Interruptverarbeitung im LPC 2468

Betriebsarten

Statuswort

HITOP:

Register werden angezeigt mit

View→

SFR Window →
ARM Processor Register

System & User	FIQ	Supervisor	Abort	IRQ	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8	R8	R8	R8	R8
R9	R9	R9	R9	R9	R9
R10	R10	R10	R10	R10	R10
R11	R11	R11	R11	R11	R11
R12	R12	R12	R12	R12	R12
R13	R13_fiq	R13_svc	R13_abt	R13_irq	R13_und
R14	R14_fiq	R14_svc	R14_abt	R14_irq	R14_und
R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)	R15 (PC)



4

Ausnahme- und Interruptverarbeitung im LPC 2468

Betriebsarten, Statuswort

- Wie kann man auf R8 der Betriebsart FIQ zugreifen?

MRS R₁, CPSR

1. Momentane Betriebsart in R1 merken.

MSR CPSR, #06 10 001

2. Auf Betriebsart FIQ umschalten

MoV R₀, R₈

3. Registerinhalt von R8 nach R0 schreiben

MSR CPSR, R₁

4. Zurückschalten auf ursprüngliche Betriebsart

5

Ausnahme- und Interruptverarbeitung im LPC 2468

Ausnahmen

- Wenn eine Ausnahme auftritt:
 - CPU rettet CPSR nach SPSR und ändert Betriebsart
 - PC wird auf Eintrag in Vektortabelle gesetzt:

*To be filled if
Instructions*

Exception	Mode	Address
Reset	Supervisor	0x00000000
Undefined Instruction	Undefined	0x00000004
Software Interrupt (SWI)	Supervisor	0x00000008
Prefetch Abort (instruction fetch memory abort)	Abort	0x0000000C
Data Abort (data access memory abort)	Abort	0x00000010
IRQ (interrupt)	IRQ	0x00000018
FIQ (fast interrupt)	FIQ	0x0000001C

6

Ausnahmen

- ▶ Rückkehr von Ausnahme:
 - ▶ CPU hat dafür (im Gegensatz zu vielen anderen) keinen speziellen Befehl.
 - ▶ Daher müssen PC und CPSR mit regulären ARM-Assembler Befehlen geändert werden.

Exception	Mode	Return Statement
Reset	Supervisor	-
Undefined Instruction	Undefined	SUBS PC,LR,#4
Software Interrupt (SWI)	Supervisor	MOVS PC,LR
Prefetch Abort (instruction fetch memory abort)	Abort	SUBS PC,LR,#4
Data Abort (data access memory abort)	Abort	SUBS PC,LR,#8
IRQ (interrupt)	IRQ	SUBS PC,LR,#4
FIQ (fast interrupt)	FIQ	SUBS PC,LR,#4

7

*CPSR → CPSR
damit auch Einschaltung der Betriebsart*

Ausnahmeroutinen

- ▶ HITOP Entwicklungsumgebung:
startup.s
 - definiert Standard-Ausnahmeroutinen, trägt die Aufrufe in die Vektortabelle ein und
 - richtet die Stacks für die Betriebsarten ein.

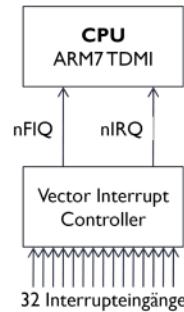
Exception	Mode	Behandlung
Reset	Supervisor	Startup-Code, Aufruf von main
Undefined Instruction	Undefined	Endlosschleife
Software Interrupt (SWI)	Supervisor	Rahmen für eigene Routinen
Prefetch Abort (instruction fetch memory abort)	Abort	Endlosschleife
Data Abort (data access memory abort)	Abort	Endlosschleife
IRQ (interrupt)	IRQ	spezielle Behandlung
FIQ (fast interrupt)	FIQ	Endlosschleife

8

Ausnahme- und Interruptverarbeitung im LPC 2468

Interruptverarbeitung

- ▶ ARM-CPU:
- ▶ 2 Interruptrequest-Eingänge: Fast Interrupt (FIQ) und Interrupt (IRQ)
- ▶ LPC2468:
 - ▶ Zusätzlich „**Vector Interrupt Controller**“ (VIC)
 - ▶ Jede Interruptquelle des Chips ist fest mit einem bestimmten Eingang des VIC verbunden.
 - ▶ Per Software kann jede Interruptquelle auf FIQ oder **vectored IRQ** eingestellt werden.
 - ▶ Idealerweise sollte nur **eine** Interruptquelle auf FIQ gestellt werden.
 - ▶ Die restlichen Quellen benutzen vectored IRQ. Hierfür kann die Reihenfolge der Abarbeitung mittels **16 Prioritätsstufen** festgelegt werden.



9

Ausnahme- und Interruptverarbeitung im LPC 2468

Interruptquellen des LPC 2468

Bit	31	30	29	28	27	26	25	24
Symbol	I2S	I2C2	UART3	UART2	TIMER3	TIMER2	GPDMA	SD/MMC
Bit	23	22	21	20	19	18	17	16
Symbol	CAN1&2	USB	Ethernet	BOD	I2C1	AD0	EINT3	EINT2/ LCD ⁽¹⁾
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SSP1	SPI/SSP0	I2C0	PWM0&1
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

10

FIQ

- Aktionen zum Eintreten in die ISR
 - 1. Speichern der Adresse der nächsten Instruktion (**PC**) in **LR**
 - 2. Speichern von **CPSR** in **SPSR**
 - 3. Setzen der Betriebsart **FIQ** in **CPSR**, Registersatz wird umgeschaltet
 - 4. Setzen des F-Bits in **CPSR (FIQ aus)**
 - 5. Setzen des **PC** auf **FIQ-Adresse der Vektortabelle**
- Maximale Dauer bis Start ISR (Interruptlatenzzeit) setzt sich zusammen:
 - 1. Synchronisation des externen Signals max 4 Takte
 - 2. Abarbeiten der aktuellen Instruktion:
Längste Instruktion ist LDM mit allen Registern max 20 Takte
 - 3. Möglicher Weise findet gerade eine **Data Abort Exception** statt (hat höhere Priorität): 3 Takte
 - 4. **FIQ-Aktivierung** 2 Takte

Insgesamt: 29 Takte, bei 48 MHz also ca. 0.6 µsec

FIQ

- Aktionen zum Verlassen der ISR
 - 1. Speichern des **LR**, verkleinert um 4, in den **PC**.
 - 2. Speichern von **SPSR** in das **CPSR**.
Damit automatisch:
 - Umschalten der Betriebsart und
 - Reaktivierung des FIQs.
- Beide Aktionen lassen sich wie folgt durchführen:

SUBS PC, LR, #4 @ ISR verlassen

Basiert auf Sonderfunktion:

Wenn S bei Datenmanipulationsbefehlen gesetzt und
das Zielregister PC ist,
dann wird SPSR nach CPSR kopiert!

FIQ-ISR: Eigenschaften

- ▶ CPU: schaltet Register R8 bis R14 um, können also von ISR beliebig verändert werden.
rettet Statusregister und Rücksprungadresse.
- ▶ Änderungen an R0 bis R7 müssen am Ende der ISR durch die ISR rückgängig gemacht werden!
- ▶ Vor Aufruf eines Unterprogramms muss LR gerettet werden!
- ▶ Unterprogramme, die vom Hauptprogramm und von der ISR aufgerufen werden, müssen wiedereintrittsfest (**reentrant**) sein:
Eine Funktion ist reentrant, wenn sie mehrmals gleichzeitig aktiv sein kann, ohne dass sich diese Aufrufe gegenseitig beeinflussen.
- ▶ Mehrfachaufrufe passieren häufig bei Bibliotheksfunktionen (`strcpy`).
- ▶ `printf` und `malloc` sind meist nicht reentrant!

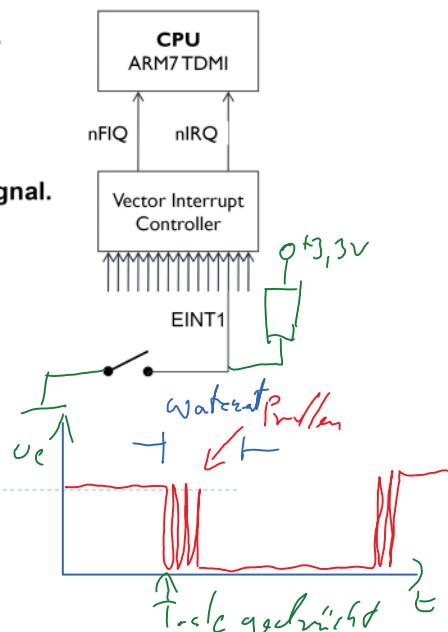
13

CE WS12

Beispiel: Auslösung eines FIQ durch eine Taste

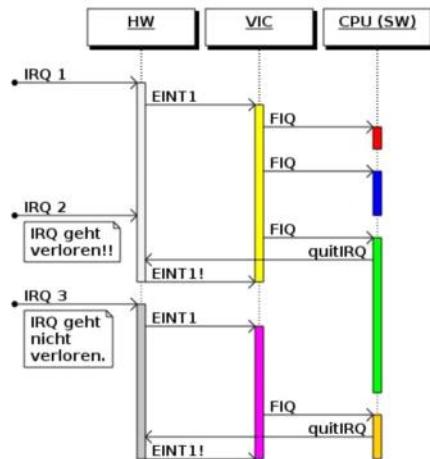
- ▶ FIQ-Serviceroutine : Zählen der Tastenbetätigungen.
- ▶ Verwendung des „Externen Interrupt 1“ (EINT1)
- ▶ Auslösung von EINT1 durch externes digitales Signal.
- ▶ EINT1 kann eingestellt werden auf:
 - Pegelsensitiv, Auslösung bei High-Pegel
 - Pegelsensitiv, Auslösung bei Low-Pegel
 - Flankensensitiv, Auslösung bei steigender Flanke
 - Flankensensitiv, Auslösung bei fallender Flanke

14



Beispiel „Zählen von externen Ereignissen“: Ablauf

- ▶ Interrupts müssen in der Regel quittiert werden.
- ▶ Ohne Quittierung: Nach Beendigung der ISR wird sie sofort erneut aktiviert!
 - System ist dauerhaft blockiert
- ▶ Interrupts können verlorengehen
- ▶ daher: Quittierung des Interrupts möglichst früh durchführen!



Beispiel „Zählen von externen Ereignissen“: ISR

1. Interrupt muss quittiert werden
 - ▶ R9 enthält die Adresse des Ports EXTINT
 - ▶ R10 die notwendige Maske

Beides wurde während der Initialisierung in die FIQ-Register geladen
2. Zähler erhöhen, R8 enthält aktuellen Zählerwert.
3. Interruptserviceroutine beenden.

```

fiq:  STR    R10, [R9]      @ Interrupt quittieren
      ADD    R8, R8, #1      @ Zaehler erhöhen
      SUBS   PC, LR, #4      @ ISR verlassen
  
```

Beispiel „Zählen von externen Ereignissen“: **Initialisierung der ISR**

```
.section .text
.global fiq,fiqinitasm

.equ EXTINT,0xE01FC140

fiqinitasm:                                @ Funktion kann nur in Betriebsart
                                              @ Supervisor aufgerufen werden!

    MRS    R0, CPSR      @ aktuelle Betriebsart retten
    MSR    CPSR_c,#0xD1  @ auf FIQ-Betriebsart umschalten
                                              @ FIQ und IRQ aus

    MOV    R8,#0          @ Zaehler loeschen
    LDR    R9,=EXTINT     @ Adr von EXTINT laden
    MOV    R10,#1<<1      @ Maske fuer EINT1 laden

    BIC    R0,#1<<6      @ FIQ aktivieren: FIQ-Bit im CPSR loeschen
    MSR    CPSR_c,R0      @ auf urspraelige Betriebsart schalten

    MOV    PC,LR          @ Unterprogramm verlassen
```

17

CE WS12

C-Programm: Auslösung eines FIQ durch eine Taste

```
extern void fiqinitasm(void);

void fiqinit(void){

    PINSEL4 |= 1<<22;           //P2.11 is EINT1

    EXTMODE    = 1<<1;          //edge sensivity
    EXTPOLAR   = 0;              //Falling edge

    VICIntSelect |= 1<<15;       //EINT1 is FIQ
    VICIntEnable |= 1<<15;       //Enable EINT1

    fiqinitasm();                //Restliche Initialisierung
                                  //mit Assemblerprogramm
}
```

Externer
Interrupt

Vector Interrupt
Controller

18

CE WS12

ISR in C programmiert

- Nicht im C-Standard vorgesehen!
- Implementierung abhängig von
 - CPU-Hersteller
 - Compiler-Hersteller
- Nachfolgend: Implementierung von ISR mittels GNU C-Compiler (gcc) für ARM-CPU's.
- Im gcc können Funktionen zusätzliche Attribute erhalten:

```
void f () __attribute__ ((interrupt ("FIQ")));
void f () __attribute__ ((interrupt ("IRQ")));
```
- Compiler rettet alle notwendigen Register und restauriert sie am Ende der ISR.
- Return am Ende der Funktion wird ersetzt durch „Return from Interrupt“

19

CE WS12

Link auf gcc-Dokumentation:
<http://gcc.gnu.org/onlinedocs/gcc/Function-Attributes.html>

Beispiel „Zählen von externen Ereignissen“: **C-Version der ISR**

- Übersetzt mit -O2: Hohe Optimierung. Im Praktikum: -O0

C-Code

```
void __attribute__ ((interrupt("FIQ"))) cfiq(void) {
    counter++;
    EXTINT = 1<<1;
}
```

Quittierung des Interrupts

Erzeugter Assemblercode

```
stmdfd sp!, {r1, r2, r3}
ldr r2, .L3
ldr r3, [r2, #0]
add r3, r3, #1
str r3, [r2, #0]

mov r3, #-536870912
add r3, r3, #2080768
mov r1, #2
add r3, r3, #256
strb r1, [r3, #64]

ldmfd sp!, {r1, r2, r3}
subs pc, lr, #4
```

20

CE WS12

Beispiel „Zählen von externen Ereignissen“: Eintrag in Vektortabelle

- Hitop: Vektortabelle ist in start.s definiert

```
_startup:
# -----
# Interrupt vector table at address 0
# -----
Reset_Vec:    LDR      PC, _ResetEntry
              LDR      PC, _Undef_Addr
              LDR      PC, _SWI_Addr
              LDR      PC, _PAbt_Addr
              LDR      PC, _DAbt_Addr
              .word   CheckSum      /* Reserved Vector */
              LDR      PC, [PC,#-0x120]
              LDR      PC, _FIQ_Addr    /* Calling the FIQ handler */

# -----
_ResetEntry:  .word   ResetEntry
_Undef_Addr:  .word   Undef_Handler
_SWI_Addr:   .word   SWI_Handler
_PAbt_Addr:  .word   PAbt_Handler
_DAbt_Addr:  .word   DAbt_Handler
_FIQ_Addr:   .word   fiq
```

Hier Startadresse der ISR eintragen

Beispiel „Zählen von externen Ereignissen“: Stack definieren

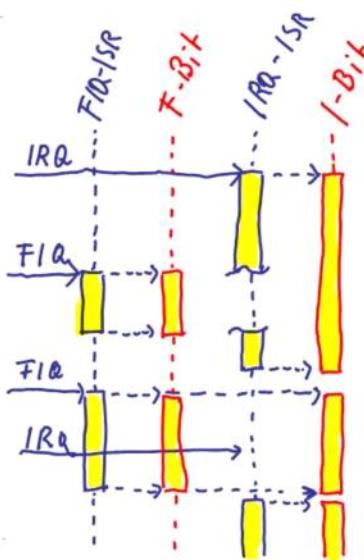
- Hitop: Stack ist in start.s definiert
- Standardeinstellung für FIQ-Stacklänge: nur 8 Worte!

```
# Stack definitions
# size in words!
.equ  UND_Stack_Size , 8
.equ  SVC_Stack_Size , 1024
.equ  ABT_Stack_Size , 8
.equ  FIQ_Stack_Size , 8
.equ  IRQ_Stack_Size , 512
```

Hier ausreichende Stacklänge eintragen

Vectored IRQ

- IRQ verhält sich innerhalb der CPU wie FIQ, hat aber eine geringere Priorität.
- Priorität entscheidet, welcher Interrupt als nächstes bearbeitet wird, wenn beide Interrupts gleichzeitig auftreten.
- Unterbrechbarkeit wird gesteuert durch Setzen und Löschen des I- und des F-Bits im Statusregister (siehe Ablaufdiagramm).
 - Aktivieren von IRQ setzt nur I-Bit.
 - Aktivieren von FIQ setzt I- und F-Bit.
- Damit kann FIQ eine laufende IRQ-ISR unterbrechen.
- Aber IRQ kann eine FIQ-ISR nicht unterbrechen!



23

CE WS12

Vectored IRQ: Zuordnung der ISR

- VIC verwaltet für jeden Interrupeingang:
 - Startadresse der ISR
 - Priorität des Interrupts
- Aktivierung eines Interrupeingangs:
 - VIC kopiert zugeordnete Startadresse in das Register **VICVectAddr**.
 - VIC setzt internes Prioritätsregister auf Priorität des Interrupts.
 - VIC aktiviert IRQ Eingang der CPU.
- Ausnahmeverarbeitung der CPU liest **VICVectAddr** (Adr 0xFFFF FF00) und verzweigt direkt in die ISR.

Vektortabelle	0000	LDR	PC, _ResetEntry	Lesen von VICVectAddr und Sprung in die ISR
	0004	LDR	PC, _Undef_Addr	
	0008	LDR	PC, _SWI_Addr	
	000C	LDR	PC, _PAbt_Addr	
	0010	LDR	PC, _DAbt_Addr	
	0014	.word	Checksum	
	0018	LDR	PC, [PC, #-0x120]	
	001C	LDR	PC, _FIQ_Addr	
			/* Reserved Vector */	
			/* Calling the IRQ handler */	

24

CE WS12

→ Auslesen von VIC Vect tabler

Vectored IRQ: Prioritäten

- ▶ Nach Auslösen des IRQs:
 - ▶ Internes Register des VIC enthält **Priorität des aktuellen Interrupts**.
- ▶ Nachfolgende Interrupts mit **kleinerer** oder **gleicher** Priorität:
 - ▶ Werden von VIC nicht bearbeitet.
- ▶ Nachfolgende Interrupts mit **höherer** Priorität:
 - ▶ VIC aktualisiert sofort **VICVectAddr** und
 - ▶ aktiviert IRQ-Eingang der CPU
- ▶ Ermöglicht die Implementierung von **unterbrechbaren (preemptiven) ISRs!**
- ▶ Erfordert allerdings zusätzlichen Programmieraufwand:
 - ▶ Frühzeitige Freigabe des IRQs.
 - ▶ Zusätzliche Verwaltungsmaßnahmen.

25

CE WS12

Vectored IRQ: Nicht unterbrechbare ISR

- ▶ Erfordert zwei wesentliche Aktivitäten:
 1. **Quittierung des Interrupts beim Interrupt auslösenden Gerät:**
 - Gerät deaktiviert IRQ.
 2. **Quittierung des Interrupts beim VIC:**
 - VIC setzt internes Prioritätenregister zurück.
Sollte erst am Ende der ISR erfolgen.
- ▶ **IRQ ist während der gesamten Abarbeitung gesperrt (I-Bit gesetzt):**
 - ▶ ISR kann nicht unterbrochen werden (bzw. nur durch FIQ).

```
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {  
  
    EXTINT = 1<<1; /* Interrupt bestätigen: */  
                /* Device deaktiviert IRQ-Leitung */  
    counter++;  
  
    VICVectAddr = 0; /* VIC mitteilen: ISR ist fertig */  
                    /* VIC setzt internes Prio Register */  
                    /* zurück */  
}
```

26

CE WS12

Vectored IRQ: Initialisierung von nicht unterbrechbaren ISRs

```

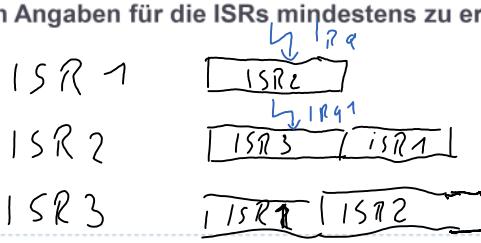
void irqinit(void){
    PINSEL4      |= 1<<24;           // P2.12 is EINT2
    EXTMODE       = 1<<2;            // edge sensitivity
    EXTPOLAR     = 0;                // Falling edge

    VICIntSelect  &= ~(1<<16);        // EINT2 is IRQ
    VICVectAddr16  = (int)little_isr; // EINT2 fires little_isr
    VICVectPriority16 = 5;           // EINT2 gets Prio 5
    VICIntEnable   |= 1<<16;          // Enable EINT2
}

```

Vectored IRQ: Nichtunterbrechbare ISR, Latenzzeiten

- ▶ Es wurden 3 ISR implementiert:
 - ▶ ISR1 hat die höchste Priorität und dauert 200 µs,
 - ▶ ISR2 hat eine mittlere Priorität und dauert 400 µs und
 - ▶ ISR3 hat die niedrigste Priorität und dauert 300 µs.
- ▶ Jede Interruptquelle feuert maximal einmal pro 1 msec.
- ▶ Welche Interrupt-Latenzzeiten sind im schlimmsten Fall auf Grund der obigen Angaben für die ISRs mindestens zu erwarten?



$$t_L = 400 \mu s$$

$$t_L = 300 \mu s + 200 \mu s = 500 \mu s$$

$$t_L = 200 \mu s + 400 \mu s = 600 \mu s$$

Vectored IRQ: Preemptive ISR

- Erfordert zwei zusätzliche Aktivitäten:
 1. Nach Quittierung des Interrupts:
 - Löschen des I-Bits (Freigabe des IRQs).
 - Retten von LR und SPSR (nicht trivial)
 2. Vor Quittierung des Interrupts beim VIC:
 - Sperren des IRQs
 - Zurückspeichern von LR und SPSR

```
void __attribute__ ((interrupt("IRQ"))) little_isr(void) {
    EXTINT = 1<<1;                                /* Interrupt bestätigen:          */
                                                       /* Device deaktiviert IRQ-Leitung */
    ENABLE_NESTED_ISR;
    .....
    .....
    DISABLE_NESTED_ISR;

    VICVectAddr = 0;                                /* VIC mitteilen: ISR ist fertig */
}

```

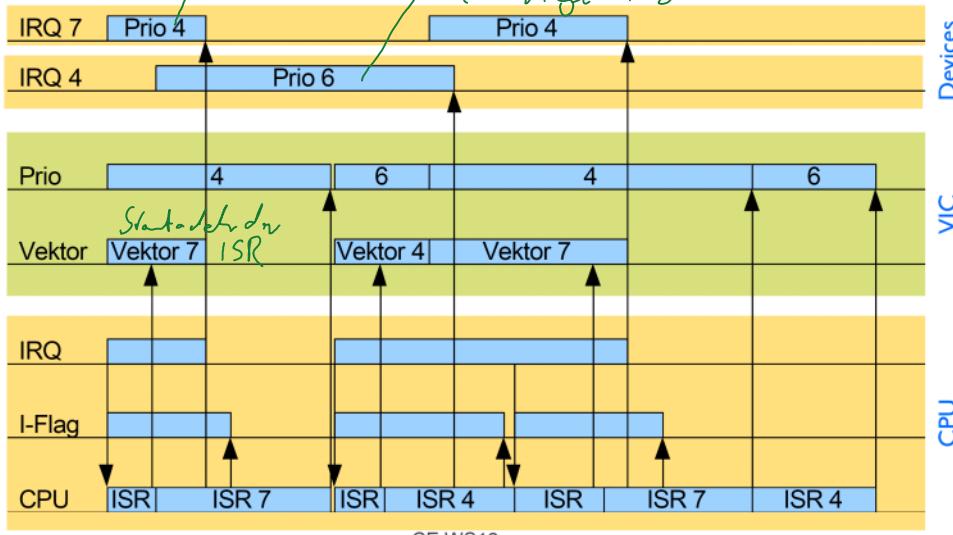
29

CE WS12

Vectored IRQ: Preemptive ISR

hohe Prio

mixt mit geringe Prio



30

CE WS12

Ausnahme- und Interruptverarbeitung im LPC 2468

Vectored IRQ: Unterbrechbare ISR

- ▶ Retten von LR funktioniert nur mit Umschaltung in eine andere Betriebsart!
- ▶ Ohne Umschaltung geht aktuelles LR beim erneuten Interrupt verloren.
- ▶ Im Beispiel: Rumpf der ISR wird in Supervisor-Mode abgearbeitet.
- ▶ Achtung: Verwendung des Supervisorstacks!

```
#define ENABLE_NESTED_ISR
asm volatile(" mrs   r1, spsr\n"
            " stmfd sp!,{r1,lr}\n"           /*SPSR_irq und LR_irq retten*/ \
            " msr   cpsr_c, #0x53\n"        /*Umschalten nach SVC-Mode */ \
            " stmfd sp!,{lr}\n" : : : "r1" ); /*LR_svc retten*/ \
```

```
#define DISABLE_NESTED_ISR
asm volatile(" ldmfd sp!,{lr}\n"           /*LR_svc restaurieren*/ \
            " msr   cpsr_c, #0xd2\n"       /*Umschalten auf IRQ-Mode*/ \
            " ldmfd sp!,{r1,lr}\n"         /*SPSR und LR wieder herstellen*/ \
            " msr   spsr_fc,r1\n" : : : "r1" );
```

Ausnahme- und Interruptverarbeitung im LPC 2468

Vectored IRQ: Unterbrechbare ISR, Latenzen

- ▶ Es wurden 3 ISR implementiert:
 - ▶ ISR1 hat die höchste Priorität und dauert 200 µs,
 - ▶ ISR2 hat eine mittlere Priorität und dauert 400 µs und
 - ▶ ISR3 hat die niedrigste Priorität und dauert 300 µs.
- ▶ Jede Interruptquelle feuert maximal einmal pro 1 msec.
- ▶ Welche Interrupt-Latenzzeiten sind im schlimmsten Fall auf Grund der obigen Angaben für die ISRs mindestens zu erwarten?

$$\begin{aligned}ISR1 &= 0\mu s \\ISR2 &\geq 200\mu s \\ISR3 &\geq 600\mu s\end{aligned}$$

Ein- und Ausschalten von Interrupts in der Hitex Laufzeitumgebung

► Funktionen in armVIC.h:

- ▶ **unsigned disableIRQ(void);**
- ▶ **unsigned enableIRQ(void);**
- ▶ **unsigned restoreIRQ(unsigned oldCPSR);**

- ▶ **unsigned disableFIQ(void);**
- ▶ **unsigned enableFIQ(void);**
- ▶ **unsigned restoreFIQ(unsigned oldCPSR);**

Computer Engineering WS 2012

CPU Ein- und Ausgabe

HTM – SHF - SWR

CPU

Ein- und Ausgabe

Übersicht

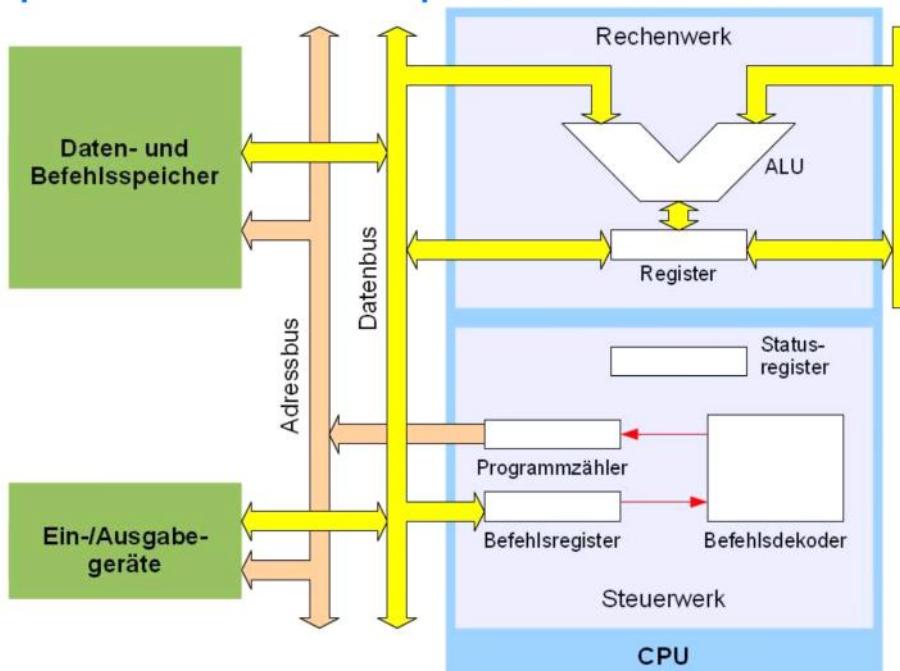


- ▶ Ein- und Ausgabe
 - ▶ Memory-Mapped I/O
 - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.

2

Ein- und Ausgabe

Prinzipieller Aufbau eines Computers



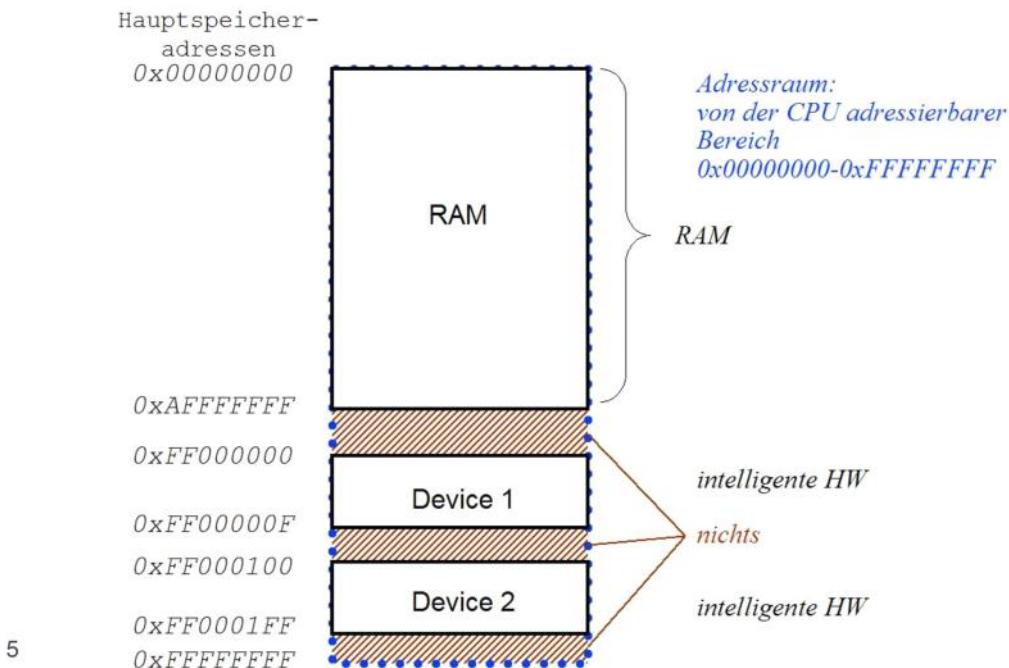
3

Wie greift man auf Register eines externen Gerätes zu?

- ▶ Alternative 1: Memory Mapped
 - ▶ Die Register sind auf Hauptspeicheradressen abgebildet (mapped).
 - ▶ Ein lesender / schreibender Zugriff auf diese Hauptspeicheradressen greift nicht auf den Speicher zu, sondern auf die entsprechenden Register des Devices.
- ▶ Alternative 2: I/O Mapped (muss die CPU unterstützen, Intel tut dies)
 - ▶ Es gibt einen weiteren Adressraum, so genannte I/O Adressen. Diese Adressen stehen in keiner Relation zu den Hauptspeicheradressen.
 - ▶ Über spezielle Befehle (in, out Assembler Befehle) wird über I/O Adressen auf die Register eines Devices zugegriffen.

4

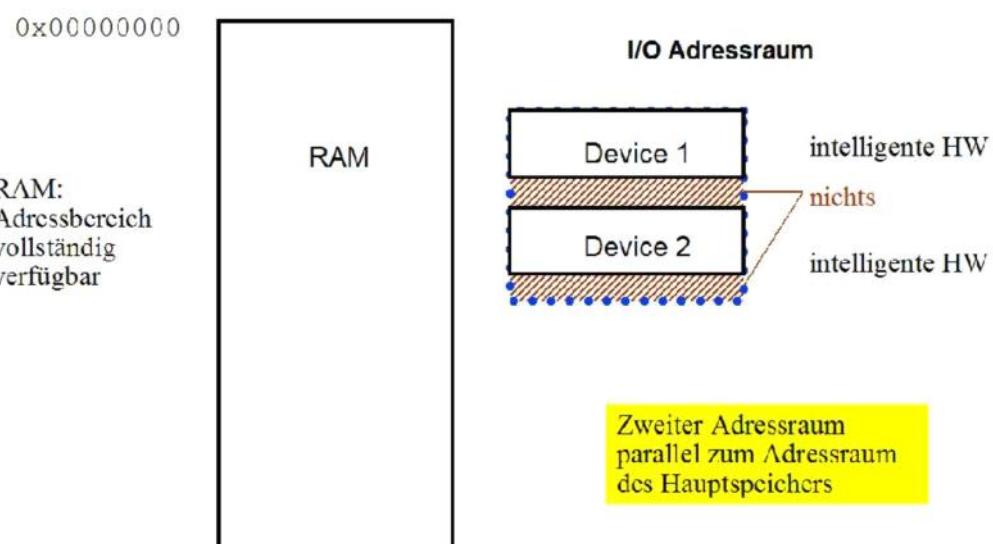
Memory-Mapped



5

Ein- und Ausgabe

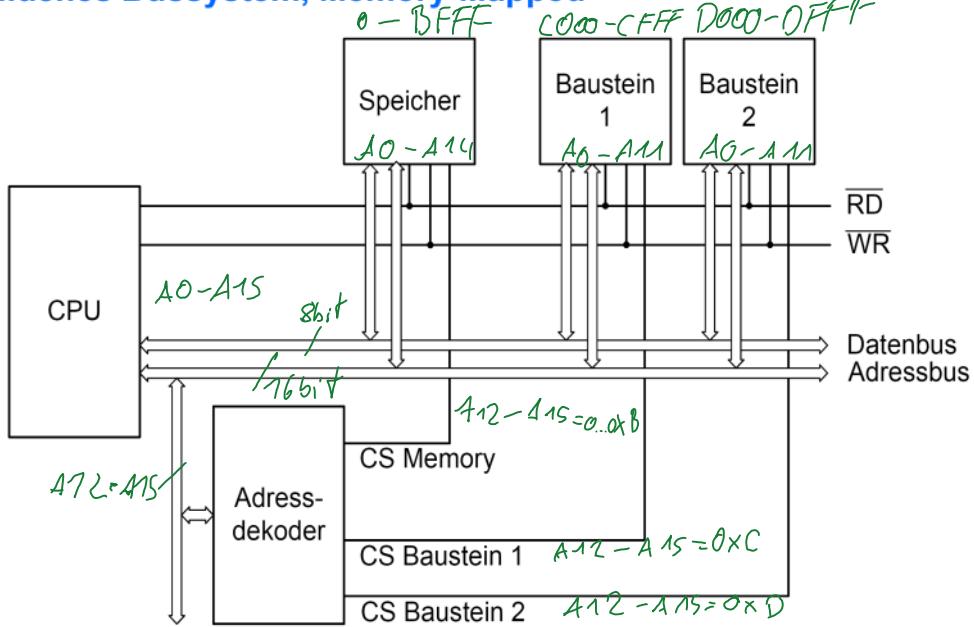
IO-Mapped



6

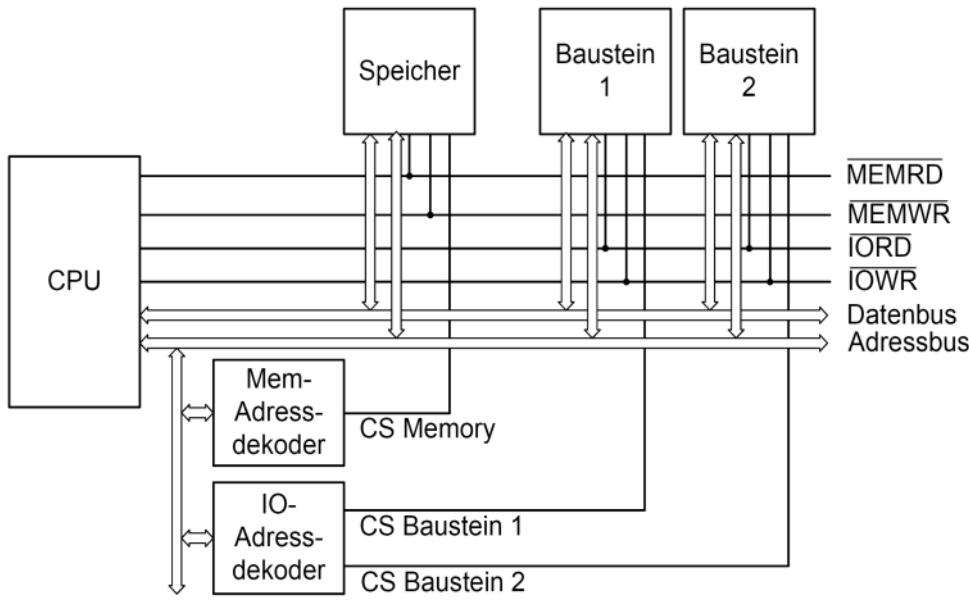
Ein- und Ausgabe

Einfaches Bussystem, Memory-Mapped



7

Einfaches Bussystem, IO-Mapped



8

Merkmale Bussysteme: Maximale Übertragungsrate, Einheit?

- ▶ hängt ab von:
 - ▶ Anzahl der gleichzeitig übertragbaren Bytes
 - gegeben durch Datenbusbreite
 - ▶ Bustaktfrequenz
 - maximale Frequenz, mit der die Signalleitungen eines Busses betrieben werden können
 - ▶ Anzahl der für die Übertragung notwendigen Bustakte

MB/sec

8, 16, 32, 64, 128

9

Merkmale Bussysteme: Buszyklenarten

- ▶ Einzelzyklusbus (single cycle)
 - ▶ Führt nur eine Datenübertragung durch
- ▶ Blockzyklus (burst cycle)
 - ▶ mehrfache aufeinander folgende Zugriffe
 - ▶ Adresse wird nur einmal übertragen

10

Merkale Bussysteme: Bustypen

- ▶ Split-Bus
 - ▶ Getrennte Adress- und Datenleitungen
 - ▶ Gleichzeitige Übertragung von Adressen und Daten möglich.
- ▶ Gemultiplexer Bus
 - ▶ Adressen und Daten werden über die gleichen Leitungen übertragen.
 - ▶ Während eines Speicherzugriffs werden zuerst die Adressen, dann die Daten übertragen.
 - ▶ Nachteil: Langsamer.
 - ▶ Vorteil: Geringerer Verdrahtungsaufwand, geringere Pin-Zahl.

11

Übersicht

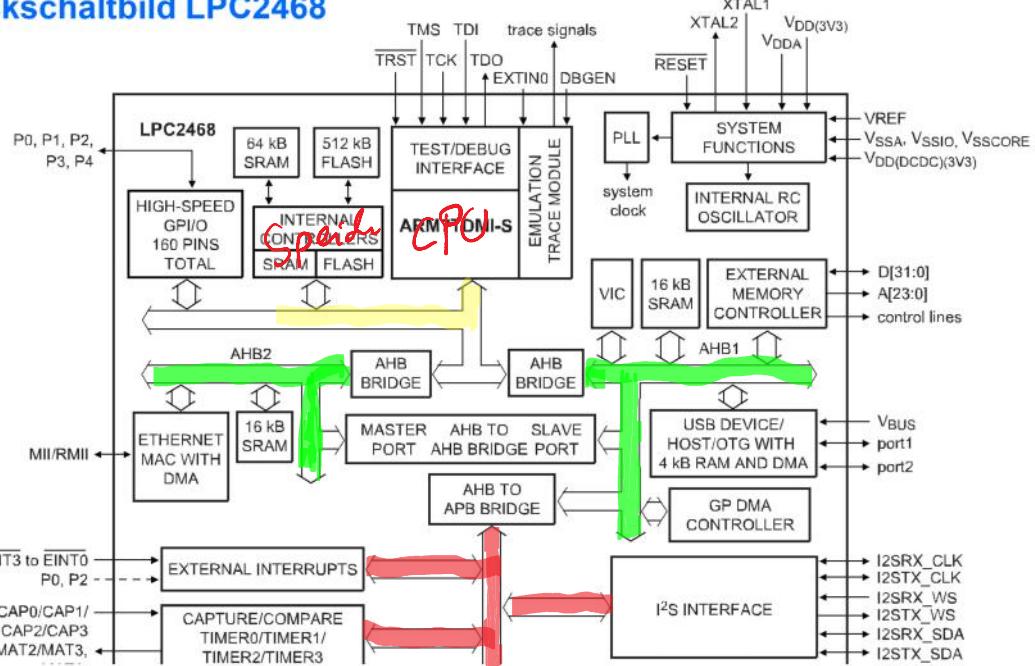
- ▶ Ein- und Ausgabe
 - ▶ Memory-Mapped I/O
 - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.

12

LPC2468

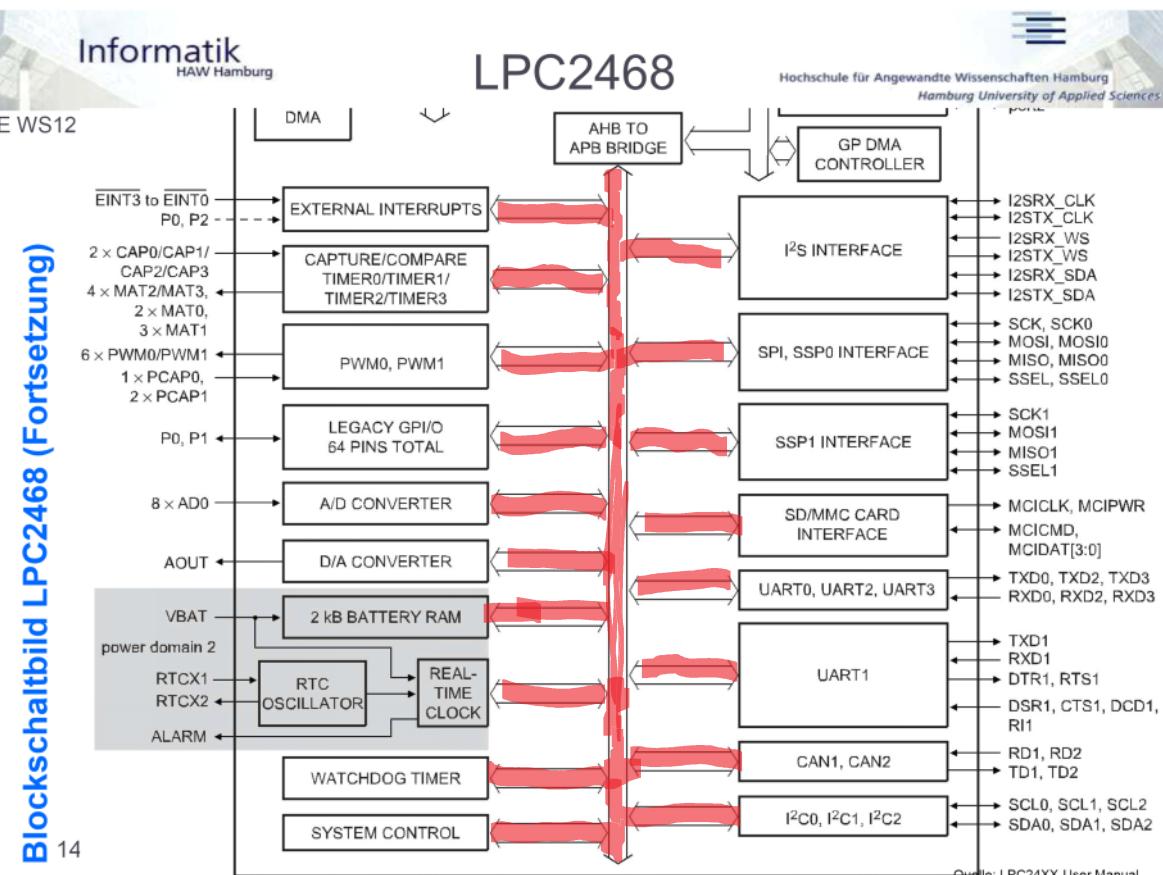
Blockschaltbild LPC2468

AMBA
ARM



13

Quelle: LPC24XX User Manual

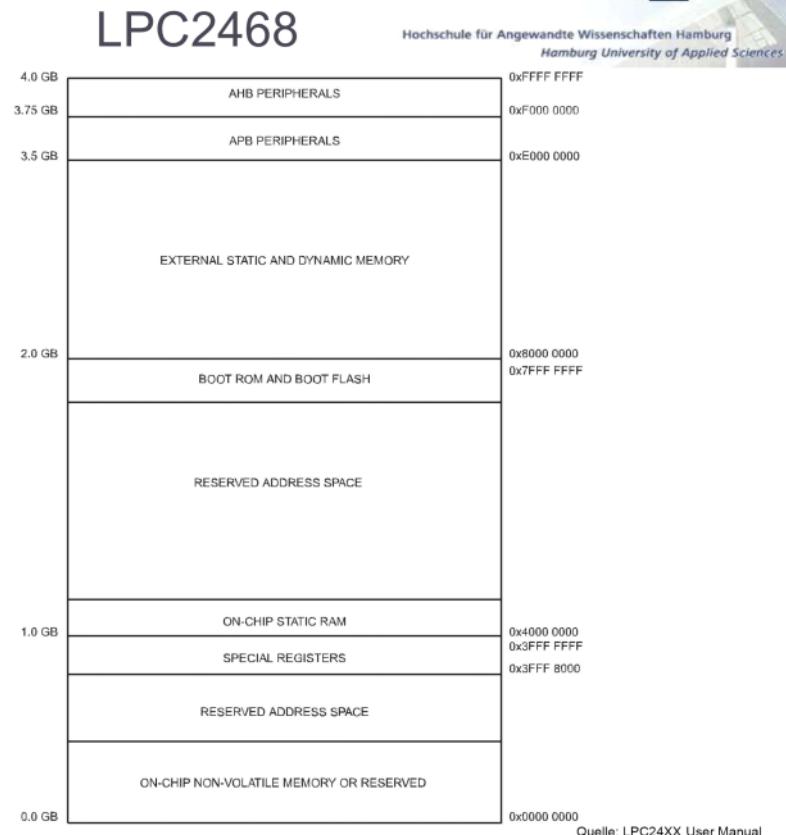


Bushierarchie

- ▶ LPC2468 enthält 3 Hierachieebenen:

1. **ARM 7 local bus**
 - High Speed
 - Anschluss von On-chip-Speicher (Flash, RAM)
 - Single Master: CPU
2. „**Advanced High-performance BUS“ (AHB)**
 - Multimasterfähig (z.B.: Bridge, USB, Ethernet)
 - burst transfers
 - single clock operation
3. „**Advanced Peripheral Bus“ (APB)**
 - Low Speed
 - Single Master: Bridge
 - Sehr einfacher Steuerbus

Speicherbelegung



Adressaufteilung APB (Ausschnitt)

- ## ► Jeder Baustein belegt 16 KBytes

APB Peripheral	Base Address	Peripheral Name
0	0xE000 0000	Watchdog Timer
1	0xE000 4000	Timer 0
2	0xE000 8000	Timer 1
3	0xE000 C000	UART0
4	0xE001 0000	UART1
5	0xE001 4000	PWM0
6	0xE001 8000	PWM1
7	0xE001 C000	I ² C0
8	0xE002 0000	SPI
9	0xE002 4000	RTC
10	0xE002 8000	GPIO
11	0xE002 C000	Pin Connect Block
12	0xE003 0000	SSP1
13	0xE003 4000	ADC
14	0xE003 8000	CAN Acceptance Filter RAM
15	0xE003 C000	CAN Acceptance Filter Registers
16	0xE004 0000	CAN Common Registers
17	0xE004 4000	CAN Controller 1
18	0xE004 8000	CAN Controller 2
19 to 22	0xE004 C000 to 0xE005 8000	Not used
23	0xE005 C000	I ² C1

CPU

Ein- und Ausgabe

Übersicht

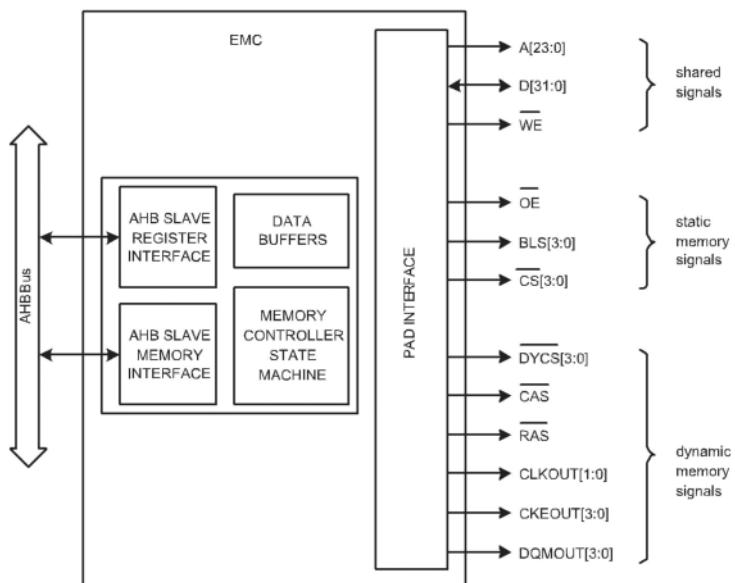
- ▶ Ein- und Ausgabe
 - ▶ Memory-Mapped I/O
 - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.

18

LPC2468

Externe Speicherschnittstelle

- ▶ Zugriff über AHB1
- ▶ Schnittstelle für
 - ▶ **Statisches RAM**
 - ▶ **Dynamisches RAM**
- ▶ Unterschiedliche Busbreiten:
 - ▶ **8 bit**
 - ▶ **16 bit**
 - ▶ **32 bit**



19

Quelle: LPC24XX User Manual

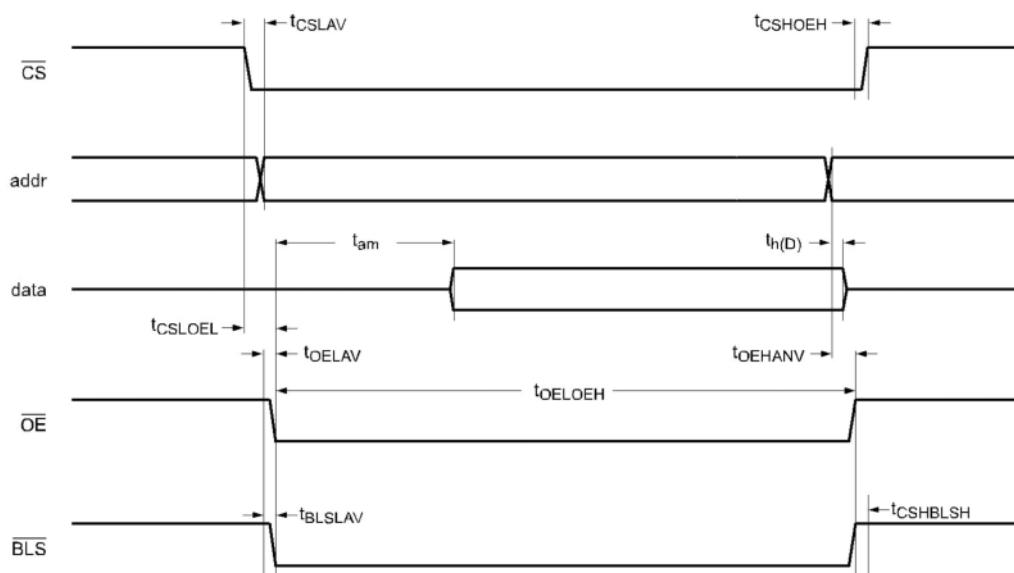
Speicherbereiche für externe Speicherschnittstelle 8 bit

Chip select pin	Address range	Memory type	Size of range
CS0	0x8000 0000 - 0x80FF FFFF	Static	16 MB
CS1	0x8100 0000 - 0x81FF FFFF	Static	16 MB
CS2	0x8200 0000 - 0x82FF FFFF	Static	16 MB
CS3	0x8300 0000 - 0x83FF FFFF	Static	16 MB
DYCS0	0xA000 0000 - 0xAF00 FFFF	Dynamic	256 MB
DYCS1	0xB000 0000 - 0xBFFF FFFF	Dynamic	256 MB
DYCS2	0xC000 0000 - 0xCFFF FFFF	Dynamic	256 MB
DYCS3	0xD000 0000 - 0xDFFF FFFF	Dynamic	256 MB

20

Quelle: LPC24XX User Manual

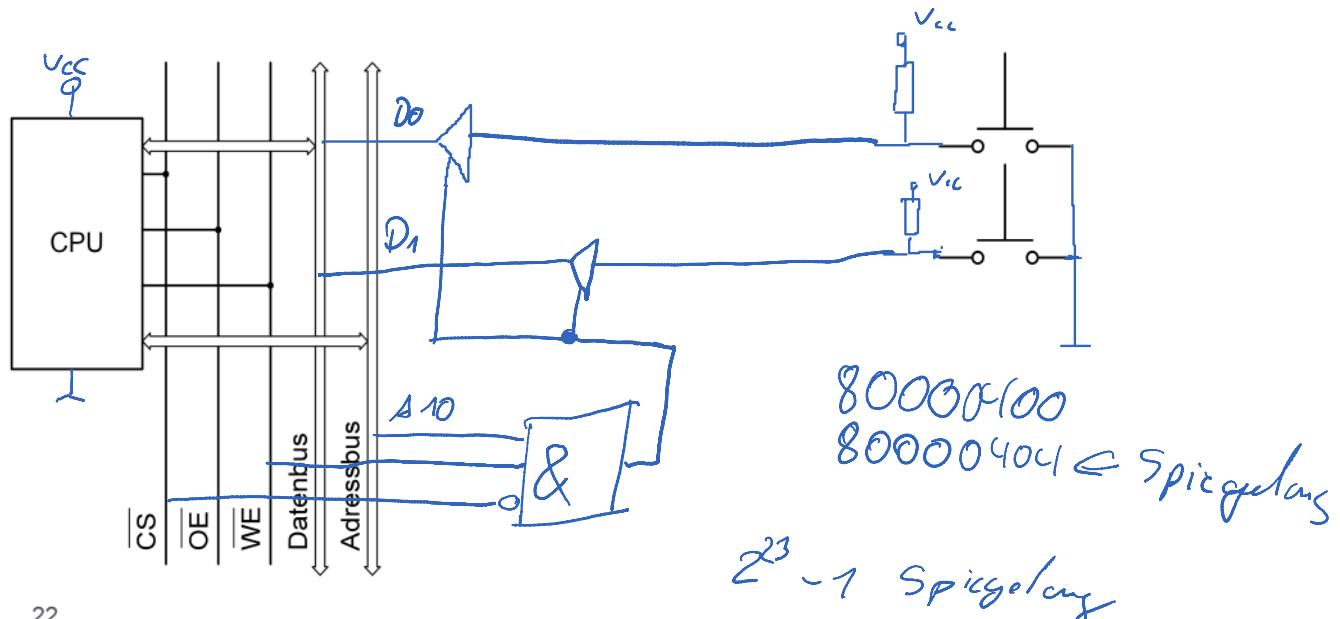
Externe Speicherschnittstelle, Lesezugriff auf statischen Speicher



21

Quelle: LPC2468 Data Sheet

Lesezugriff, Anschließen von Tasten



22

Lesezugriff

- Ausgewählte Speicheradresse: $CS0 + \underline{\text{RO}} = 0x8000\ 0000$
- Assembler

$\text{LDR RO, } \underline{\text{RO}} = 0x80000400$

LDR RO, [RO]

AND RO, RO, #3

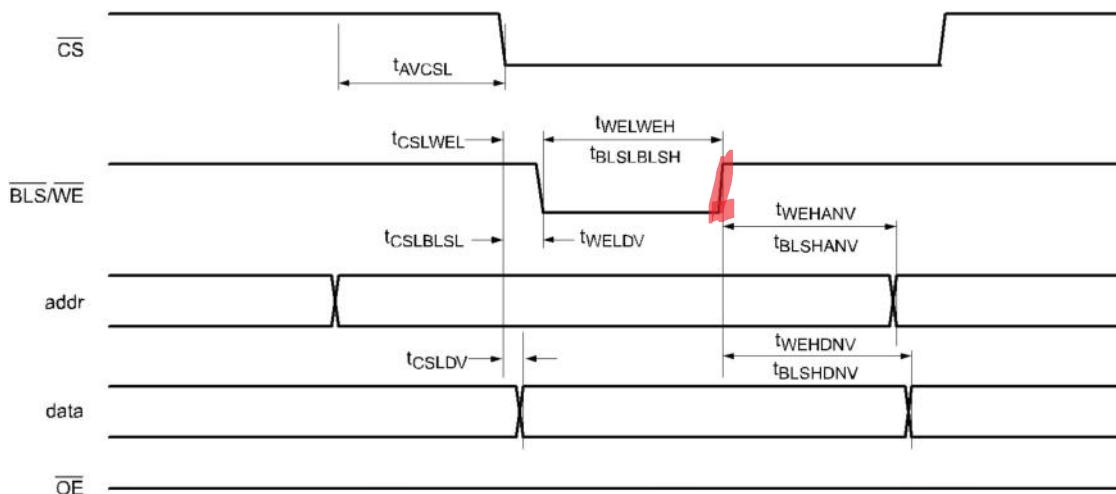
- C unsigned int Taste;
unsigned int volatile *pTaste;

$pTaste = (\text{unsigned int} *) 0x80000400;$

$Taste = *pTaste;$

23

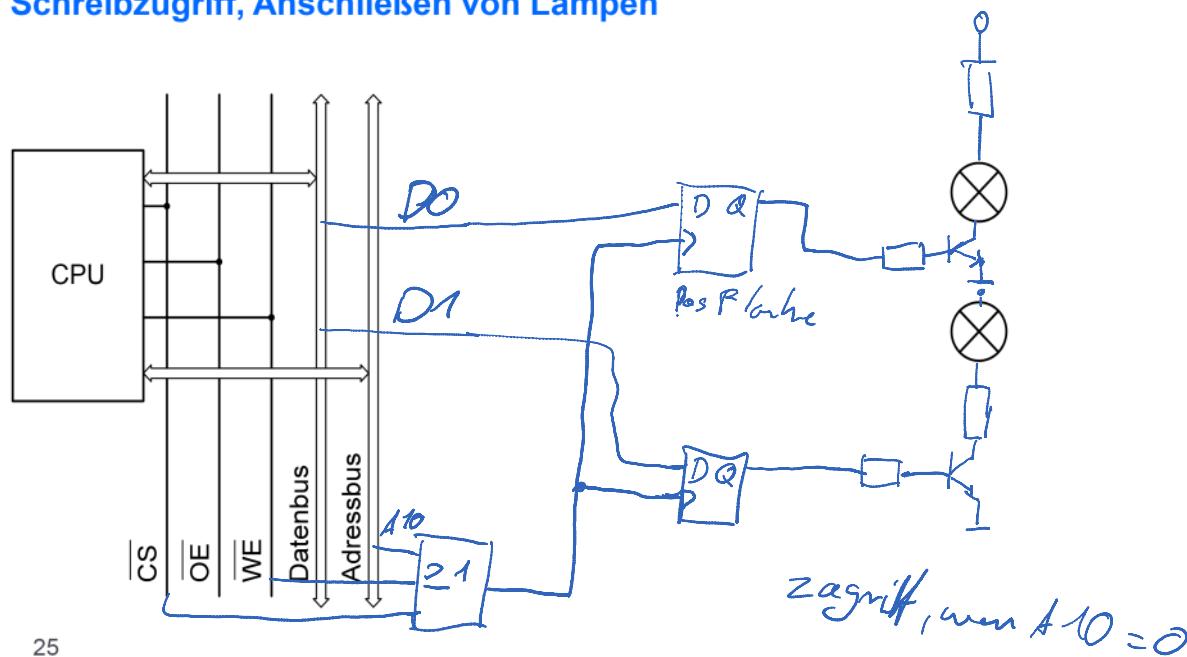
Externe Speicherschnittstelle, Schreibzugriff auf statischen Speicher



24

Quelle: LPC2468 Data Sheet

Schreibzugriff, Anschließen von Lampen



25

Schreibzugriff

- ▶ Ausgewählte Speicheradresse: $CS0 + \text{Offset} = 0x8000\ 0009$
- ▶ Assembler

```

MOV R1 #3           beide Längen
LDR R0, #0x80000000
STR R1[R0]

```

- ▶ C

26

unsigned int volatile *pLarpe = (unsigned int volatile *) 0x80000000;

*pLarpe = 3;
 [*pLarpe = *pLarpe & ~2;]
 read-modify-write-zugriff
 geht nicht mit HW, kann Schreibe

Übersicht

- ▶ Ein- und Ausgabe
 - ▶ Memory-Mapped I/O
 - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.



27

Steuerung des Stromverbrauchs

- ▶ Betriebszustände zum Reduzieren des Stromverbrauchs:
 - ▶ **Idle mode,**
 - ▶ **Sleep mode,**
 - ▶ **Power-down mode,**
 - ▶ **Deep power-down mode.**
- ▶ Stromverbrauch wird wesentlich durch die **Taktfrequenz** bestimmt
 - ▶ CPU-Taktfrequenz kann per Software verändert werden
- ▶ Steuerung des Leistungsverbrauchs von **Peripherie-Bausteinen**:
 - ▶ Nicht benötigte Bausteine können ausgeschaltet werden (siehe PCONP-Register).
 - ▶ Taktfrequenzen der einzelnen Komponenten lassen sich individuell einstellen (siehe PCLKSEL0 and PCLKSEL1).

28

Ein- und Ausschalten von Peripherie-Komponenten

- ▶ Einstellung über PCONP:
 - ▶ Für jede Komponente steht 1 Bit zur Verfügung:
 - 0: Komponente ist deaktiviert.
 - 1: Komponente wird mit Takt versorgt.

Ausschnitt von
PCONP:

29

Bit	Symbol	Description	Reset value
0	-	Unused, always 0	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I ² C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1

Takteinstellung Peripherie-Komponenten

- ▶ Einstellung über PCLKSEL0 und PCLKSEL1:
 - ▶ In PCLKSELx stehen für jede Komponente 2 Bits zur Verfügung.
 - ▶ Damit kann jede Komponente auf 4 verschiedene Taktfrequenzen eingestellt werden (siehe Tabelle).
 - ▶ Standard ist CCLK/4, CCLK ist die CPU-Frequenz, z.B.: 48 MHz

Ausschnitt von PCLKSEL0:

Bit	Symbol	Description
1:0	PCLK_WDT	Peripheral clock selection for WDT.
3:2	PCLK_TIMER0	Peripheral clock selection for TIMER0.
5:4	PCLK_TIMER1	Peripheral clock selection for TIMER1.
7:6	PCLK_UART0	Peripheral clock selection for UART0.
9:8	PCLK_UART1	Peripheral clock selection for UART1.

Bedeutung der Bits:

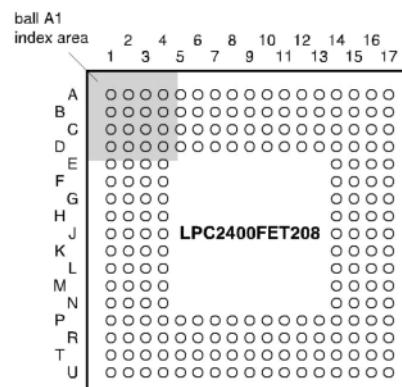
00	PCLK_xyz = CCLK/4
01	PCLK_xyz = CCLK
10	PCLK_xyz = CCLK/2
11	PCLK_xyz = CCLK/8 PCLK_xyz = CCLK/6 CAN1, CAN2, CAN filtering

30

Anschlüsse (Pins)

- ▶ Baustein hat 208 Pins
- ▶ Möglich wird dies durch ein Ball Grid Array (BGA)
- ▶ Anschlüsse über kleine Lotkugelchen (Balls).
- ▶ Dadurch Anordnung in mehreren Reihen und Spalten möglich.

LPC2400 pinning
TFBGA208 package



Transparent top view

31

Anschlüsse (Pins)

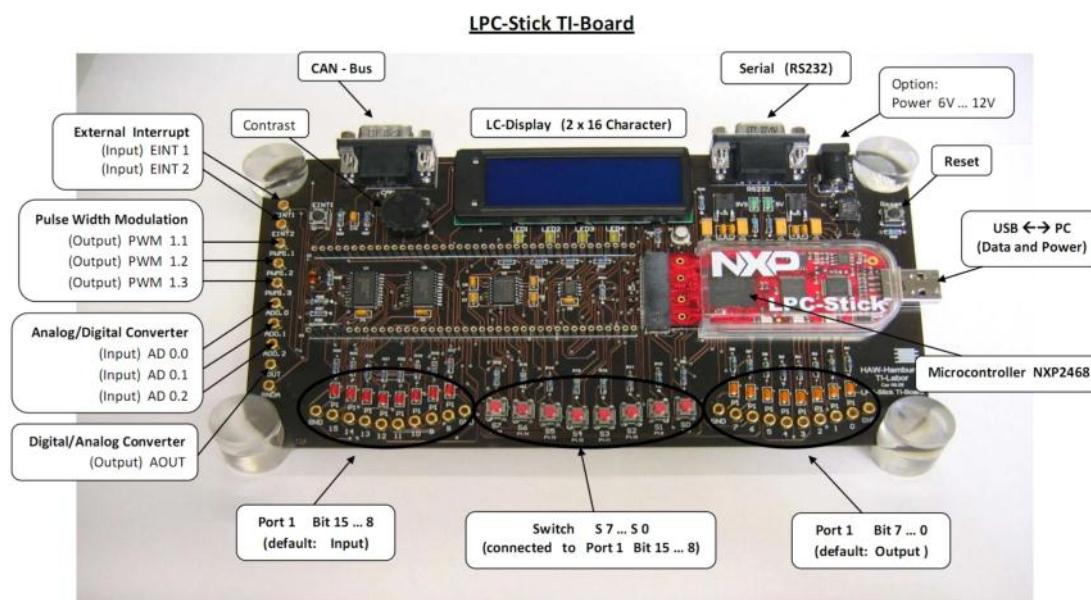
- ▶ Die meisten Pins können zwischen 4 verschiedene Funktionen per Software umgeschaltet werden.
- ▶ Hierzu dienen die 10 Register PINSEL0 bis PINSEL9.
- ▶ Standardmäßig ist die „General Purpose IO“ (GPIO)-Funktion aktiviert.

PINSEL4	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
1:0	P2[0]	GPIO Port 2.0	PWM1[1]	TXD1	TRACECLK	00
3:2	P2[1]	GPIO Port 2.1	PWM1[2]	RXD1	PIPESTAT0	00
5:4	P2[2]	GPIO Port 2.2	PWM1[3]	CTS1	PIPESTAT1	00
7:6	P2[3]	GPIO Port 2.3	PWM1[4]	DCD1	PIPESTAT2	00
9:8	P2[4]	GPIO Port 2.4	PWM1[5]	DSR1	TRACESYNC	00
11:10	P2[5]	GPIO Port 2.5	PWM1[6]	DTR1	TRACEPKT0	00
13:12	P2[6]	GPIO Port 2.6	PCAP1[0]	RI1	TRACEPKT1	00
15:14	P2[7]	GPIO Port 2.7	RD2	RTS1	TRACEPKT2	00
32						

Anschlüsse (Pins)

- ▶ Viele Pins können mit einem chipinternen Pull-up oder Pull-down Widerstand beschaltet werden.
- ▶ Hierzu dienen die 10 Register PINMODE0 bis PINMODE9.
- ▶ Standardmäßig ist der Pull-up Widerstand aktiviert.

PINMODE0 to PINMODE9 Values	Function	
00	Pin has an on-chip pull-up resistor enabled.	
01	Reserved. This value should not be used.	
10	Pin has neither pull-up nor pull-down resistor enabled.	
11	Pin has an on-chip pull-down resistor enabled.	
PINMODE4	Symbol	Description
1:0	P2.00MODE	PORT2 pin 0 on-chip pull-up/down resistor control.
...		
31:30	P2.15MODE	PORT2 pin 15 on-chip pull-up/down resistor control.



34

CPU Ein- und Ausgabe

Übersicht

- ▶ Ein- und Ausgabe
 - ▶ Memory-Mapped I/O
 - ▶ I/O-based I/O
- ▶ Bussysteme des LPC2468.
- ▶ Zugriff auf externe Komponenten.
Typischer Ablauf eines Buszugriffs.
- ▶ Systemeinstellungen für LPC2468-Komponenten.
- ▶ General Purpose I/O (GPIO) LPC2468.

35

General Purpose Input/Output

- Chipselect wird aktiviert beim Zugriff auf die Variable GPIO1_IOPIN.

- GPIO1_IOPIN ist in der Headerdatei lpc24xx.h definiert.

- Eingabe:** Alle 32 Bit lesen:

```
int x;
x = GPIO1_IOPIN;
printf( "%08X", x );
```

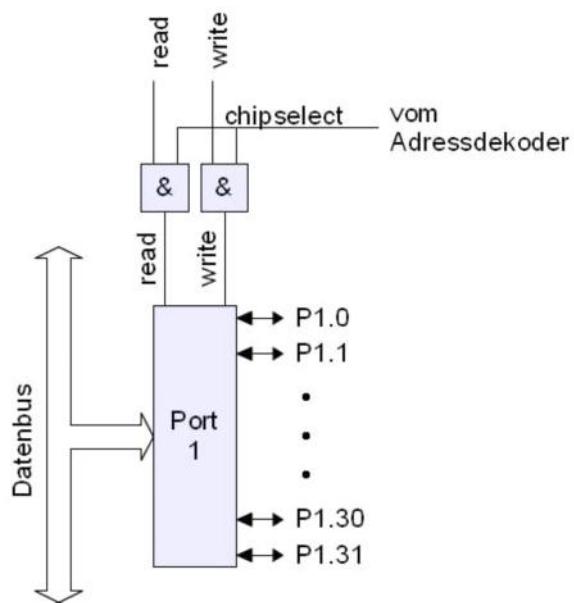
- Ausgabe:** Alle 32 Bit verändern:

```
GPIO1_IOPIN = 0x88FF0011;
```

- Fazit:**

- Mit GPIO1_IOPIN können nur alle 32 Ein-/Ausgabeleitungen gleichzeitig bearbeitet werden.

- Wie können einzelne Bits behandelt werden?



LPC2468: General Purpose Input/Output

- GPIOx_IODIR** steuert individuell die Ein- und Ausgaberichtung jedes einzelnen Pins.

- GPIOx_IOMASK** steuert den Zugriff über die Ports GPIOx_IOPIN, GPIOx_IOSET und GPIOx_IOCLR.
Ermöglicht Auswahl einzelner Bits.
Bit n == 0: Bit n kann gelesen oder beschrieben werden
Bit n == 1: Beim Lesen ist dieses Bit immer 0
Beim Schreiben wird dieses Bit nicht verändert

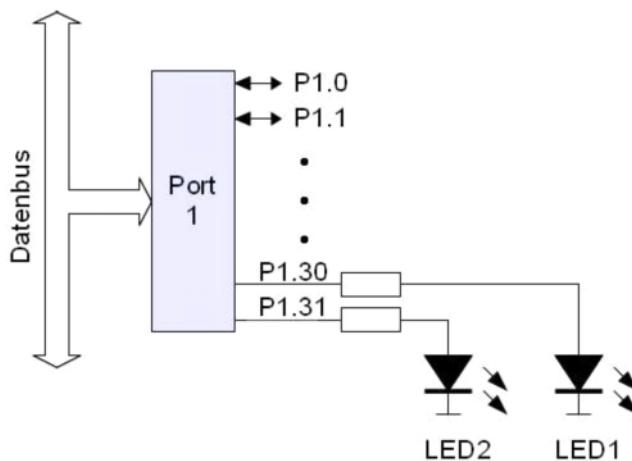
- GPIOx_IOPIN** Lesen des momentanen Zustandes der mit GPIOx_IOMASK ausgewählten Bits.
Schreiben der mit GPIOx_IOMASK ausgewählten Bits.

- GPIOx_IOSET** Setzen der mit GPIOx_IOMASK ausgewählten Bits.
Bit n == "1": Ausgang n wird High (logisch 1).
Bit n == "0": Ausgang n wird nicht verändert.

- GPIOx_IOCLR** Löschen der mit GPIOx_IOMASK ausgewählten Bits.
Bit n == "1": Ausgang n wird Low (logisch 0).
Bit n == "0": Ausgang n wird nicht verändert.

Ansteuerung einzelner Bits

- ▶ LED1 einschalten:



- ▶ LED1 ausschalten:

38

ÜBUNG: Ein- und Ausgabe, Bitmanipulation

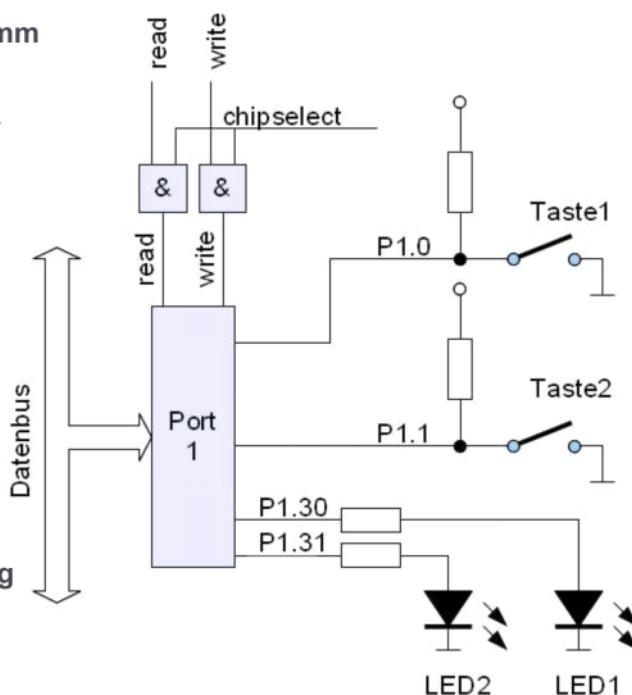
- ▶ Erstellen Sie ein Unterprogramm zum Ansteuern der LEDs:

```
void setLed( int leds );
```

mit folgendem Verhalten:

leds	LED1	LED2
0	aus	aus
1	ein	aus
2	aus	ein
3	ein	ein

- ▶ a) ohne Ausnutzung der LPC2xxx Hardware
- ▶ b) mit Hardwareunterstützung



39

Computer Engineering

WS 2012

Timer/Counter

HTM – SHF - SWR

Übersicht

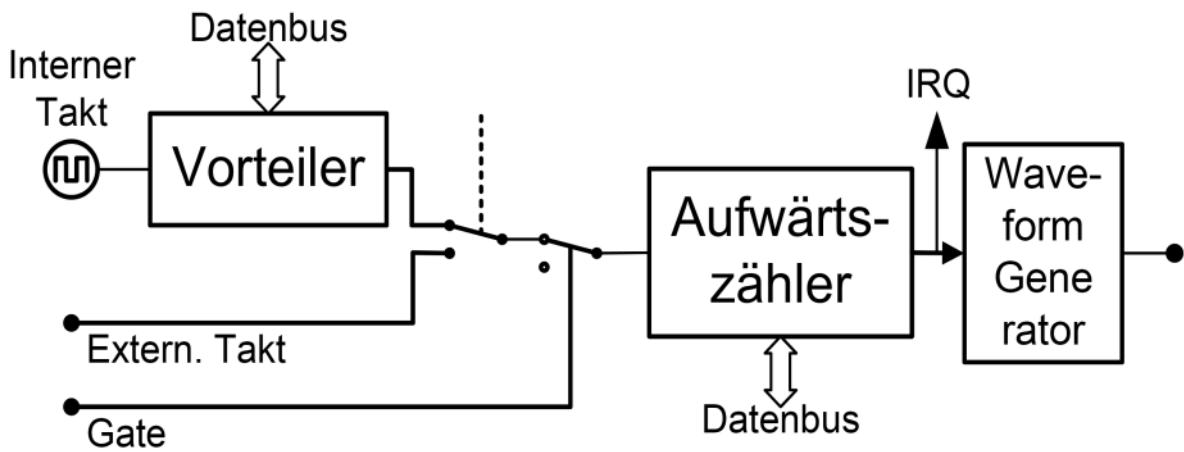


- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468

Aufgaben

- ▶ Erzeugen von Zeitfunktionen
 - ▶ Programmteile, die zu genau vorgegebenen Zeitpunkten ausgeführt werden sollen.
 - ▶ Externe Signale zur Steuerung von Peripheriegeräten.
- ▶ Messung von Zeitfunktionen
 - ▶ Dauer von externen Signalen.
 - ▶ Frequenz von externen Signalen.
- ▶ Zählen von Ereignissen.

Prinzipieller Aufbau



Timer/Counter

Anwendungen:

Zeitgeber

- Periodisch oder Single Shot
- Interner Takt
- Beispielanwendungen
 - Systemtakt für Betriebssysteme
 - Schrittmotorsteuerung

Zeitmessung

- Interner Takt
- Start und/oder Stop durch externes Ereignis
- Beispielanwendungen
 - Bestimmung der Baudate

Frequenzmessung

- Externer Takt
- Start und Stop des Zählers per Software

Zähler

- Externe Ereignisse werden als Takt verwendet
- Beispielanwendungen
 - Auswertung von Drehimpulsgebern

Überwachung von und Reaktion auf Timer-Ereignisse

- ▶ Polling

- ▶ Kontinuierliche Abfrage
 - des Zählerstandes oder
 - des Statusregisters:

```
/* warte, bis Timer-Ereignis auftritt */  
  
while(( TIMER0_IR & MRO) == 0 ) {  
}
```

- ▶ Interrupt

- ▶ Automatische Änderung von Signalen an Anschlusspins

- ▶ Mögliche Aktionen an Ausgängen bei Timer-Ereignissen:

- Setzen
- Löschen
- Toggeln

Timer/Counter

Zeitliche Auflösung

- ▶ bestimmt durch verwendete
 - ▶ Taktfrequenz

$$\Delta T = \frac{N_{Vorteiler}}{f_{Takt}}$$

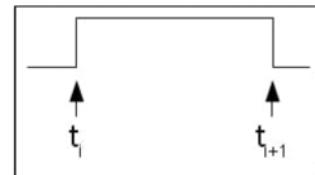
Intervall

- ▶ bestimmt durch verwendete
 - ▶ Taktfrequenz
 - ▶ Anzahl Bits des Zählers

$$T_{Max} = 2^N \cdot \frac{N_{Vorteiler}}{f_{Takt}}$$

Übung: Dimensionierung eines Timers

- ▶ Mit Hilfe eines 16-Bit Timers soll eine Impulsdauer zwischen 0.5 und 2.0 Sekunden gemessen werden. Die Systemfrequenz beträgt 8 MHz. Als Vorteiler stehen die Werte 1, 8, 64, 256 und 1024 zur Verfügung.



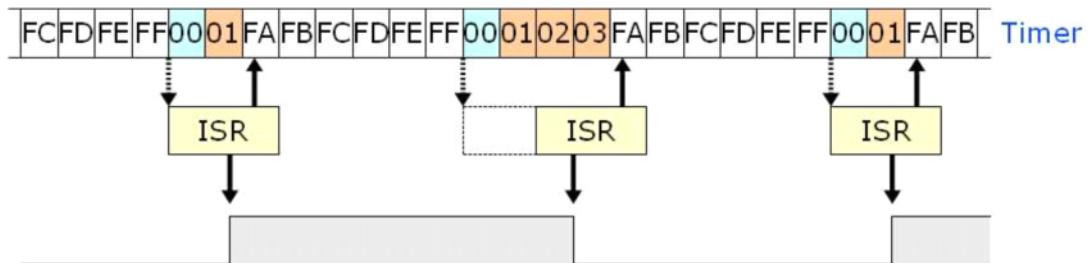
1. Wie muss der Vorteiler eingestellt werden?
2. Wie genau kann die Zeit erfasst werden?
Verwenden Sie bitte handhabbare Einheiten.

Übersicht



- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468

Erzeugung eines periodischen Ausgangssignals

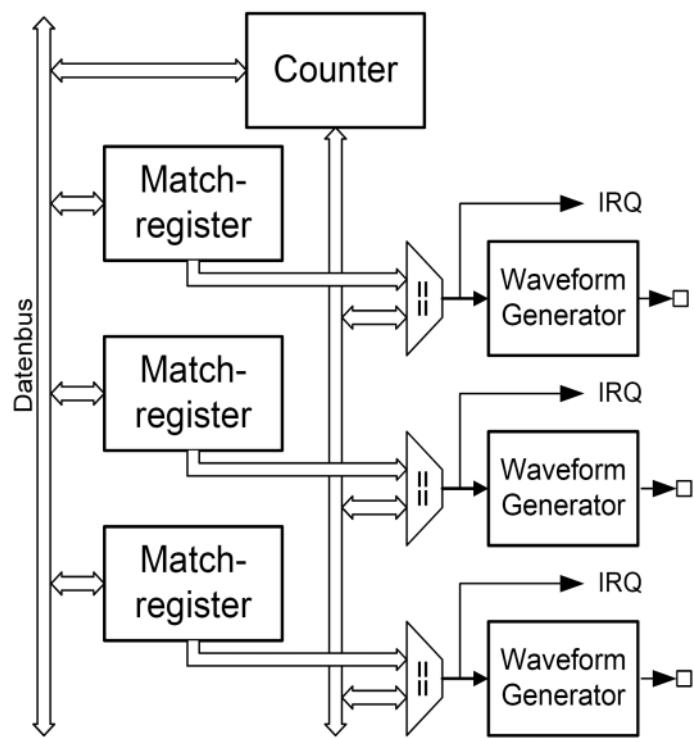


- ▶ **Ablauf:**
 - ▶ Beim Überlauf des Zählers wird Overflow-Interrupt ausgelöst.
 - ▶ In der ISR wird der Zähler auf den Ausgangswert zurückgesetzt.
- ▶ **Probleme:**
 - ▶ Latenz- und Bearbeitungszeit des Interrupts bewirken:
 - Falsche Frequenz
 - Jitter.
 - ▶ Timer kann nur ein Signal zur Zeit erzeugen.

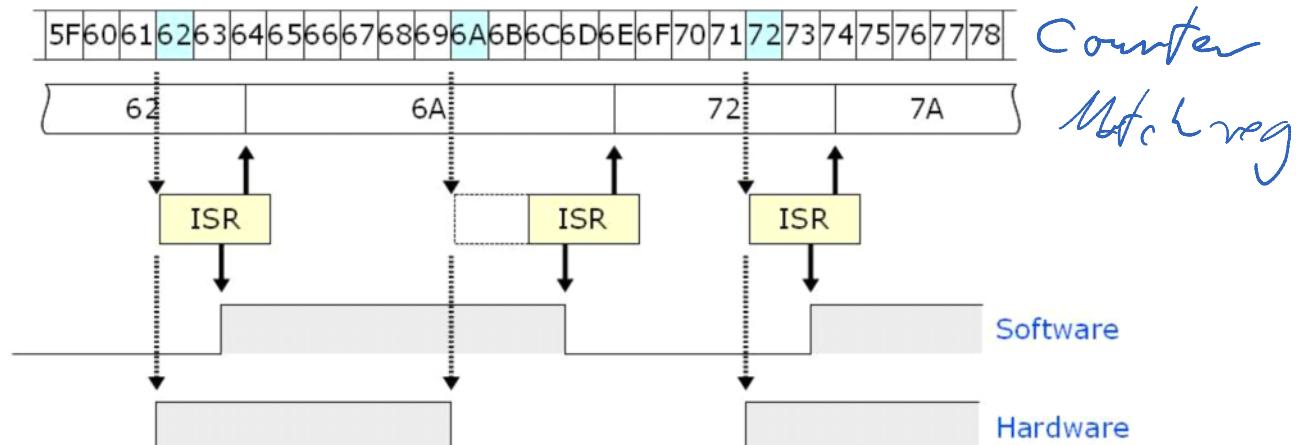
Timer/Counter

Timer mit Output-Compare-Funktion

- Ermöglicht die Erzeugung von mehreren unabhängigen Signalen



Output-Compare-Funktion: Periodisches Ausgangssignal



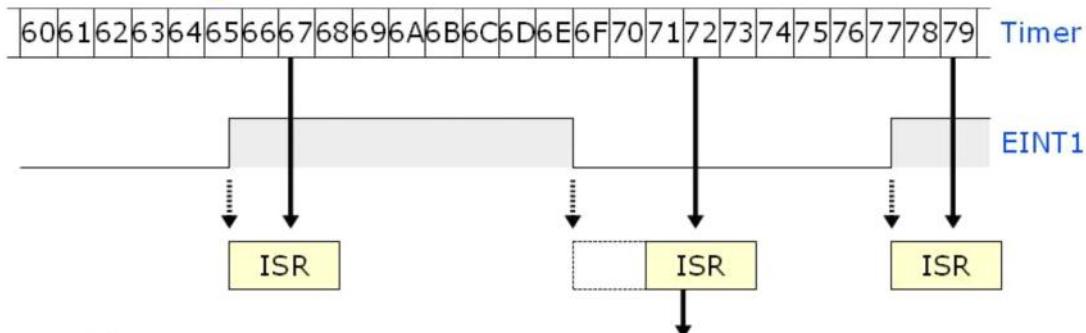
- ▶ Hardwaregesteuerter Ausgang:
 - ▶ taktgenau
- ▶ Softwaregesteuerter Ausgang:
 - ▶ richtige Frequenz
 - ▶ aber Jitter

Übersicht



- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468

Zeitmessung



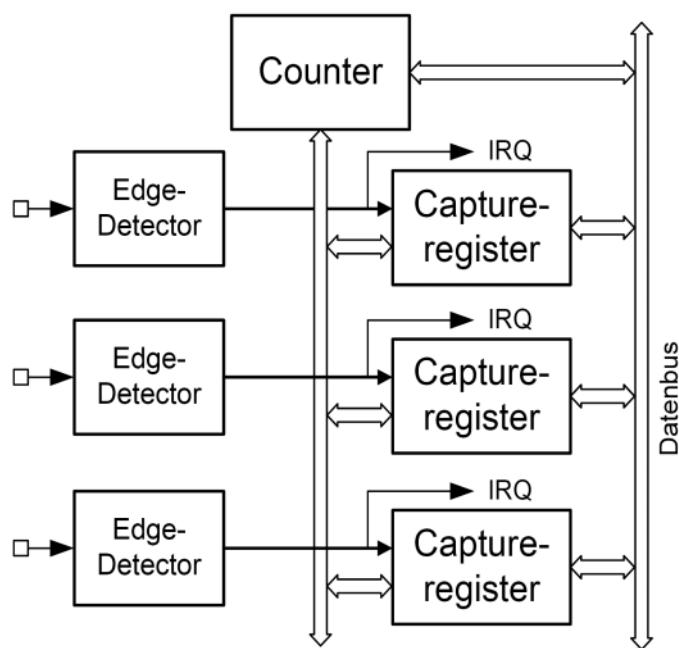
► Idee:

- ▶ Externes Signal löst Interrupt aus.
- ▶ In der ISR werden die Zeitpunkte mit Hilfe des intern getakteten Timers bestimmt.
- ▶ Aber Latenz- und Bearbeitungszeiten des Interrupts bewirken:
 - ▶ Ungenaue Messung
 - ▶ Jitter in Messergebnisse

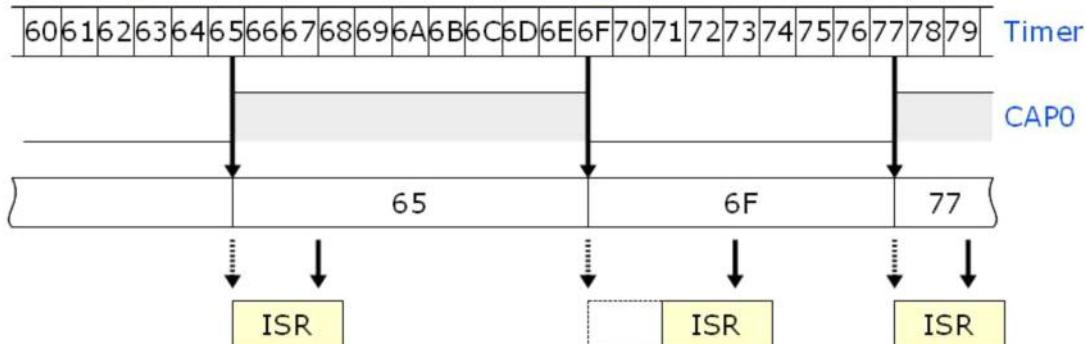
Timer

Timer mit Input-Capture-Funktion

- Ermöglicht das taktgenaue Messen von externen Ereignissen.



Input-Capture-Funktion: Zeitmessung



- ▶ Latenz- und Bearbeitungszeit des Interrupts haben keinen Einfluss auf die Messung:
 - ▶ **taktgenaue Bestimmung möglich.**
- ▶ Aber Ergebnis muss bis zum Eintreffen des nächsten Ereignisses gelesen werden.

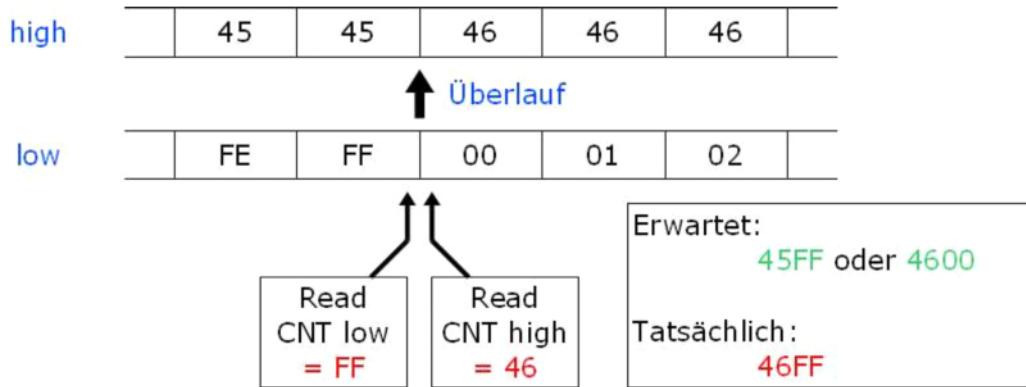
Übersicht



- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468

CE WS12

Zugriff auf 16-Bit Zähler mittels 8-Bit Bus



- ▶ Lesen des 16-Bit Registers erfordert zwei Zugriffe.
- ▶ Normalerweise Ungenauigkeit von 1
- ▶ Bei gleichzeitiger Änderung des High-Registers:
 - **Abweichung von 255**

18



Zugriff auf 16-Bit Zähler mittels 8-Bit Bus:

Sicherer Lesezugriff per Software

▶ Problematisch:

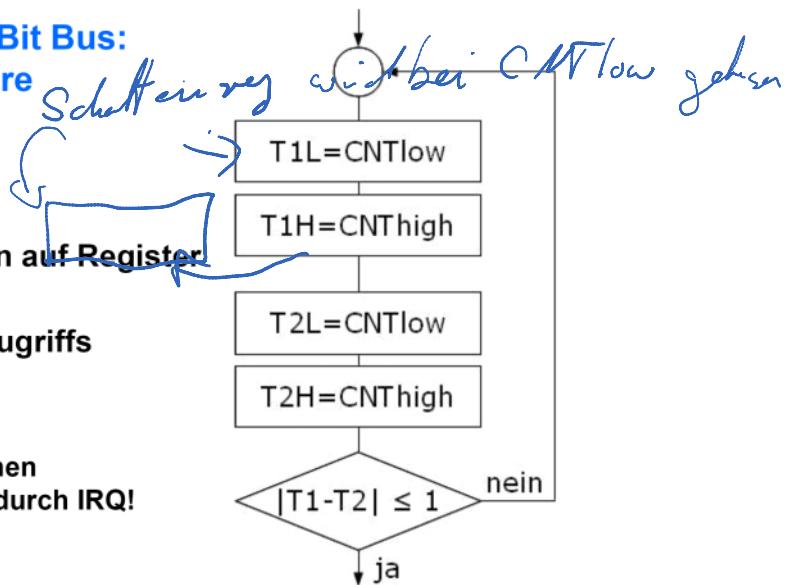
▶ mindestens 4 Leseoperationen auf Register

▶ Zeitliche Ungenauigkeit des Zugriffs

▶ Achtung:

- Unterbrechung möglich zwischen Lesezugriffe auf low und high durch IRQ!

▶ Probleme bei Schreibzugriffe?



LPC2468: Lese- und Schreibzugriffe unproblematisch,
da 32-Bit Zugriffe und 32-Bit Register (atomarer Zugriff möglich)

Übersicht

- ▶ Einleitung
- ▶ Output-Compare Funktion
- ▶ Input-Capture Funktion.
- ▶ Mehrfachzugriff auf Counter-Register.
- ▶ Timer/Counter im LPC2468



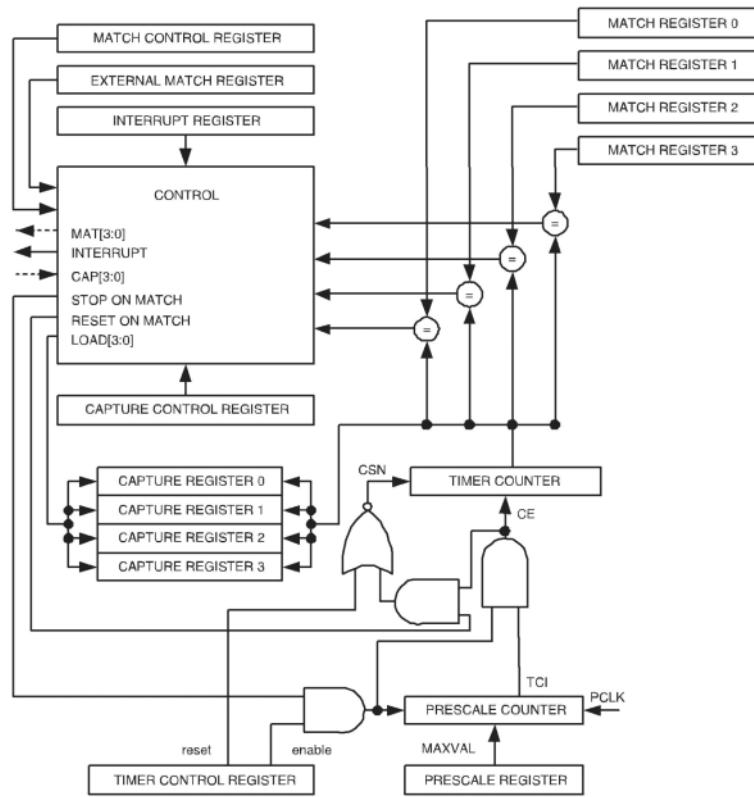
Timer-Bausteine im LPC2468

- ▶ LPC2468 hat 4 Timer-Bausteine: Timer0 bis Timer3.
- ▶ Zusätzlich 2 „Pulse Width Modulators“ (PWM).
- ▶ Hohe zeitliche Auflösung:
 - ▶ Alle Register sind haben 32-Bit.
 - ▶ Maximale Taktfrequenz: Systemtakt (typisch 48 MHz).
- ▶ 4 Capture-Register zum Erfassen externer Signale.
- ▶ 4 Match-Register mit jeweils 3 Betriebsarten:
 - ▶ Kontinuierlicher Betrieb.
 - ▶ Stop beim Erreichen des Match-Wertes (Single Shot).
 - ▶ Zurücksetzen des Zählers beim Erreichen des Match-Wertes.
- ▶ 4 Ausgänge / 4 Eingänge pro Timer möglich.

Timer/Counter

Timer-Bausteine im LPC2468: Blockdiagramm

22



Timer-Bausteine im LPC2468: Datenregister

- ▶ Timer Counter Register **TIMERn_TC**
 - ▶ Aktueller Zählerstand
- ▶ Prescale Register **TIMERn_PR**
 - ▶ Maximaler Zählerstand des Vorteilerzählers.
- ▶ Prescale Counter Register **TIMERn_PC**
 - ▶ Aktueller Zählerstand des Vorteilerzählers.
- ▶ Match Register **TIMERn_MR[0/1/2/3]**
 - ▶ Wert des Match Registers
- ▶ Capture Register **TIMERn_CR[0/1/2/3]**
 - ▶ Wert des Capture Registers.

Timer-Bausteine im LPC2468: Steuerregister

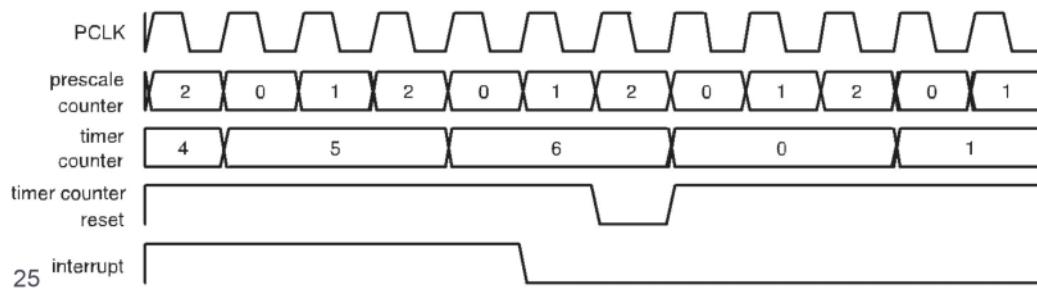
- Interrupt Register **TIMERn_IR**
 - Zeigt an, ob ein Match- oder Capture-Event aufgetreten ist.
Muss per Software zurückgesetzt werden.
- Timer Control Register **TIMERn_TCR**
 - Reset und Freigabe des Zählers.
- Count Control Register **TIMERn_CTCR**
 - Einstellung Betriebsmode: Zeitgeber oder Zähler.
Auswahl des Zähleingangs.
- Match Control Register **TIMERn_MCR**
 - Festlegung, was bei einem Match-Event passieren soll:
Interrupt, Reset des Zählerstandes, Stop des Zählers.
- Capture Control Register **TIMERn_CCR**
 - Festlegung, welche Flanke erfasst werden soll.
Freigabe des Interrupts.
- External Match Register **TIMERn_EMR**
 - Festlegung, was bei einem Match-Event mit dem externen Match-Ausgang (MATn.m) passieren soll: High, Low, Toggle

Timer-Bausteine im LPC2468: Anwendungsbeispiel 1

► Kontinuierlicher Zähler

```

PCONP      |= (1<<22); // enable timer
PCLKSEL1   |= (1<<12); // set clock to cclk (48 MHz)
TIMER2_TCR = 0x02;      // reset timer
TIMER2_MR0 = 6;          // set match register 0
TIMER2_PR  = 2;          // set prescaler
TIMER2_IR  = 0xff;       // reset all interrupts
TIMER2_MCR = 3<<0;      // interrupt on MR0, reset on MR0,
                         // do not stop timer on match
TIMER2_TCR = 0x01;       // start timer
    
```

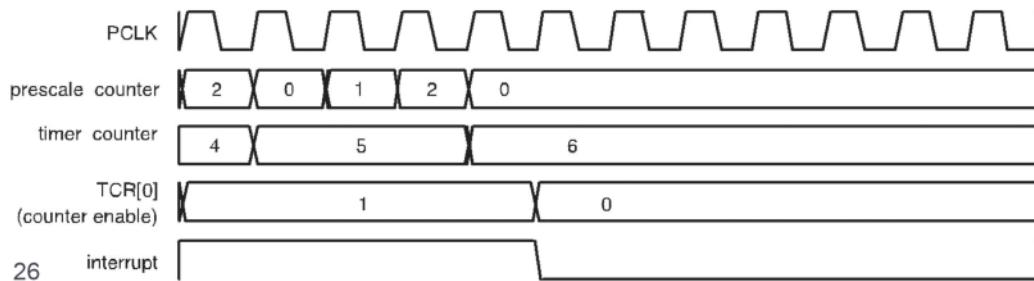


Timer-Bausteine im LPC2468: Anwendungsbeispiel 2

▶ Single Shot

```

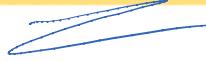
PCONP      |= (1<<22); // enable timer
PCLKSEL1   |= (1<<12); // set clock to cclk (48 MHz)
TIMER2_TCR = 0x02;      // reset timer
TIMER2_MR0 = 6;         // set match register 0
TIMER2_PR  = 2;         // set prescaler
TIMER2_IR  = 0xff;      // reset all interrupts
TIMER2_MCR = 5<<0;     // interrupt on MR0, no reset on MR0,
                        // stop timer on match
TIMER2_TCR = 0x01;      // start timer
    
```



Timer-Bausteine im LPC2468: Anwendungsbeispiel 3

- ▶ LED2 auf dem TI-Board blinkt mit 1 Sekunde
- ▶ Verwendung des externen Match Signals MAT0.0

```
PCONP |= (1<<1); // enable timer
PCLKSEL0 |= (1<<2); // set clock to cclk
PINSEL3 |= (3<<24); // P1.28 is MAT0.0
TIMER0_TCR = 0x02; // reset timer
TIMER0_PR = 0; // set prescaler
TIMER0_IR = 0xff; // reset all interrrupts
TIMER0_MCR = 3<<0; // interrupt on MR0, reset on MR0,
// do not stop timer on match
TIMER0_EMR = 3<<4; // toggle external match bit
TIMER0_TCR = 0x01; // start timer
TIMER0_MR0 = 48000000 /*Hz*/ / 2 /*Hz*/; -7
```



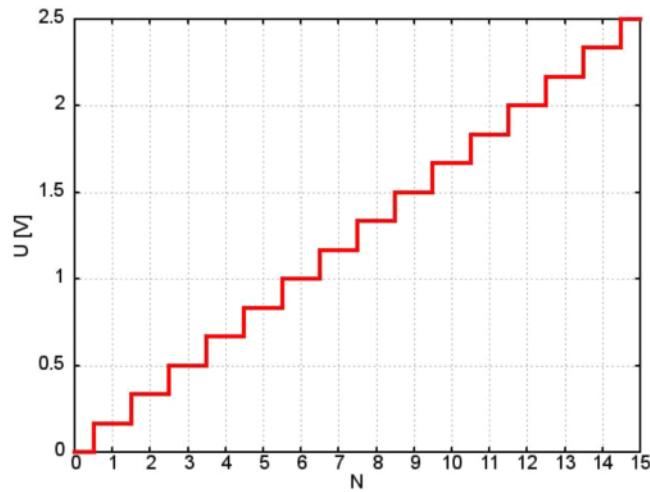
Computer Engineering WS 2012

PWM

HTM – SHF - SWR

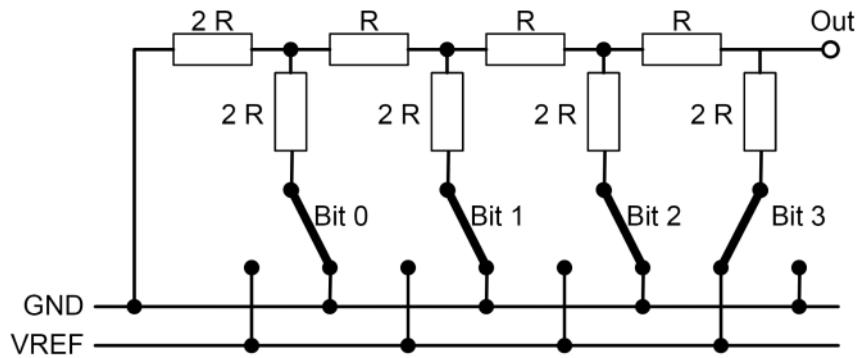
Prinzip

- ▶ Erzeugung einer analogen Spannung
- ▶ Eingang:
 - ▶ Integerzahl N im Bereich N_{\min} bis N_{\max} ,
z.B. M-Bit Wandler: 0 bis $2^M - 1$
- ▶ Ausgang:
 - ▶ Analoge Spannung U im Bereich U_{\min} bis U_{\max}
 - ▶ Linear abhängig von N



Prinzipieller Aufbau

► R/2R-Netzwerke



LPC2468: DAC

- ▶ Eigenschaften:
 - ▶ 10-Bit Digital-Analog Wandler
 - ▶ Separate Anschlüsse für Spannungsversorgung: V_{DDA} , V_{SSA}
 - ▶ Separater Anschluss für Referenzspannung: V_{REF}
 - Im Praktikum: $V_{REF} = 3,3$ Volt
 - ▶ $A_{OUT} = \text{VALUE} / 1024 * V_{REF}$
- ▶ Initialisierung:
 - ▶ Takt einstellen: PCLKSEL0
 - ▶ Ausgangspin AOUT aktivieren und konfigurieren:
PINSEL1 und PINMODE1.
- ▶ Ausgabe:
 - ▶ Wert in das Register DACR schreiben.
Bitanordnung beachten.

DAC: Nachteile

- ▶ Aufwändige analoge Schaltung
 - ▶ Kostenintensiv
 - ▶ Störanfällig

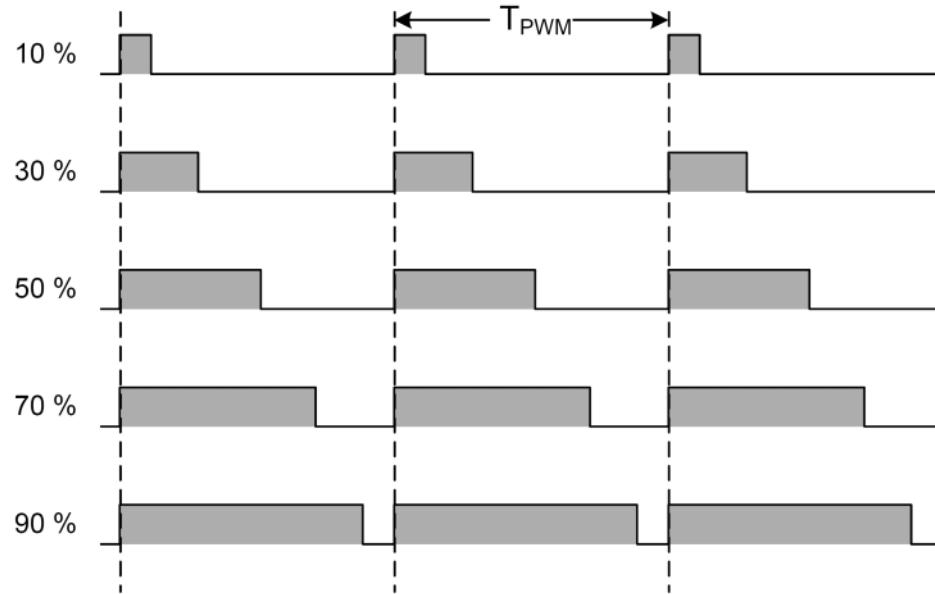
Pulsweitenmodulation (PWM)

- ▶ Kann mit reinen digitalen Schaltungen realisiert werden
- ▶ Kostengünstig in Mikrocontroller integrierbar.
- ▶ Vielfältige Einsatzmöglichkeit, z.B.:
 - ▶ Erzeugung analoger Signale,
 - ▶ Ansteuerung von Gleichstrommotoren,
 - ▶ Ansteuerung von Schritt- und bürstenlosen Motoren,
 - ▶ Signalübertragung (z.B.: Funkfernsteuerungen).

CE WS12

Grundprinzip

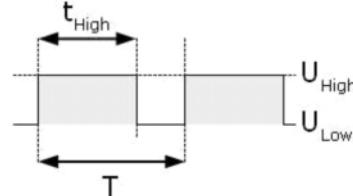
- ▶ Signal mit vorgegebener Frequenz und
- ▶ variabler Pulspausenzeit.



Tastverhältnis (Duty Cycle)

- Verhältnis der Dauer des High-Zustandes zur Periodendauer

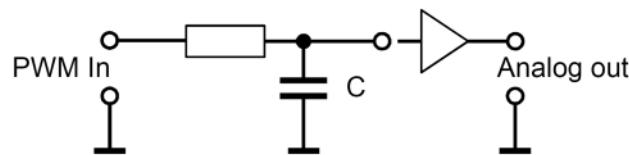
$$D = \frac{t_{High}}{T}$$



Spannungsmittelwert

$$U_m = U_{Low} + D \cdot (U_{High} - U_{Low})$$

- Lineare Beziehung zwischen U_m und D !
- Mittelwertberechnung mit Hilfe eines Tiefpassfilters



7

Tastverhältnis (Duty Cycle)

- ▶ Basiert auf Zähler mit M Bit.
- ▶ Zähler wird mit festem Takt f_{Takt} betrieben.
- ▶ Zähler wird beim Erreichen von N_{TOP} zurückgesetzt.
- ▶ Steuerung des PWM-Ausgangs:
 - ▶ z.B.: High beim Zurücksetzen des Zählers und
 - ▶ Low beim Erreichen des Match-Registers.

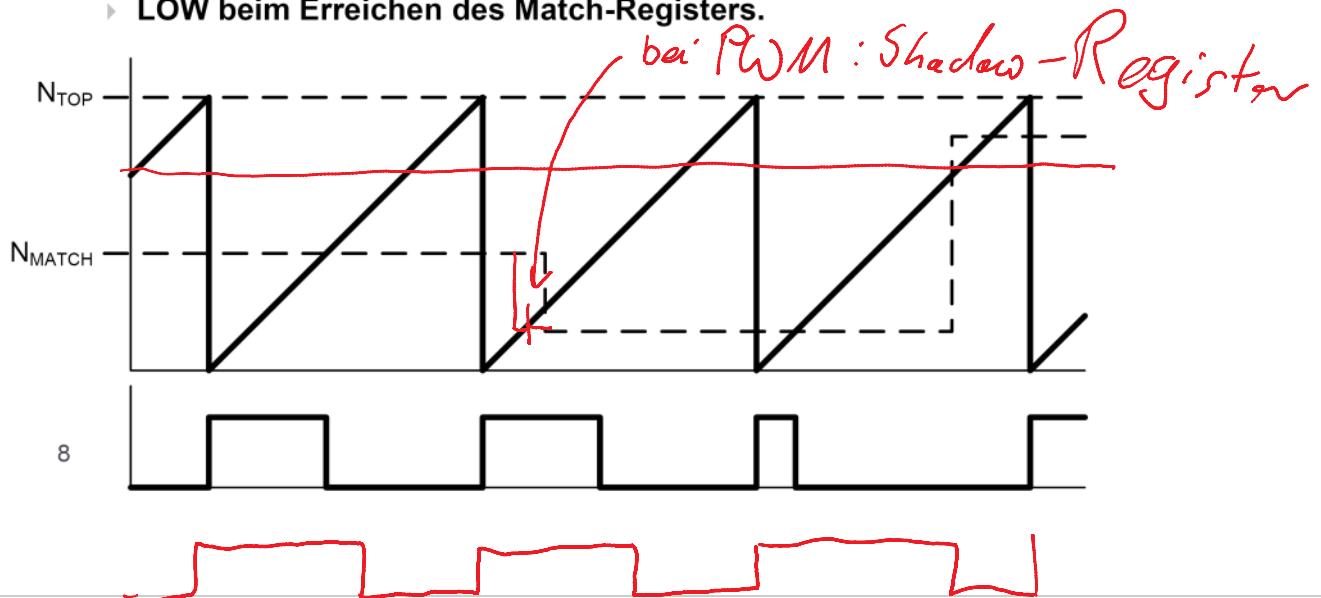


Abbildung auf physikalische Größen

- ▶ Meist linearer Zusammenhang.
- ▶ Beispiel Temperatursollwert (z.B. für Heizung):
 - ▶ Gefordert: Bereich von -40 °C bis 100 °C
 Auflösung 0.2 °C
 - ▶ Realisierung:

PWM-Zähler	Temperatur
0	-40 °C
200	0 °C
700	100 °C
 - ▶ Mindestens erforderlich: 10-Bit Zähler
 - ▶ Annahme Taktfrequenz $f_{Takt} = 8 \text{ MHz}$
 - ▶ $f_{PWM} = f_{Takt} / N_{TOP} = 11.4 \text{ kHz}$

Kenngrößen

- ▶ f_{PWM} Frequenz der PWM-Ausgangsimpulse.
- ▶ f_{Takt} Taktfrequenz des PWM-Zählers.
- ▶ Auflösung (Resolution):

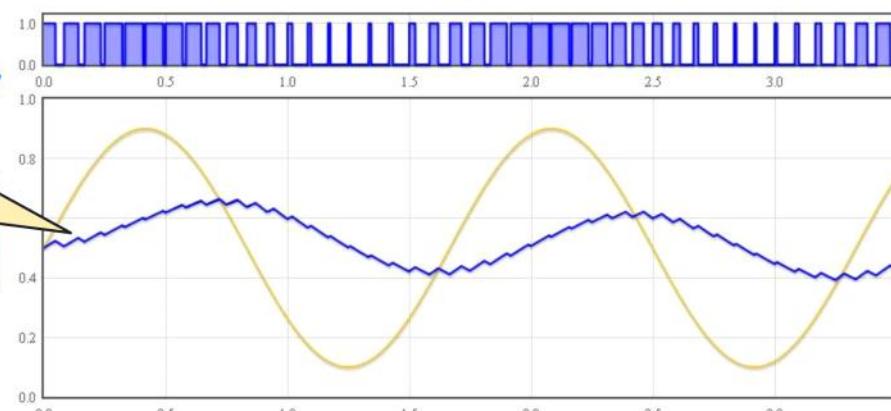
- ▶ Bereich (Range):

- ▶ Einschwingzeit (settling time):

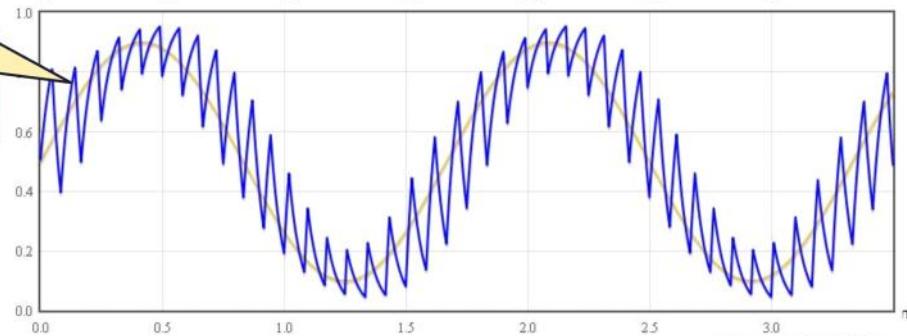
CE WS12

Optimierung Tiefpassfilter

Grenzfrequenz
zu niedrig

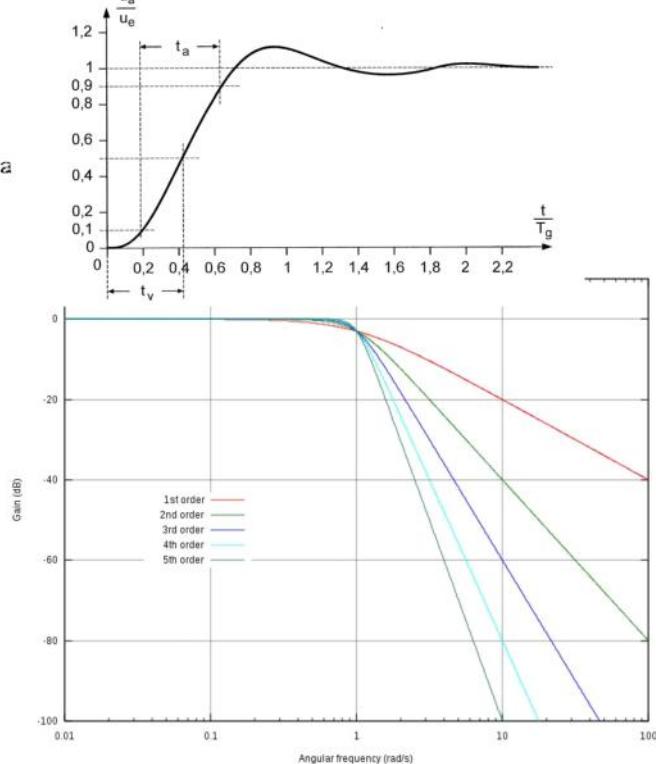
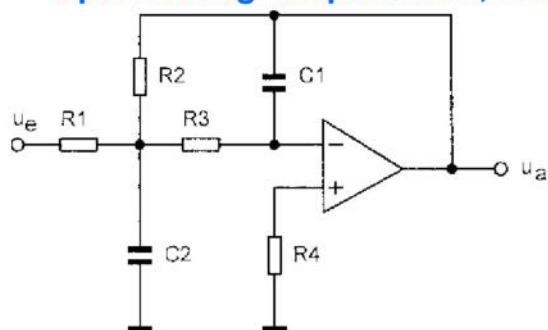


Grenzfrequenz
zu hoch

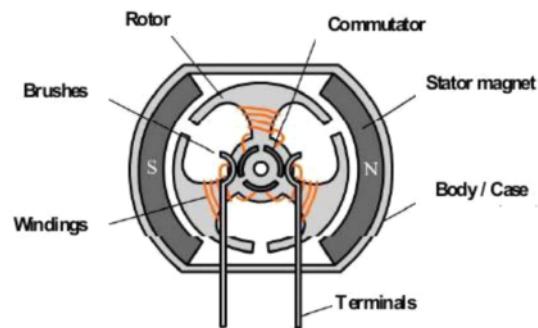
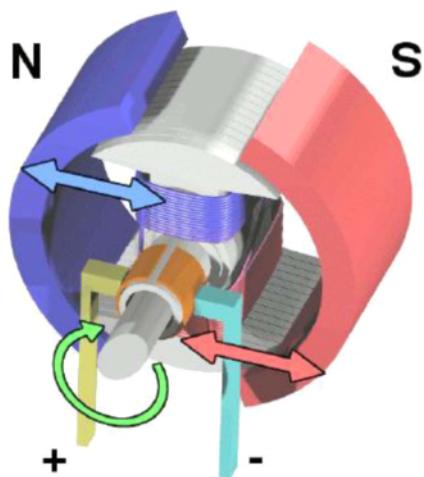


11

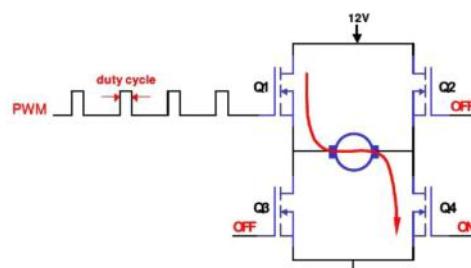
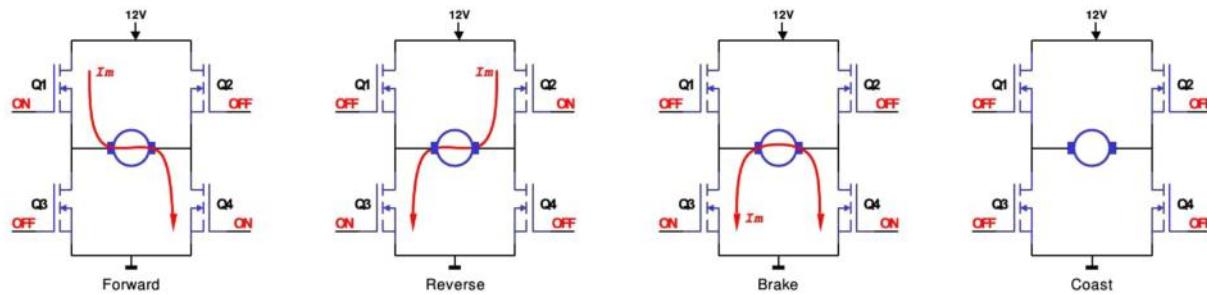
Optimierung Tiefpassfilter, Butterworthfilter zweiter Ordnung



Aus: Trampert,
„Messen, Steuern und Regeln
mit AVR Mikrocontrollern“
und Wikipedia



Anwendung PWM: Ansteuerung von Gleichstrommotoren



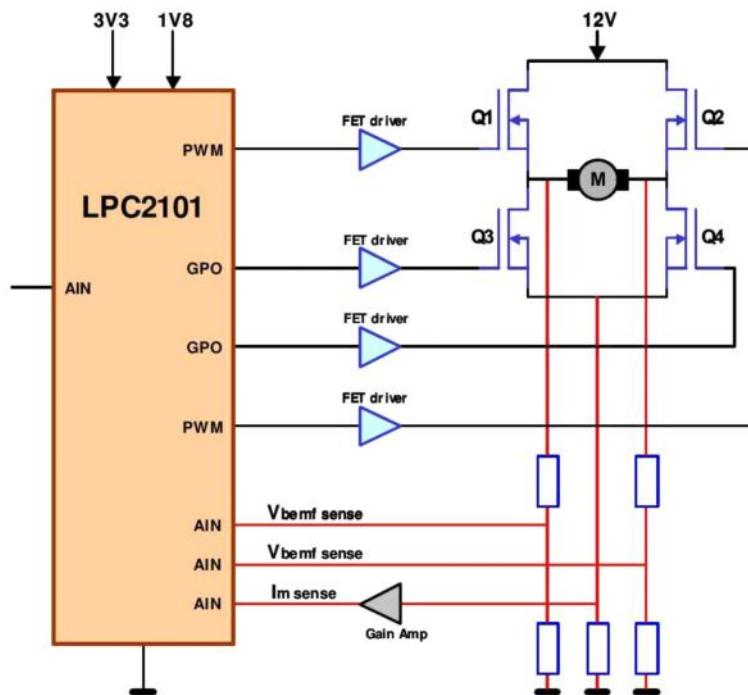
14

aus http://www.nxp.com/documents/application_note/AN10513.pdf



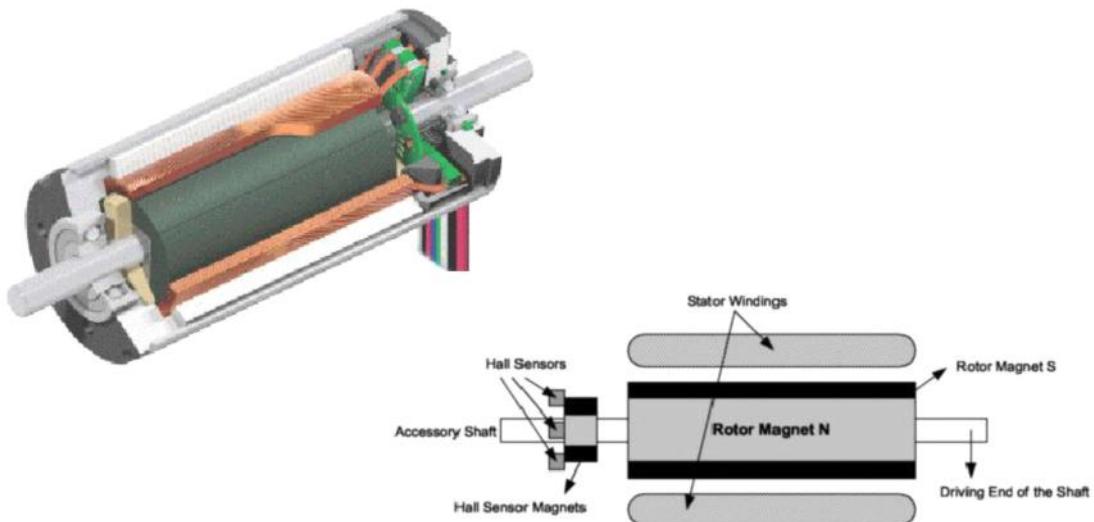
PWM

Anwendung PWM:
Ansteuerung von
Gleichstrommotoren



aus http://www.nxp.com/documents/application_note/AN10513.pdf

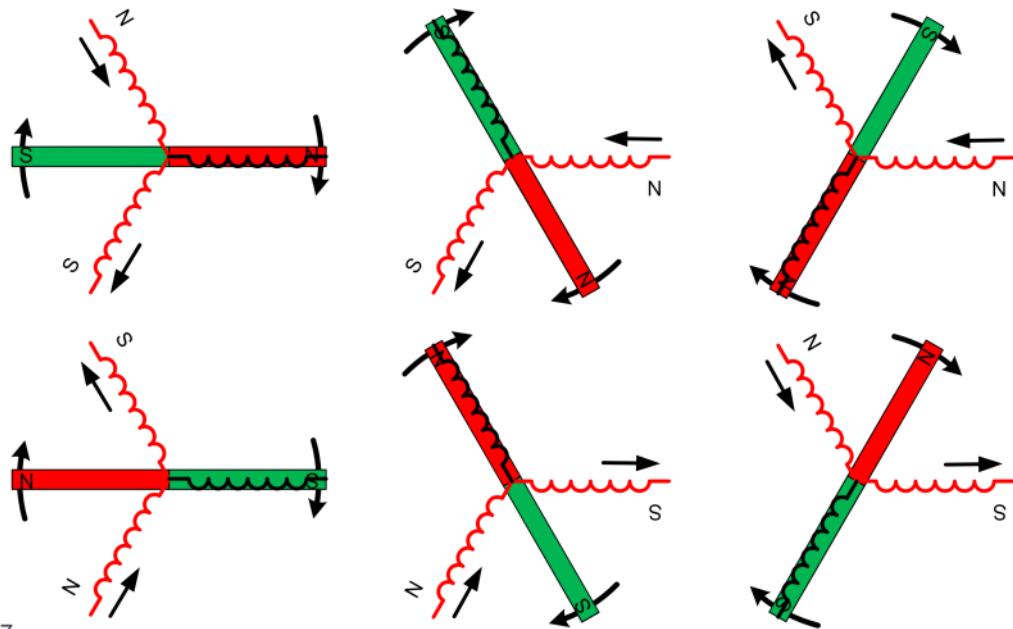
Anwendung PWM: Ansteuerung bürstenloser Motoren



16

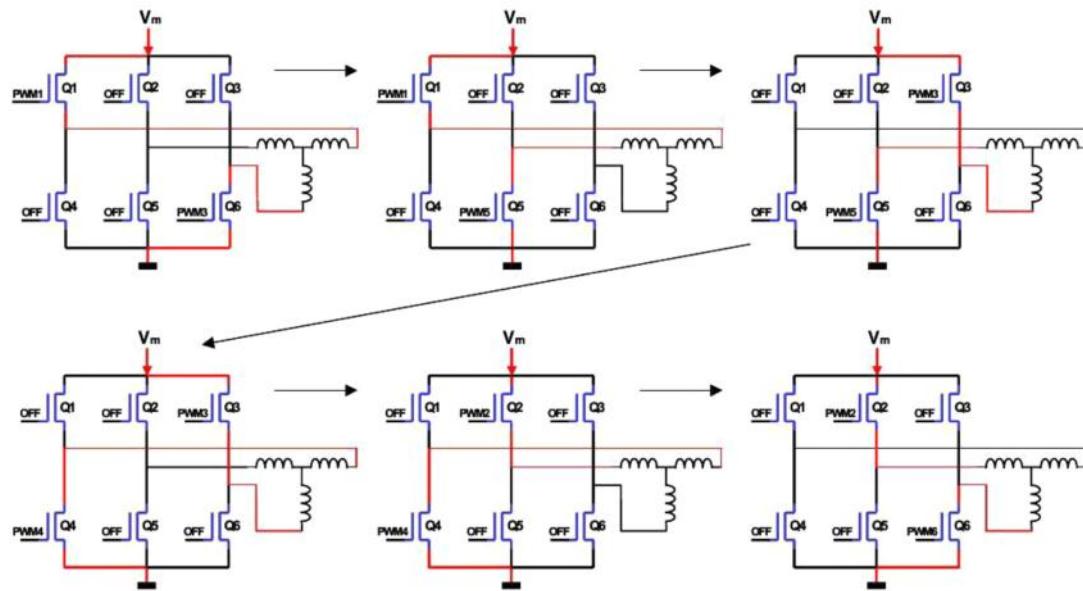
aus http://www.nxp.com/documents/application_note/AN10661.pdf

Anwendung PWM: Ansteuerung bürstenloser Motoren



17

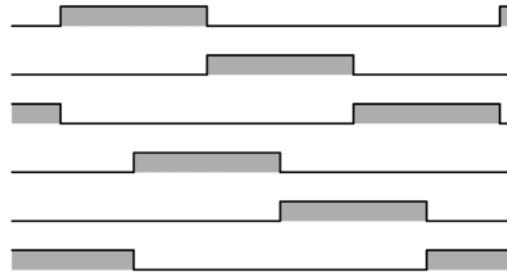
Anwendung PWM: Ansteuerung bürstenloser Motoren



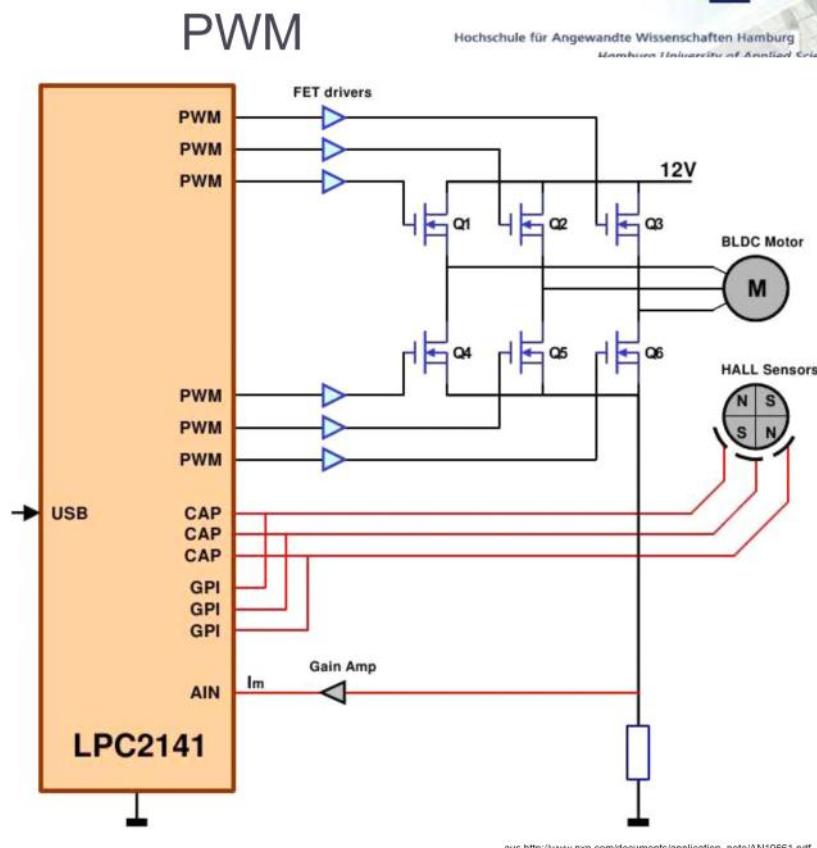
18

aus http://www.nxp.com/documents/application_note/AN10661.pdf

Anwendung PWM: Ansteuerung bürstenloser Motoren



Anwendung PWM: Ansteuerung bürstenloser Motoren



aus http://www.nxp.com/documents/application_note/AN10661.pdf

LPC2468 PWM: Eigenschaften

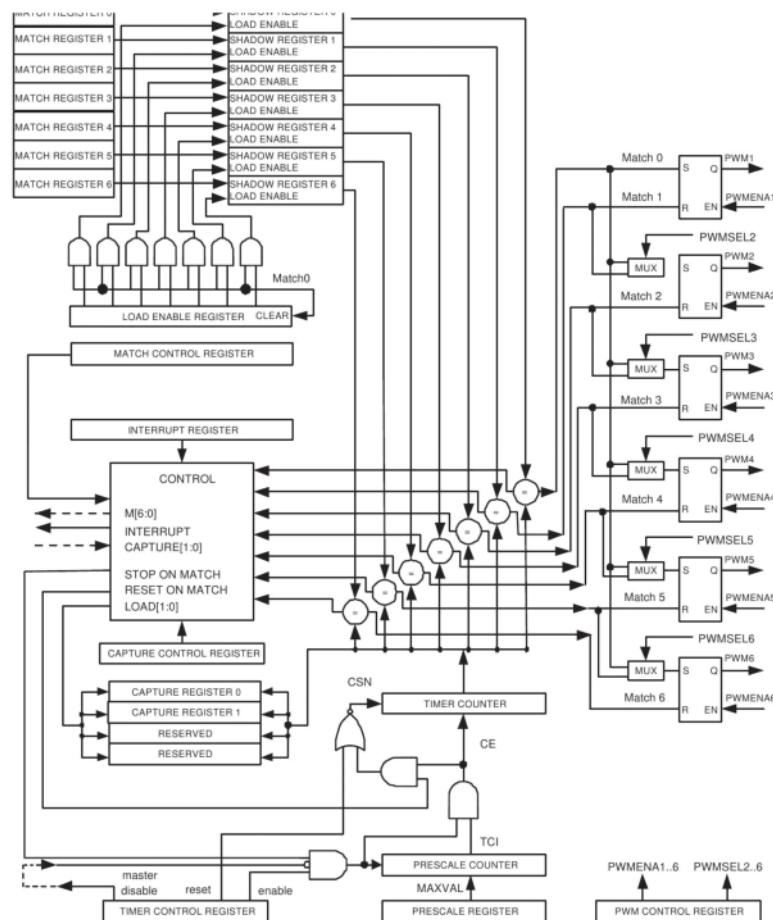
- ▶ 2 identische PWM-Komponenten
- ▶ Erweiterung der Timer/Counter-Komponenten
- ▶ Zwei Betriebsarten der Ausgänge:
 - ▶ Single edge controlled (bis zu 6 Ausgänge)
 - ▶ Double edge controlled (bis zu 3 Ausgänge)



LPC2468 PWM: Erweiterung gegenüber Timer/Counter

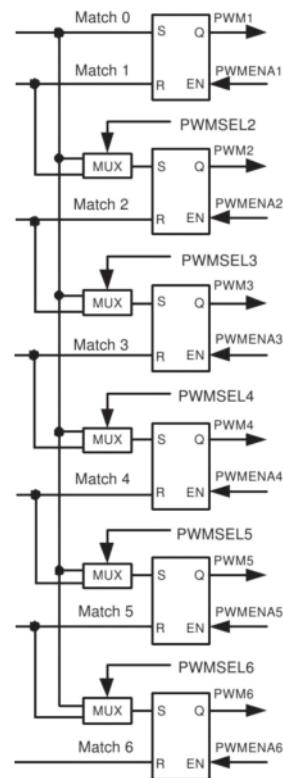
- ▶ Reset Timer/Counter statt auf 0 auf 1.
- ▶ Zusätzliche Steuerung für PWM-Ausgänge.
- ▶ 7 Match Register.
- ▶ Jedes Match-Register hat ein Shadow-Register.
- ▶ Zusätzliches Register zum Freigeben neuer Match-Werte.
- ▶ Freigegebene Werte werden erst mit nächstem Match 0 – Ereignis übernommen.

22



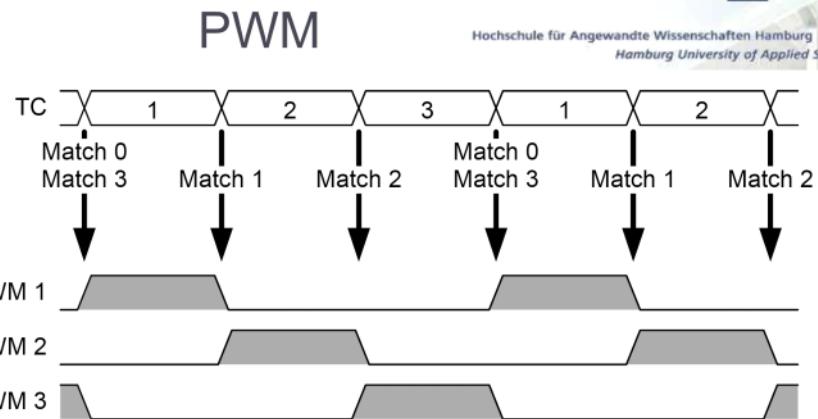
LPC2468 PWM: PWM-Ausgänge

- ▶ Jeder Ausgang wird durch RS-Flipflop gesteuert
- ▶ **Single edge controlled:**
 - ▶ Ausgang x wird aktiviert durch Match 0
 - ▶ Ausgang x wird zurückgesetzt durch Match x
- ▶ **Double edge controlled:**
 - ▶ Ausgang x wird aktiviert durch Match x-1
 - ▶ Ausgang x wird zurückgesetzt durch Match x
- ▶ Steuerbits:
 - ▶ **PWMENAx:**
 - 0: Ausgang gesperrt
 - 1: Ausgang freigegeben
 - ▶ **PWMSELx:**
 - 0: Single edge controlled
 - 1: Double edge controlled



CE WS12

Beispiel: Lauflicht



```

PCONP |= (1<<6);                                // Enable PWM1
PCLKSEL0 |= (1<<12);                             // Clock == CCLK (48 MHz)
PINSEL4 |= (1<<0) | (1<<2) | (1<<4);          // P2.0 ist PWM1.1, P2.1 ist PWM1.2, P2.2 ist PWM1.3
PWM1_PR = 48000000-1;                            // Prescaler, PWM-Takt = 1 Hz
PWM1_MCR = (1<<0) | (1<<1);                  // Interrupt on PWMMR0, Reset on PWMMR0
PWM1_PCR = (1<<2) | (1<<3) |                   // PWM1.2 double edge, PWM1.3 double edge,
    (1<<9) | (1<<10) | (1<<11);           // PWM1.1 PWM1.2 PWM1.3 enabled
PWM1_MR0 = 3;                                     // PWM-Period
PWM1_MR1 = 1;                                     // PWM-Match 1
PWM1_MR2 = 2;                                     // PWM-Match 2
PWM1_MR3 = 3;                                     // PWM-Match 3
PWM1_LER = (1<<0) | (1<<1) |                   // Latch Enable
    (1<<2) | (1<<3);

PWM1_TCR = (1<<0) | (1<<3);                  // Counter Enable, PWM Enable

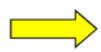
```

Computer Engineering WS 2012

Serielle Busse

HTM – SHF - SWR

Übersicht



- ▶ Einleitung
 - ▶ Kodierung, Dekodierung
 - ▶ Taktrückgewinnung
 - ▶ Galvanische Trennung
- ▶ RS232
- ▶ I2C, SPI

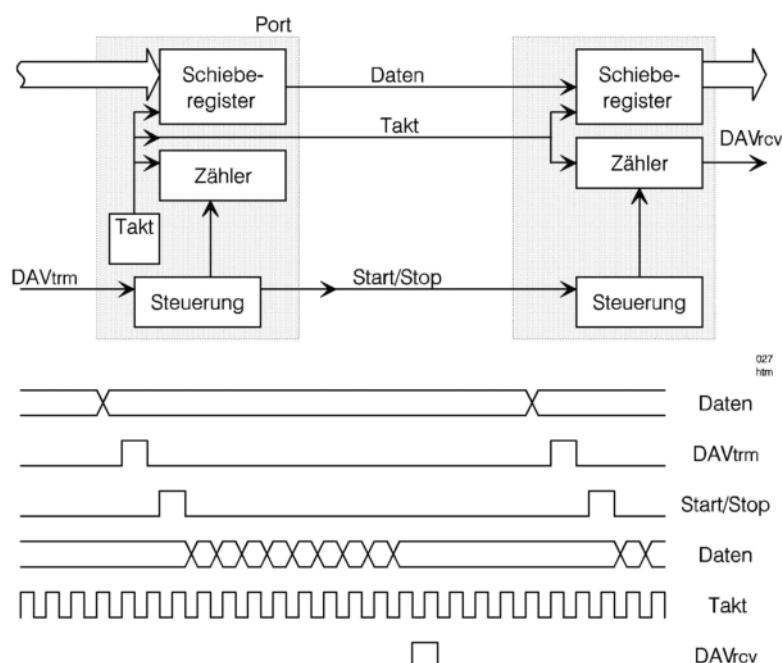
LPC 2468: Serielle Bussysteme

Name	Anzahl	Geschwindigkeit
RS232	4	300 Bit/s bis 4 MBit/s
I2C	3	bis 400kBit/s
CAN	2	25 kBit/s bis 1 MBit/s
I2S	1	bis 6,144 MBit/s
SPI	3	bis 24 MBit/s
USB 2.0	1	12 MBit/s
Ethernet	1	10 MBit/s, 100 MBit/s

Serielle Busse

Serielle Übertragung

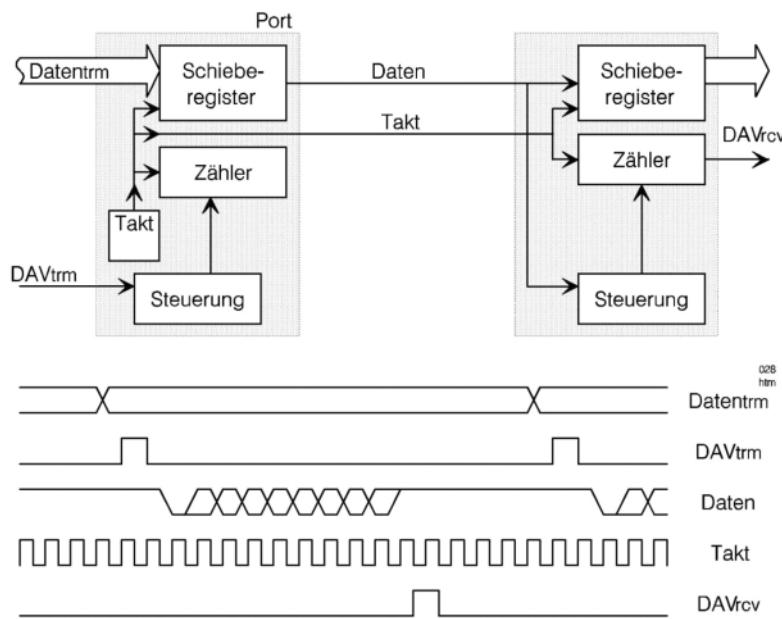
- Übertragungs-geschwindigkeit:
 - festgelegt durch Taktrate
 - Taktrate bestimmt die Anzahl Schritte pro Sekunde
 - = **Baudrate**
 - Übertragungs-verfahren:
 - **Bit: Synchron**
 - **Byte: Asynchron**
 - Implementierung:
 - **SPI**



CE WS12

Start-/Stopperkennung

- ▶ Spezielle Kennzeichnung von Datenanfang und – ende erforderlich
- ▶ z.B.: Start-/Stoppbits (RS232)
 - ▶ Ein Startbit, · z. B. == 0
 - ▶ Ein oder mehrere Stoppbits · z. B. == 1
- ▶ Achtung:
 - ▶ Erkennung ist nicht immer eindeutig!



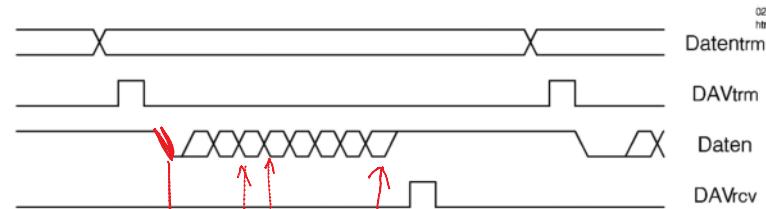
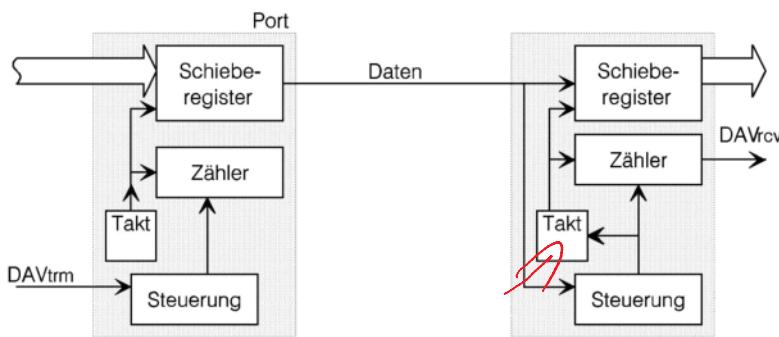
CE WS12

Taktrückgewinnung

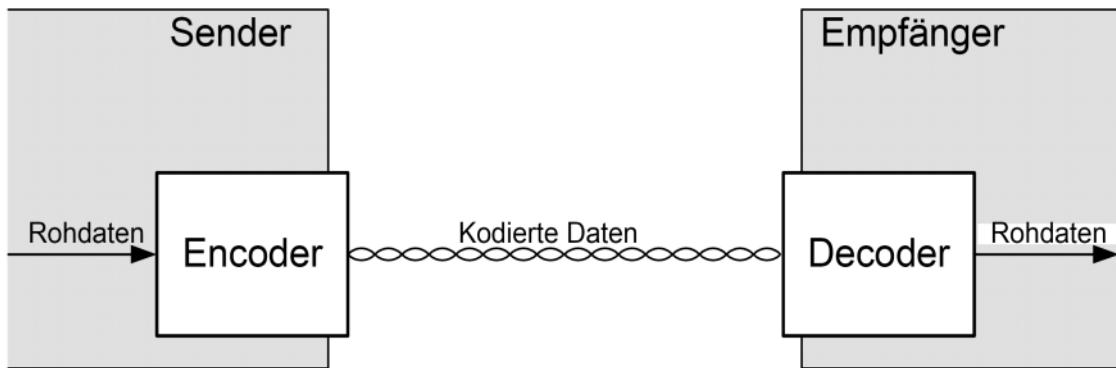
- ▶ Synchronisation mittels Flanken auf der Datenleitung
- ▶ Garantiert:
 - ▶ Fallende Flanke zu Beginn des Startbits

⇒ Referenzpunkt für Takt-synchronisation

- ▶ Erforderliche Genauigkeit des Taktgenerators?



Data Encoding / Decoding



► Optimierungskriterien:

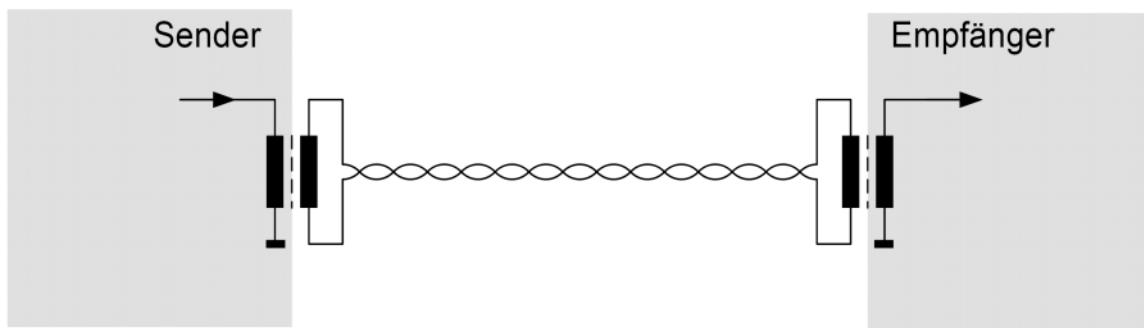
1. **Gute Taktrückgewinnung: Möglichst viele Flanken.**
2. **Geringe Bandbreite: Möglichst wenig zusätzliche Bits.**
3. **Einfache galvanische Trennung: Mittelwert des Signals konstant.**
4. **Vertauschung der Adern erlaubt.**

Galvanische Trennung von Sender und Empfänger

- ▶ **Ziel:**
Sender und Empfänger sollen **nicht leitend** miteinander verbunden sein.
- ▶ Erforderlich zur
 - ▶ **Vermeidung von elektromagnetischen Störungen.**
 - ▶ **Verbesserung der elektromagnetischen Verträglichkeit (EMV).**
 - ▶ **Vermeidung von „Brummschleifen“**
(z.B.: Messtechnik, Audioanwendungen).
 - ▶ **Erhöhung der Sicherheit** (z.B.: Medizintechnik).
- ▶ Realisierung: Übertrager, Optokoppler.

Galvanische Trennung von Sender und Empfänger

- Realisierung mit Übertrager:

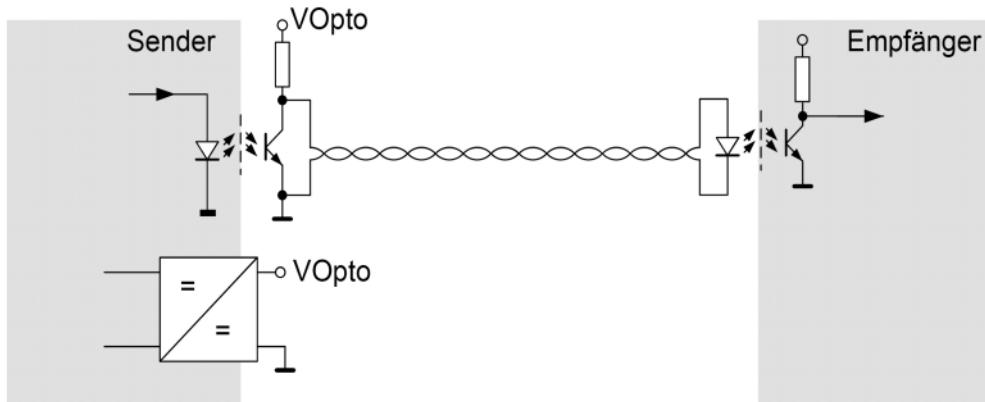


- Vorteile:

- Nachteile:

Galvanische Trennung von Sender und Empfänger

- Realisierung mit Optokoppler:



- Vorteile:

- Nachteile:

CE WS12

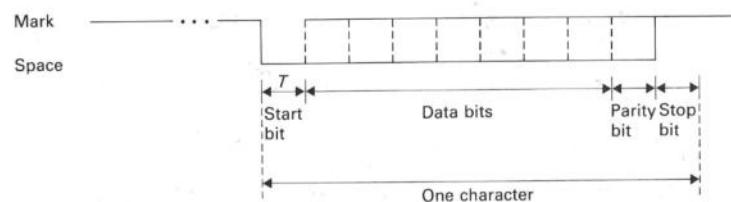
RS232

► Anwendung

- Standardisiert für Verbindung zwischen **Endgerät** (PC, Drucker, Terminal) und **Modem**
- Heute: Direkte Verbindung zwischen Endgeräte (**Nullmodem**)

► Eigenschaften:

- **Anzahl Daten:**
5 bis 8 Bits



- **1 Startbit**

- **1 bis 2 Stopbits**

- **Parität:**
keine,
gerade, ungerade

- **Datenraten, z.B.:**

300, 2400,

9600, 19200

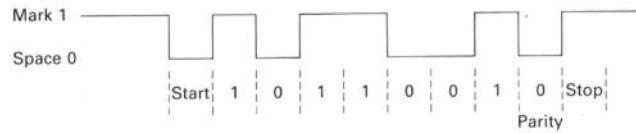
57600, 115200

- **Spannungspegel:**

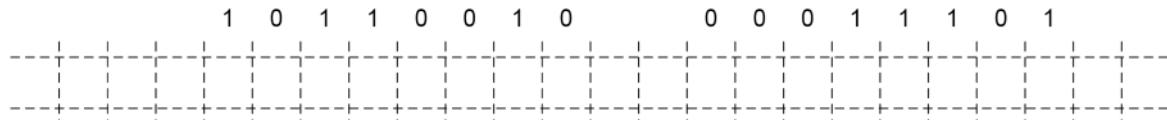
Mark (1) = -5 bis -15 Volt

Space (0) = +5 bis +15 Volt

Example: Letter M = ASCII \$4D = 1001101₂ (even parity)



Data Encoding / Decoding: RS232



1. **Taktrückgewinnung:**

2. **Bandbreite:**

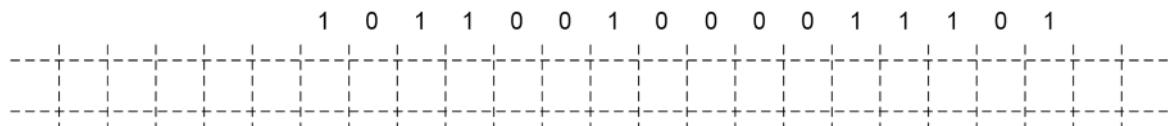
3. **Galvanische Trennung:**

4. **Vertauschung der Adern:**

CE WS12

Data Encoding / Decoding: **Manchester-Kodierung**

- **Kodierung mittels Flanken:** negative Flanke = 0, positive Flanke = 1



1. **Taktrückgewinnung:**

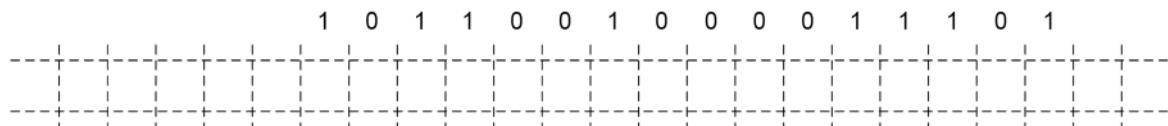
2. **Bandbreite:**

3. **Galvanische Trennung:**

4. **Vertauschung der Adern:**

Data Encoding / Decoding: Differentielle Manchester-Kodierung

- Kodierung mittels Flanken: Flanke = 0, keine Flanke = 1



1. Taktrückgewinnung:

2. Bandbreite:

3. Galvanische Trennung:

4. Vertauschung der Adern:

Übersicht

- ▶ Einleitung
 - ▶ Kodierung, Dekodierung
 - ▶ Taktrückgewinnung
 - ▶ Galvanische Trennung
- ▶ RS232
- ▶ I2C, SPI



Verbindung von Komponenten auf einer Leiterplatte:

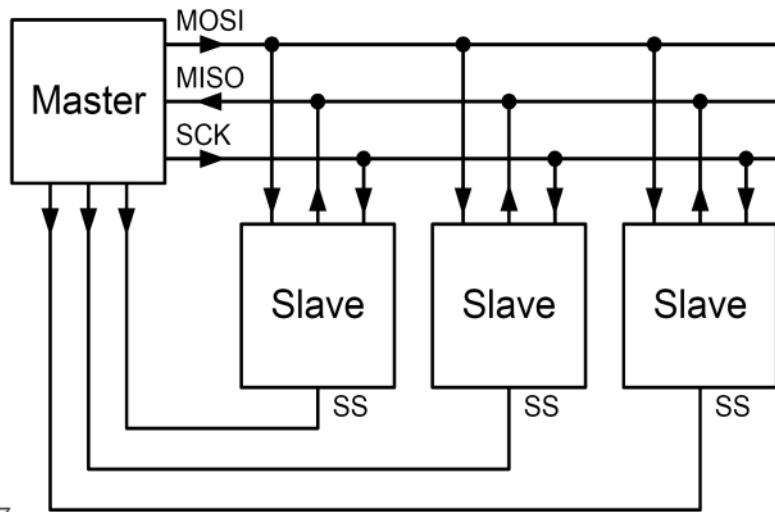
I₂C und SPI

	I ₂ C	SPI
Master	Multi Master	Single Master
Datenleitungen	2	≥ 4
Übertragung	Half Duplex	Full Duplex
Geschwindigkeit	400 kBit/s	24 MBit/s
Overhead	hoch	gering

CE WS12

Serial Peripheral Interface (SPI)

- ▶ Entwicklung der Fa. Motorola.
- ▶ Ziel: kostengünstiger Anschluss von Peripheriekomponenten.
- ▶ Wenig spezifiziert.

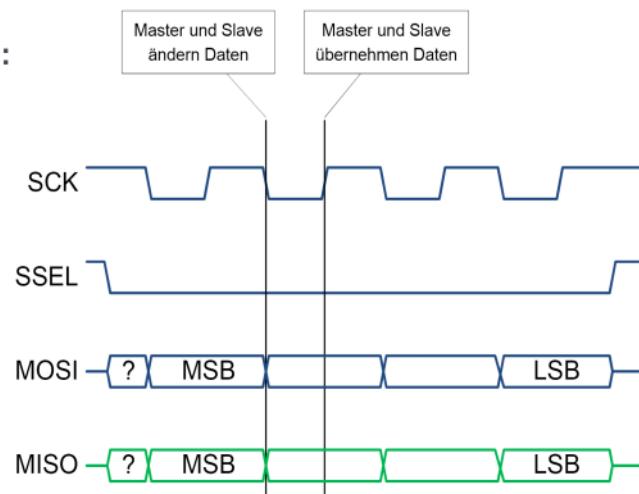


17

CE WS12

Serial Peripheral Interface (SPI)

- ▶ Synchrone Übertragung
- ▶ 4 verschiedene Betriebsarten:
 - ▶ positive oder negative aktive Flanke.
 - ▶ SCK in Ruhe Low oder High
 - ▶ Vollduplex-Übertragung



Implementierung im LPC2468: SSP0 und SSP1

- ▶ Eigenschaften:
 - ▶ Verschiedene Protokolle:
 - Motorola SPI
 - 4-wire Texas Instruments SSI
 - National Semiconductor Microwire
 - ▶ Master oder Slave Betrieb
 - ▶ 8 Frames FIFOs für Empfangs- und Senderichtung
 - ▶ Framelänge 4 bis 16 Bits
 - ▶ Direct Memory Access (DMA) Betrieb möglich
 - ▶ Maximale Taktfrequenz: PCLK/2 (bei $f_{SYS} = 48$ MHz: 24 MHz)

Implementierung im LPC2468: **Register**

Generic Name	Description
CR0	Control Register 0. Selects the serial clock rate, bus type, and data size.
CR1	Control Register 1. Selects master/slave and other modes.
DR	Data Register. Writes fill the transmit FIFO, and reads empty the receive FIFO.
SR	Status Register
CPSR	Clock Prescale Register
IMSC	Interrupt Mask Set and Clear Register
RIS	Raw Interrupt Status Register
MIS	Masked Interrupt Status Register
ICR	SSPICR Interrupt Clear Register
DMACR	DMA Control Register

Implementierung im LPC2468: Initialisierung

- ▶ Einstellungen in:
 - ▶ **PCONP**
 - Komponente einschalten.
 - ▶ **PCLKSELx**
 - Taktfrequenz für Komponente einstellen.
 - ▶ **PINSELx**
 - Sonderfunktionen der Pins aktivieren.
 - ▶ **SSP0CR0, SSP0CR1, SSP0CPSR**
 - Konfiguration der SPI-Schnittstelle

Implementierung im LPC2468: SSPn Control Register 0

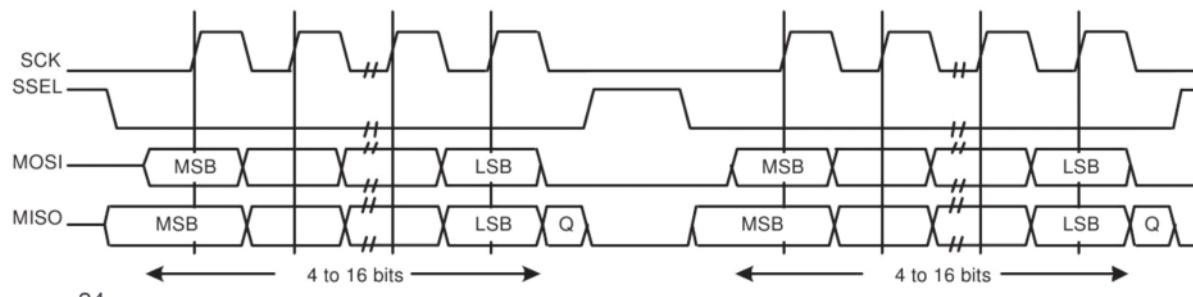
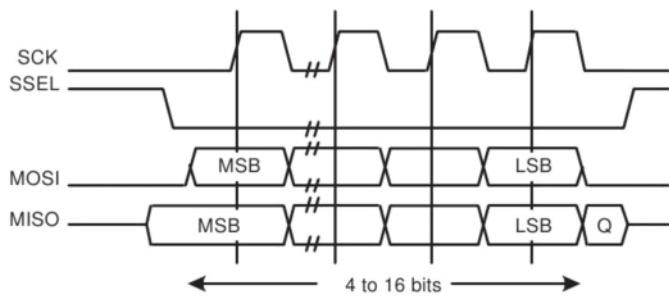
Bit	Symbol	Value	Description
3:0	DSS		Data Size Select. This field controls the number of bits transferred in each frame. Values 0000-0010 are not supported and should not be used.
		0011	4 bit transfer
		0100	5 bit transfer
		0101	6 bit transfer
		0110	7 bit transfer
		0111	8 bit transfer
		1000	9 bit transfer
		1001	10 bit transfer
		1010	11 bit transfer
		1011	12 bit transfer
		1100	13 bit transfer
		1101	14 bit transfer
		1110	15 bit transfer
		1111	16 bit transfer
5:4	FRF		Frame Format.
		00	SPI
		01	TI
		10	Microwire
		11	This combination is not supported and should not be used.
6	CPOL		Clock Out Polarity. This bit is only used in SPI mode.
		0	SSP controller maintains the bus clock low between frames.
		1	SSP controller maintains the bus clock high between frames.
7	CPHA		Clock Out Phase. This bit is only used in SPI mode.
		0	SSP controller captures serial data on the first clock transition of the frame, that is, the transition away from the inter-frame state of the clock line.
		1	SSP controller captures serial data on the second clock transition of the frame, that is, the transition back to the inter-frame state of the clock line.
15:8	SCR		Serial Clock Rate. The number of prescaler-output clocks per bit on the bus, minus one. Given that CPSDVSR is the prescale divider, and the APB clock PCLK clocks the prescaler, the bit frequency is $PCLK / (CPSDVSR \times [SCR+1])$.

Implementierung im LPC2468: SSPn Control Register 1

Bit	Symbol	Value	Description
0	LBM	0	Loop Back Mode. During normal operation.
		1	Serial input is taken from the serial output (MOSI or MISO) rather than the serial input pin (MISO or MOSI respectively).
1	SSE	0	SSP Enable. The SSP controller is disabled.
		1	The SSP controller will interact with other devices on the serial bus. Software should write the appropriate control information to the other SSP registers and interrupt controller registers, before setting this bit.
2	MS	0	Master/Slave Mode. This bit can only be written when the SSE bit is 0. The SSP controller acts as a master on the bus, driving the SCLK, MOSI, and SSSEL lines and receiving the MISO line.
		1	The SSP controller acts as a slave on the bus, driving MISO line and receiving SCLK, MOSI, and SSSEL lines.
3	SOD		Slave Output Disable. This bit is relevant only in slave mode (MS = 1). If it is 1, this blocks this SSP controller from driving the transmit data line (MISO).
7:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

CE WS12

Implementierung im LPC2468:
CPOL = 0 und CPHA = 0



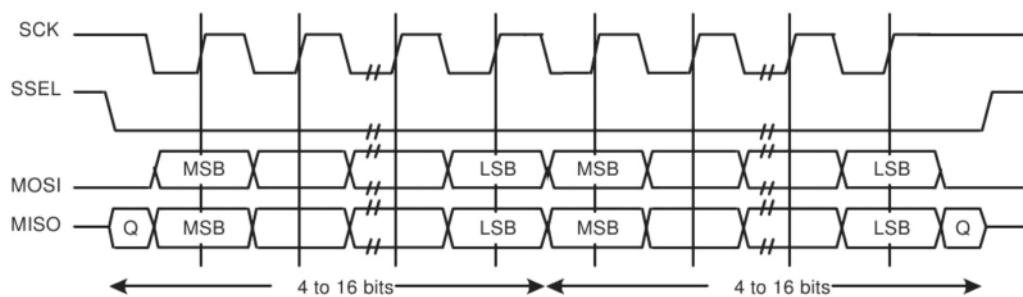
24

CE WS12

Implementierung im LPC2468:

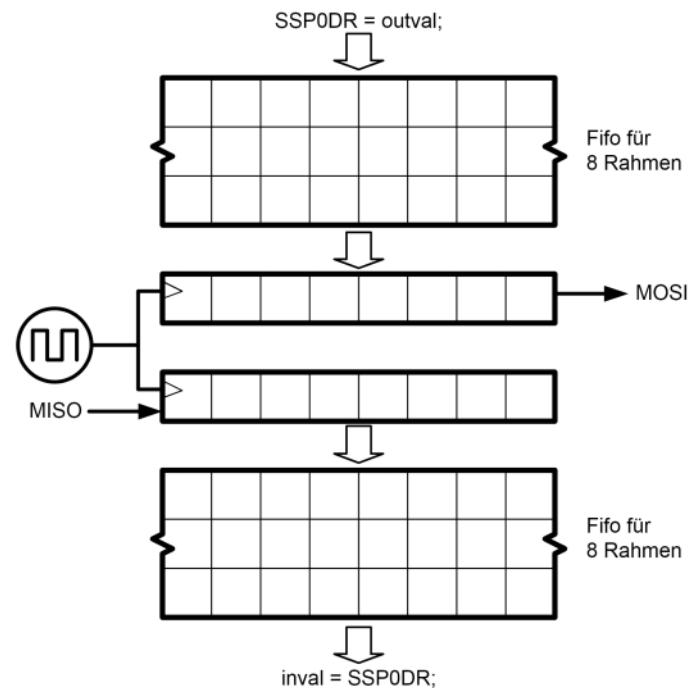
CPOL = 1 und CPHA = 1

- ▶ Ruhezustand von SCK: High
- ▶ Aktive Flanke: Zweite Flanke nach Aktivierung von SSEL



CE WS12

Implementierung im LPC2468:



26

CE WS12

Implementierung im LPC2468:

Bit	Symbol	Description
0	TFE	Transmit FIFO Empty. This bit is 1 if the Transmit FIFO is empty, 0 if not.
1	TNF	Transmit FIFO Not Full. This bit is 0 if the Tx FIFO is full, 1 if not.
2	RNE	Receive FIFO Not Empty. This bit is 0 if the Receive FIFO is empty, 1 if not.
3	RFF	Receive FIFO Full. This bit is 1 if the Receive FIFO is full, 0 if not.
4	BSY	Busy. This bit is 0 if the SSPn controller is idle, or 1 if it is currently sending/receiving a frame and/or the Tx FIFO is not empty.
7:5	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.

Computer Engineering WS 2012

Speicher

HTM – SHF - SWR

Übersicht

- ▶ Random Access Memory (RAM)
 - ▶ Statisch
 - ▶ Dynamisch
- ▶ Flashspeicher
- ▶ SPI-Flashspeicher

Klassifizierung

- ▶ Flüchtige Speicher:

- ▶ Verlieren Inhalt, wenn Spannung abgeschaltet wird
- ▶ Beispiele:
 - ▶ Statisches RAM SRAM
 - ▶ Dynamisches RAM DRAM

- ▶ Nicht flüchtige Speicher:

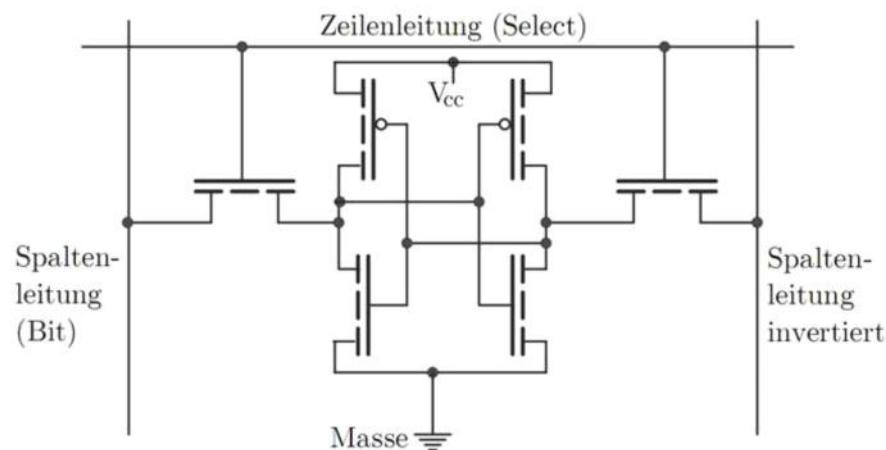
- ▶ Speicherinhalt bleibt auch nach Abschalten der Versorgungsspannung erhalten
- ▶ Beispiele:
 - ▶ Nicht lösbar: PROM
 - ▶ Lösbar: EPROM, Flash
 - ▶ Batteriegepuffertes RAM NVRAM

Statisches RAM (SRAM)

- ▶ Wahlfreier Zugriff auf alle Speicheradressen
→ „Random Access Memory“.
- ▶ Speicherelemente mit Flipflops realisiert:
 - ▶ Bei anliegender Versorgungsspannung bleibt Zustand dauerhaft erhalten.
- ▶ Sehr schnell
- ▶ Hoher Platzbedarf, dadurch teuer.
- ▶ Datenerhaltung benötigt sehr wenig Energie:
 - ▶ Mit Pufferung über Batterie:
Speicherung bis zu 10 Jahre möglich
- ▶ Verwendung:
 - ▶ Schnelle Speicher, Caches
 - ▶ Register

Statisches RAM (SRAM)

- ▶ Speicherung mit Flipflops
- ▶ z.B.: Aufbau mit 6 Transistoren

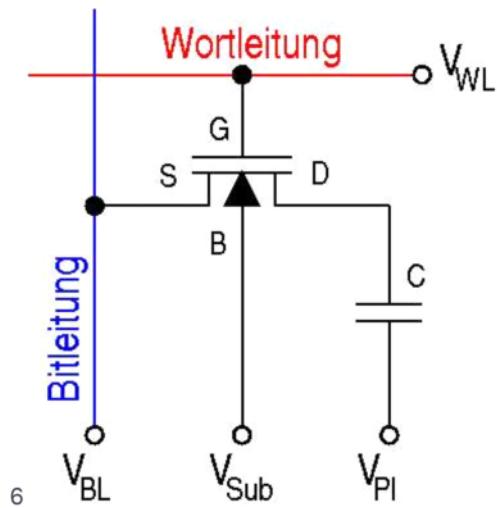


5

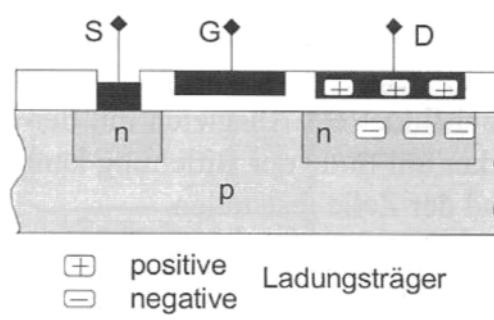
CE WS12

Dynamisches RAM (DRAM)

- ▶ Speicherung in Kondensatoren
- ▶ z.B.: geladen: 1, entladen: 0
- ▶ Kondensator entlädt sich mit der Zeit
- ▶ Refresh notwendig.



aus: Wikipedia (de)



aus: Bähring, Mikrorechnertechnik

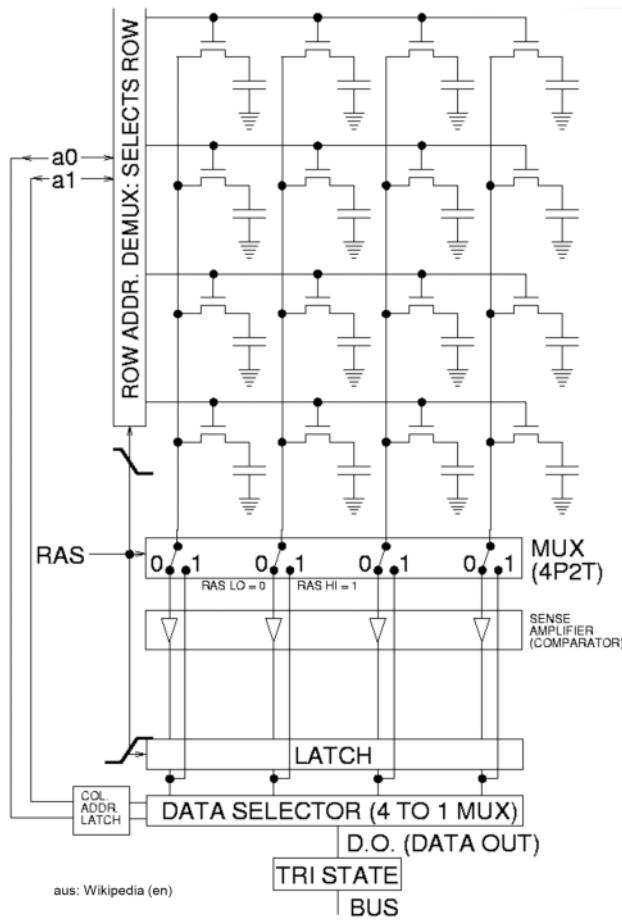


Speicher

DRAM: Prinzip

- ▶ Aufteilung der Adressen in
 - ▶ Spaltenadresse:
 - ▶ Auswahl mit CAS „Column Address Select“
 - ▶ Reihenadresse:
 - ▶ Auswahl mit RAS „Row Address Select“
- ▶ Lesen:
 - ▶ Übertragen einer vollständigen Reihe ins Latch
 - ▶ Dabei Zerstörung der Daten:
 - ▶ Zurückschreiben erforderlich.
 - ▶ Zugriff auf Bits im Latch sehr schnell:
 - ▶ Unterstützt Burst-Zugriff.

7

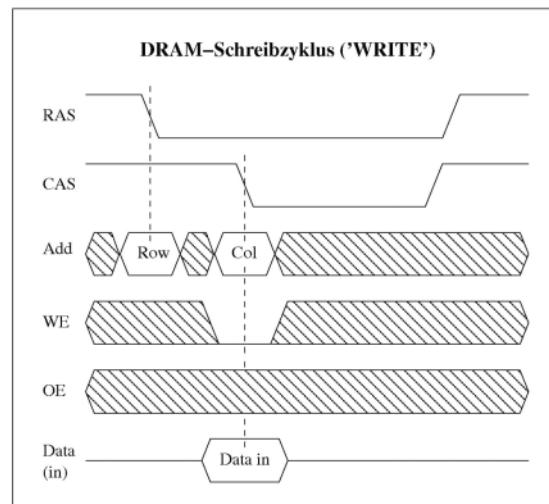
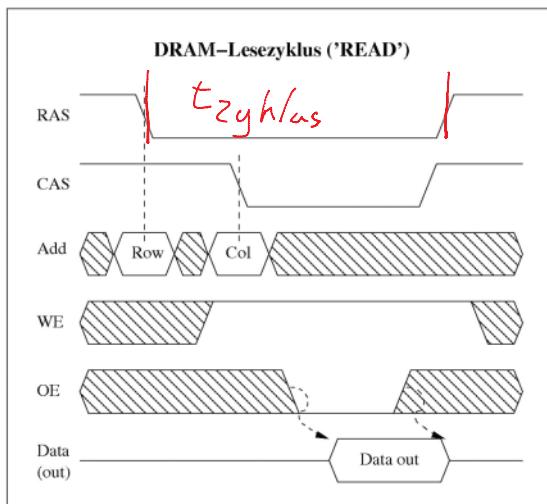


aus: Wikipedia (en)

CE WS12

DRAM: Lese- und Schreibzyklus

- ▶ Adressleitungen liefern nacheinander Zeilen- und Spaltenadressen.
- ▶ Zykluszeit beachten:
Zusätzliche Zeit zum Zurückschreiben der Zeile notwendig.



DRAM: Weiterentwicklungen

- ▶ **Fast Page Mode (FPM)**
 - ▶ Zeile bleibt aktiv, es muss nur noch die Spalte gesetzt werden:
 - ▶ Mehrfaches aktivieren von CAS: Burst Mode.
- ▶ **Extended Data Output (EDO)**
 - ▶ Ausgang bleibt aktiv, bis neue Daten vorliegen.
 - ▶ Im Hintergrund kann schon der nächste Zugriff bearbeitet werden.
- ▶ **Synchronous DRAM (SDRAM)**
 - ▶ Zusätzlicher Systemtakt:
 - ▶ Alle Komponenten arbeiten synchron.
- ▶ **Double Data Rate Synchronous DRAM (DDR-SDRAM)**
 - ▶ Übertragung bei steigender und fallender Flanke des Systemtakts
 - ▶ Dadurch doppelter Durchsatz möglich

Übersicht

- ▶ Random Access Memory (RAM)
 - ▶ Statisch
 - ▶ Dynamisch
- ▶ Flashspeicher
- ▶ SPI-Flashspeicher

CE WS12

Technologien

- Maskenprogrammiert (Read Only Memory, **ROM**)
 - Programmierung erfolgt bei Herstellung.
 - Einfach, kostengünstig.
 - Kann nicht mehr verändert werden.
 - Gut geeignet für Großserienprodukte
- Benutzerprogrammiert (Programmable Read Only Memory, **PROM**)
 - Benutzer kann Speicher einmalig programmieren.
 - Wegen Programmierlogik Aufwand höher als ROM.
- Löschbar (Erasable Programmable Read Only Memory, **EPROM**)
 - Benutzer kann Speicher mehrfach programmieren.
 - Programmierung nur möglich von „1“ nach „0“.
 - „1“ nur durch Löschen des gesamten Speichers.
 - Löschen mit UV-Licht oder elektrisch (**FLASH**)
 - Geringe garantierte Schreibzyklenzahl (z.B.: 10.000).
- Elektrisch löschbar (Electrically Erasable Programmable Read Only Memory, **EEPROM**)
 - Ermöglicht byteweises Löschen.
 - Hoher Logikaufwand, teuer.
 - Höhere garantierte Schreibzyklenzahl (z.B.: 1.000.000)
 - Größerer erlaubter Temperaturbereich zum Programmieren.
 - Gut für kleine Datenmengen, z.B.: Konfigurationsdaten Microcontroller

Flash-Speicher

Performance

- Flash-Speicher ist deutlich langsamer als RAM.
- Wird schnell zum Flaschenhals in einem Microcomputersystem.

Bei niedriger Taktfrequenz:

- Programme können direkt aus dem Flash heraus ausgeführt werden:
XIP (eXecute in Place)
- Beschleunigung:
 - Verwendung größerer Wortbreiten:
 - z.B.: 128-Bit bei einer 32-Bit CPU.
Pro Speicherzugriff werden gleichzeitig 4 * 32-Bit Worte gelesen.
 - Verwendung von Cache Speicher

Bei höherer Taktfrequenz:

- Programm wird vom Flash ins RAM kopiert und von dort ausgeführt.

CE WS12

Flash-Speicher Architekturen

► NOR-Architektur

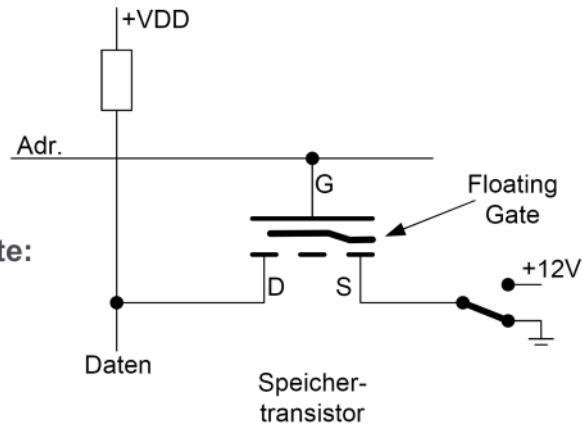
- Inhalt kann wahlfrei adressiert werden.
- Lesegeschwindigkeit höher.
- Gut für Programmspeicher in Mikrocomputersystemen.

► NAND-Architektur

- Einfachere Struktur, dadurch kleinere Bitzellen:
 - höhere Kapazitäten möglich,
 - preiswerter.
- Zugriff nur in Burst von z.B. 512 Bytes möglich.
- Schreib- und Löschtätigkeiten sind wesentlich schneller.
- Löscheinheiten.
- Programme können nicht direkt aus NAND-Speicher ausgeführt werden.
- Gut für Massenspeicher: USB-Stick Memory-Card, Solid State Disk.
- Können bereits bei Auslieferung verstreute „Bad Blocks“ haben.

Aufbau von NOR-Bausteinen

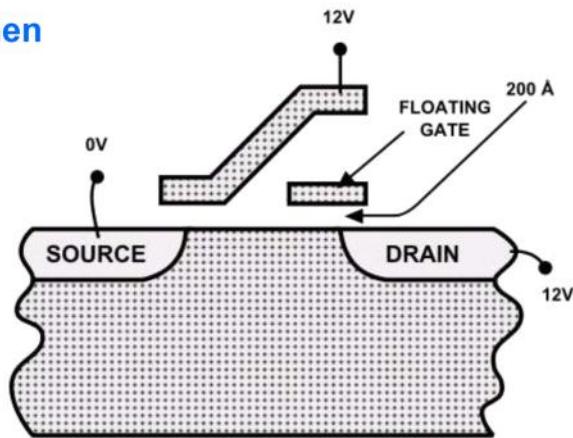
- ▶ Datenspeicherung durch Ladungsträger auf Floating Gate
- ▶ Lesen durch Aktivierung des Gates:
 - ▶ Keine Ladung auf Floating Gate:
 - ▶ Transistor schaltet durch
 - ▶ Datenleitung = 0 V
 - ▶ Ladung auf Floating Gate:
 - ▶ Transistor sperrt
 - ▶ Datenleitung = +VDD



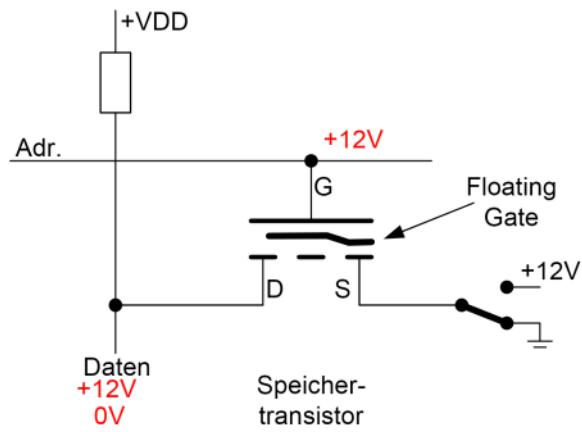
Programmieren von NOR-Bausteinen

Channel Hot Electron Injection

- ▶ Spannung am Steuergate macht Tunnel leitend.
- ▶ Zwischen Source und Drain wird eine hohe Spannung geschaltet.
- ▶ Dadurch hohe Beschleunigung der Elektronen:
 - ▶ Einige Elektronen gelangen dabei auf das Floating Gate.
- ▶ Diese negative Ladung sorgt dann im Normalbetrieb dafür, dass der Tunnel immer gesperrt ist.



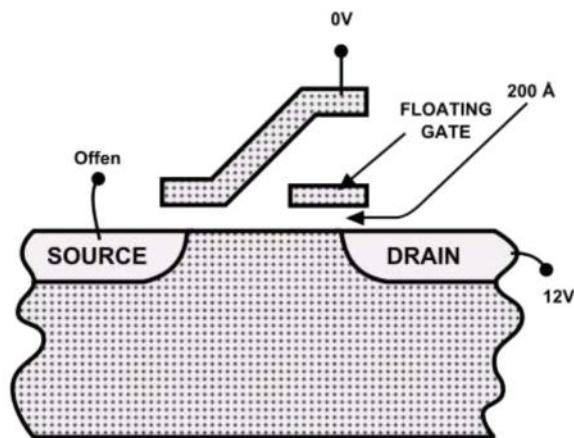
Programmieren von NOR-Bausteinen



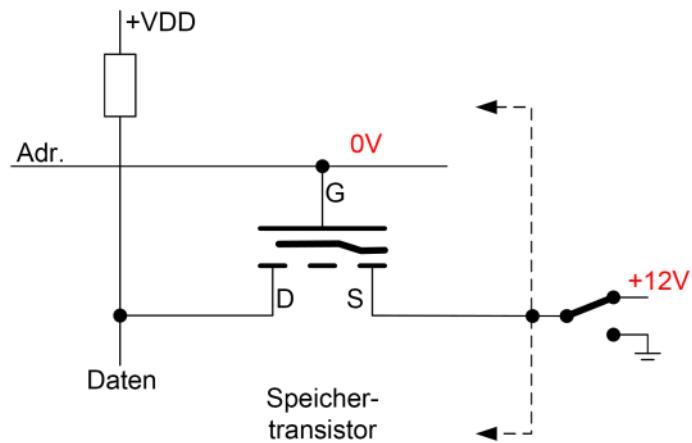
Löschen von NOR-Bausteinen

Fowler-Nordheim tunneling

- ▶ Quantenmechanischer Effekt:
 - ▶ Elektronen „tunneln“ vom Floating Gate zum Drain
- ▶ Negative Ladung auf dem Gate wird abgebaut.
- ▶ Damit kann im Normalbetrieb wieder Strom zwischen Source und Drain fließen.



Löschen von NOR-Bausteinen



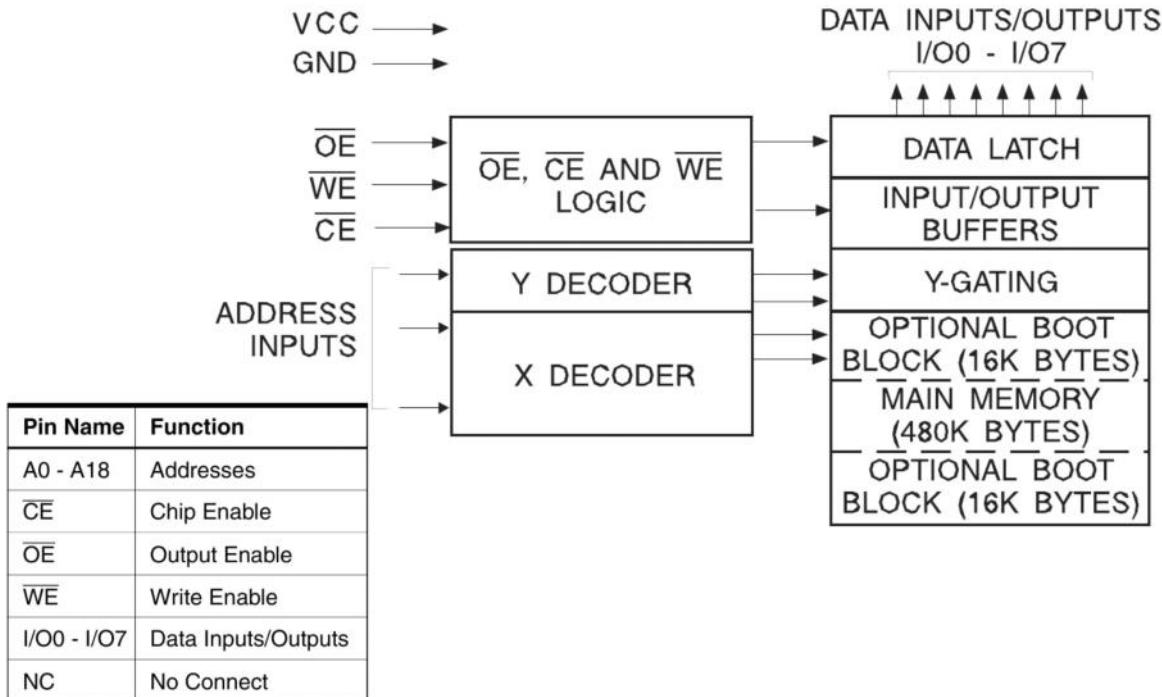
Beispiel: AT29LV040A (8 * 512kBits)

- ▶ Features
 - ▶ Single Voltage, Range 3V to 3.6V Supply
 - ▶ Software Protected Programming
 - ▶ Fast Read Access Time – 150 ns
 - ▶ Low Power Dissipation
 - 15 mA Active Current
 - 50 µA CMOS Standby Current
 - ▶ Sector Program Operation
 - Single Cycle Reprogram (Erase and Program)
 - 2048 Sectors (256 Bytes/Sector)
 - Internal Address and Data Latches for 256 Bytes
 - ▶ Two 16K Bytes Boot Blocks with Lockout
 - ▶ Fast Sector Program Cycle Time – 20 ms Max
 - ▶ Internal Program Control and Timer
 - ▶ DATA Polling for End of Program Detection
 - ▶ Typical Endurance > 10,000 Cycles

19

CE WS12

Beispiel: AT29LV040A (8 * 512kBits)



Übersicht

- ▶ Random Access Memory (RAM)
 - ▶ Statisch
 - ▶ Dynamisch
- ▶ Flashspeicher
- ▶ SPI-Flashspeicher



CE WS12

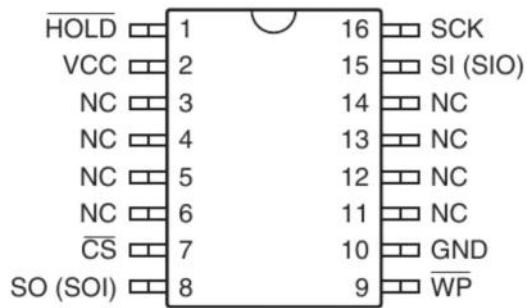
Beispiel: AT25DF641

- ▶ **Eigenschaften**
 - ▶ **Serial Peripheral Interface (SPI) Compatible**
 - ▶ **Very High Operating Frequencies (85 MHz for SPI)**
 - ▶ **Flexible, Optimized Erase Architecture for Code + Data Storage Applications**
 - Uniform 4-Kbyte Block Erase
 - Uniform 32-Kbyte Block Erase
 - Uniform 64-Kbyte Block Erase
 - Full Chip Erase
 - ▶ **Individual Sector Protection with Global Protect/Unprotect Feature**
 - 128 Sectors of 64-Kbytes Each
 - ▶ **Hardware Controlled Locking of Protected Sectors via WP Pin**
 - ▶ **Sector Lockdown : Make Any Combination of 64-Kbyte Sectors Permanently Read-Only**
 - ▶ **128-Byte Programmable OTP Security Register**
 - ▶ **Flexible Programming: Byte/Page Program (1 to 256 Bytes)**
 - ▶ **Fast Program and Erase Times**
 - 1.0 ms Typical Page Program (256 Bytes) Time
 - 50 ms Typical 4-Kbyte Block Erase Time
 - 250 ms Typical 32-Kbyte Block Erase Time
 - 400 ms Typical 64-Kbyte Block Erase Time
 - ▶ **Program and Erase Suspend/Resume**
 - ▶ **Automatic Checking and Reporting of Erase/Program Failures**
 - ▶ **Endurance: 100,000 Program/Erase Cycles**
 - ▶ **Data Retention: 20 Years**

22

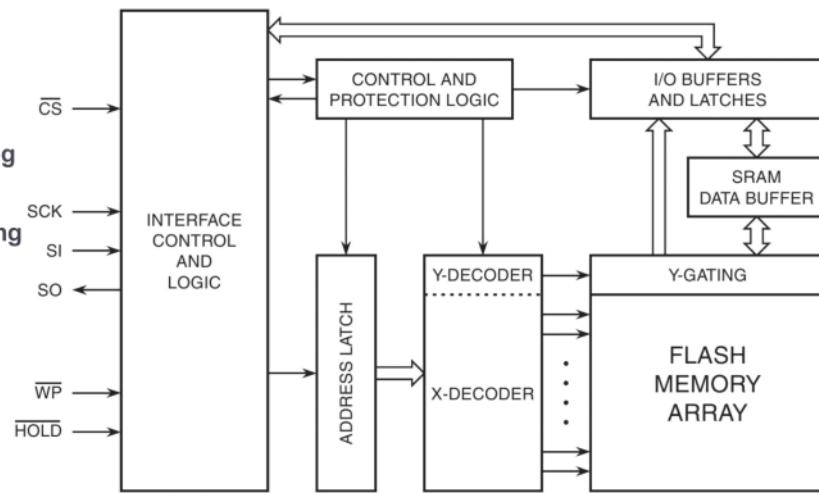
Beispiel: AT25DF641

Anschlüsse



Blockschaltbild SPI-Flashspeicher

- ▶ **SCK:**
Serieller Takt
- ▶ **SI:**
Serieller Dateneingang
- ▶ **SO:**
Serieller Datenausgang
- ▶ **CS:**
Chip select
- ▶ **WP:**
Write Protect
- ▶ **Hold:**
Einfrieren des
Datentransfers



Flash-Speicher

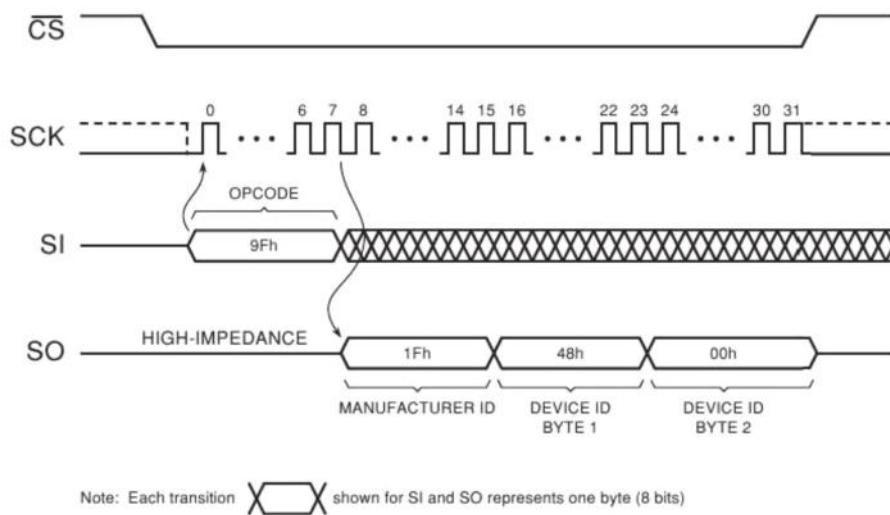
Beispiel: AT25DF641

Kommandos

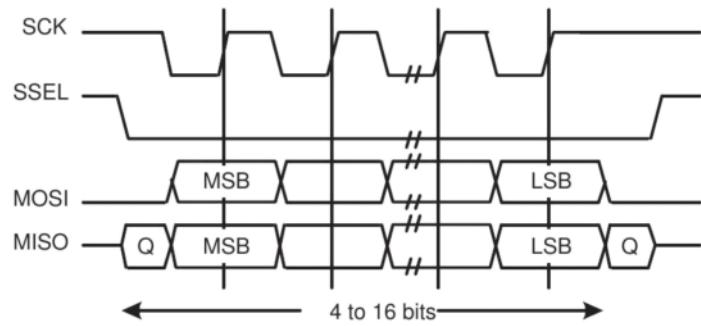
Command	Opcode	Clock Frequency	Address Bytes	Dummy Bytes	Data Bytes
Read Commands					
Read Array	1Bh 0001 1011	Up to 100 MHz	3	2	1+
	0Bh 0000 1011	Up to 85 MHz	3	1	1+
	03h 0000 0011	Up to 50 MHz	3	0	1+
Dual-Output Read Array	3Bh 0011 1011	Up to 85 MHz	3	1	1+
Program and Erase Commands					
Block Erase (4-KBytes)	20h 0010 0000	Up to 100 MHz	3	0	0
Block Erase (32-KBytes)	52h 0101 0010	Up to 100 MHz	3	0	0
Block Erase (64-KBytes)	D8h 1101 1000	Up to 100 MHz	3	0	0
	60h 0110 0000	Up to 100 MHz	0	0	0
Chip Erase	C7h 1100 0111	Up to 100 MHz	0	0	0
Byte/Page Program (1 to 256 Bytes)	02h 0000 0010	Up to 100 MHz	3	0	1+
Dual-Input Byte/Page Program (1 to 256 Bytes)	A2h 1010 0010	Up to 100 MHz	3	0	1+
Program/Erase Suspend	B0h 1011 0000	Up to 100 MHz	0	0	0
Program/Erase Resume	D0h 1101 0000	Up to 100 MHz	0	0	0
Protection Commands					
Write Enable	06h 0000 0110	Up to 100 MHz	0	0	0
Write Disable	04h 0000 0100	Up to 100 MHz	0	0	0
Protect Sector	36h 0011 0110	Up to 100 MHz	3	0	0
Unprotect Sector	39h 0011 1001	Up to 100 MHz	3	0	0
Global Protect/Unprotect	Use Write Status Register Byte 1 Command				
Read Sector Protection Registers	3Ch 0011 1100	Up to 100 MHz	3	0	1+
Security Commands					
Sector Lockdown	33h 0011 0011	Up to 100 MHz	3	0	1
Freeze Sector Lockdown State	34h 0011 0100	Up to 100 MHz	3	0	1
Read Sector Lockdown Registers	35h 0011 0101	Up to 100 MHz	3	0	1+
Program OTP Security Register	9Bh 1001 1011	Up to 100 MHz	3	0	1+
Read OTP Security Register	77h 0111 0111	Up to 100 MHz	3	2	1+
Status Register Commands					
Read Status Register	05h 0000 0101	Up to 100 MHz	0	0	1+
Write Status Register Byte 1	01h 0000 0001	Up to 100 MHz	0	0	1
Write Status Register Byte 2	31h 0011 0001	Up to 100 MHz	0	0	1
Miscellaneous Commands					
Reset	F0h 1111 0000	Up to 100 MHz	0	0	1
Read Manufacturer and Device ID	9Fh 1001 1111	Up to 85 MHz	0	0	1 to 4
Deep Power-Down	B8h 1011 1001	Up to 100 MHz	0	0	0
Resume from Deep Power-Down	ABh 1010 1011	Up to 100 MHz	0	0	0

Beispiel: AT25DF641

Kommando: „Read Manufacturer and Device ID“



SPI-Flashspeicher: Auswahl der SPI-Betriebsart in der CPU



SPI Frame Format with CPOL = 1 and CPHA = 1

Beispiel: AT25DF641

Kommando: „Read Manufacturer and Device ID“

```
GPIO0_IOCLR = (1<<16); //SSEL

SSP0DR = 0x9f;           //Opcode
SSP0DR = 0;              //Send 3 dummy bytes
SSP0DR = 0;
SSP0DR = 0;

id=0;
for( j=0; j<4; j++ ){
    while( (SSP0SR & (1<<2)) == 0 ){
    }
    id = (id<<8) | SSP0DR;
}

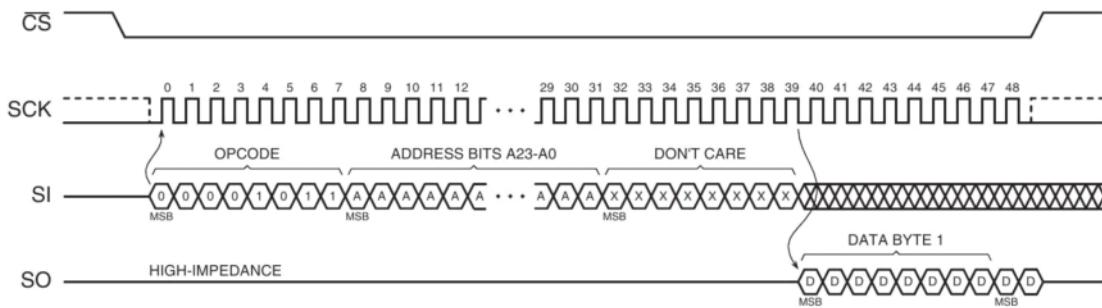
GPIO0_IOSET = (1<<16); //SSEL

printf( "id=%08X\n", id & 0xfffffff ); //first byte unused
```

Daten Lesen

- ▶ Unterschiedliche Kommandos in Abhängigkeit von der verwendeten Taktfrequenz und Übertragungsart:
 - ▶ Unterschiedliche Anzahl von „Don't care“-Bytes (Wartezeit).
 - ▶ Single- und Dual-Output (Datenausgabe parallel über SI und SO).
- ▶ Kommando und Adresse müssen nur einmal übertragen werden.
- ▶ Danach beliebige Anzahl Daten.

Read Array – 0Bh Opcode



Flash-Speicher

Löschen

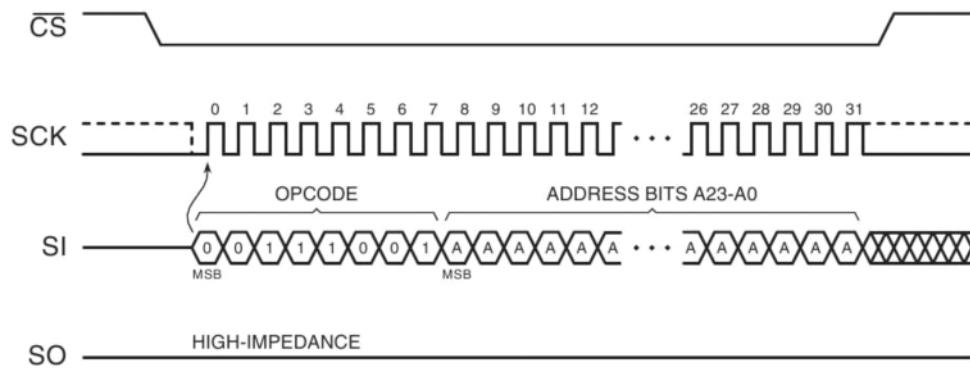
- ▶ Löschen kann nur blockweise erfolgen.
- ▶ Blockgröße variabel:
 - ▶ 128 Blöcke à 64 kB oder
 - ▶ 256 Blöcke à 32 kB oder
 - ▶ 2048 Blöcke à 4 kB
- ▶ Ablauf:
 1. **Unprotect Sectors:**
 - Write Enable
 - Unprotect
 2. **Erase Sectors:**
 - Write Enable
 - Erase Sectors
 3. **Warten:**
 - Read Status Register und prüfe RDY/BSY

Internal Sectoring for Protection, Lockdown, and Suspend Functions				64KB Block Erase (D8h Command)	32KB Block Erase (52h Command)	Block Erase (20h Command)	Block Address Range
64KB (Sector 127)	32KB	4KB	7FFFFFh – 7FF000h				
		4KB	7FFF00h – 7FE000h				
		4KB	7FDFFh – 7FD000h				
		4KB	7FCFFFFh – 7FC000h				
		4KB	7FBFFFFh – 7FB000h				
		4KB	7FAFFFFh – 7FA000h				
		4KB	7F9FFFFh – 7F9000h				
		4KB	7F8FFFFh – 7F8000h				
		4KB	7F7FFFFh – 7F7000h				
		4KB	7F6FFFFh – 7F6000h				
		4KB	7F5FFFFh – 7F5000h				
		4KB	7F4FFFFh – 7F4000h				
		4KB	7F3FFFFh – 7F3000h				
		4KB	7F2FFFFh – 7F2000h				
		4KB	7F1FFFFh – 7F1000h				
		4KB	7F0FFFFh – 7F0000h				
64KB (Sector 126)	32KB	4KB	7EFFFFFFh – 7EF000h				
		4KB	7EEFFFFh – 7EE000h				
		4KB	7EDFFFFh – 7ED000h				
		4KB	7ECFFFFh – 7EC000h				
		4KB	7DBFFFFh – 7EB000h				
		4KB	7EAFFFFh – 7EA000h				
		4KB	7E9FFFFh – 7E9000h				
		4KB	7EBFFFFh – 7EB000h				
		4KB	7E7FFFFh – 7E7000h				
		4KB	7E6FFFFh – 7E6000h				
		4KB	7E5FFFFh – 7E5000h				
		4KB	7E4FFFFh – 7E4000h				
		4KB	7E3FFFFh – 7E3000h				
		4KB	7E2FFFFh – 7E2000h				
		4KB	7E1FFFFh – 7E1000h				
		4KB	7E0FFFFh – 7E0000h				
64KB (Sector 0)	32KB	4KB	00FFFFFFh – 00F000h				
		4KB	00EFFFFh – 00E000h				
		4KB	00DFFFFh – 00D000h				
		4KB	00CFFFFh – 00C000h				
		4KB	00BFFFFh – 00B000h				
		4KB	00AFFFFh – 00A000h				
		4KB	009FFFFh – 009000h				
		4KB	008FFFFh – 008000h				
		4KB	007FFFFh – 007000h				
		4KB	006FFFFh – 006000h				
		4KB	005FFFFh – 005000h				
		4KB	004FFFFh – 004000h				
		4KB	003FFFFh – 003000h				
		4KB	002FFFFh – 002000h				
		4KB	001FFFFh – 001000h				
		4KB	000FFFFh – 000000h				

CE WS12

Unprotect Sector

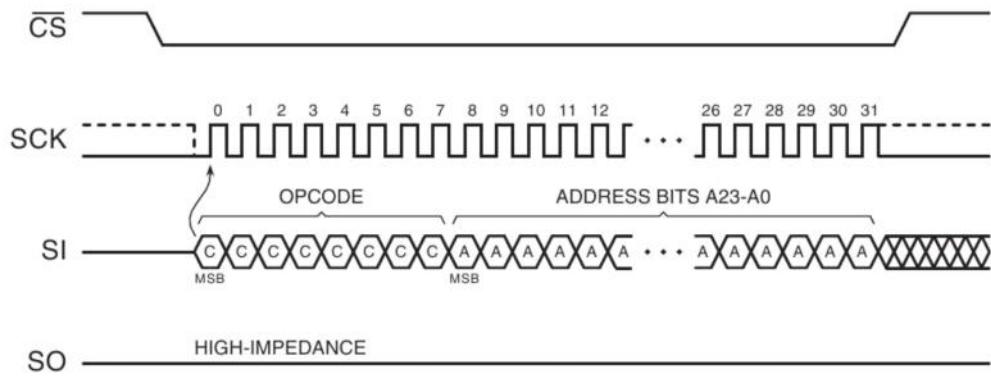
- › Vor Ausführung: „Write Enable“ senden
- › Opcode 0x39



31

Block Erase

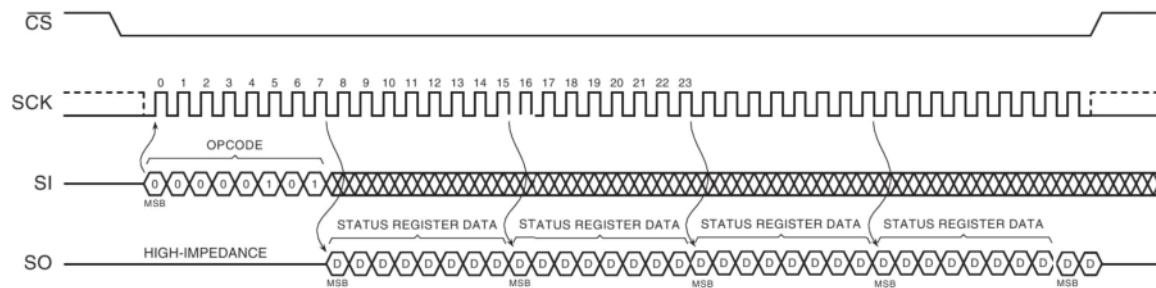
- ▶ Vor Ausführung: „Write Enable“ senden
- ▶ Opcode: 0x20 (4-KBytes), 0x52 (32-kBytes), 0xD8 (64-kBytes)



CE WS12

Auf „Fertig“ warten

- ▶ Kontinuierlich Statusregister lesen.
- ▶ Opcode: 0x05.
 - ▶ Muss nur einmal gesendet werden.
 - ▶ Danach kann der Inhalt des Statusregisters solange ausgelesen werden, bis RDY/BSY == 0.



Flash-Speicher

Programmieren

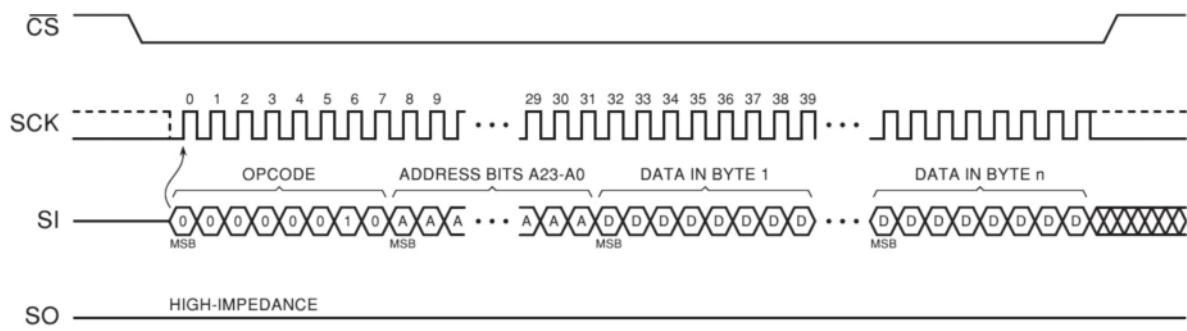
- ▶ Programmieren von einzelnen Bytes möglich.
- ▶ Pro Programmiervorgang können aber nur Bytes innerhalb eines 256 Bytes Block programmiert werden.
- ▶ Bytes müssen zuvor gelöscht werden:
 - ▶ Gelöschte Bytes haben den Wert 0xFF.
- ▶ Ablauf:
 1. **Unprotect Sectors:**
 - Write Enable
 - Unprotect
 2. **Program Sectors:**
 - Write Enable
 - **Program Sectors:**
Notwendiger Zeitablauf wird automatisch vom Chip generiert.
 3. **Warten:**
 - Read Status Register und prüfe RDY/BSY

34

Page Program (02h Command)	Page Address Range
256 Bytes	7FFFFFh - 7FFF00h
256 Bytes	7FFEFFFh - 7FFE00h
256 Bytes	7FFDFFn - 7FFD00h
256 Bytes	7FFCFFFh - 7FFC00h
256 Bytes	7FFBFFn - 7FFB00h
256 Bytes	7FFAFFn - 7FFA00h
256 Bytes	7FF9FFFh - 7FF900h
256 Bytes	7FF8FFFh - 7FF800h
256 Bytes	7FF7FFFh - 7FF700h
256 Bytes	7FF6FFFh - 7FF600h
256 Bytes	7FF5FFFh - 7FF500h
256 Bytes	7FF4FFFh - 7FF400h
256 Bytes	7FF3FFFh - 7FF300h
256 Bytes	7FF2FFFh - 7FF200h
256 Bytes	7FF1FFFh - 7FF100h
256 Bytes	7FF0FFFh - 7FF000h
256 Bytes	7FEFFFh - 7FEF00h
256 Bytes	7FEFFFh - 7FE000h
256 Bytes	7FECFFFh - 7FEC00h
256 Bytes	7FEBFFFh - 7EBE00h
256 Bytes	7FAFFFh - 7FAE00h
256 Bytes	7FE9FFFh - 7FE900h
256 Bytes	7FE8FFFh - 7FE800h
:	
256 Bytes	0017FFFh - 001700h
256 Bytes	0016FFFh - 001600h
256 Bytes	0015FFFh - 001500h
256 Bytes	0014FFFh - 001400h
256 Bytes	0013FFFh - 001300h
256 Bytes	0012FFFh - 001200h
256 Bytes	0011FFFh - 001100h
256 Bytes	0010FFFh - 001000h
256 Bytes	000FFFh - 000F00h
256 Bytes	000EFFh - 000E00h
256 Bytes	000DFFn - 000D00h
256 Bytes	000CFFn - 000C00h
256 Bytes	000BFFn - 000B00h
256 Bytes	000AFFn - 000A00h
256 Bytes	0009FFFh - 000900h
256 Bytes	0008FFFh - 000800h
256 Bytes	0007FFFh - 000700h
256 Bytes	0006FFFh - 000600h
256 Bytes	0005FFFh - 000500h
256 Bytes	0004FFFh - 000400h
256 Bytes	0003FFFh - 000300h
256 Bytes	0002FFFh - 000200h
256 Bytes	0001FFFh - 000100h
256 Bytes	0000FFFh - 000000h

Programmierkommando

- ▶ Bestandteile des Kommandos:
 - ▶ **Opcode 0x02**
 - ▶ **3-Bytes Adresse**
 - ▶ **N Daten**
- ▶ Falls Datenmenge 256-Bytes Blockgrenze überschreitet:
 - ▶ „Wrap-around“ auf Anfang des gleichen Blocks!
- ▶ Programmievorgang beginnt automatisch nach Freigabe der CS-Leitung



Computer Engineering

WS 2012

Sytemeinstellungem

HTM – SHF - SWR

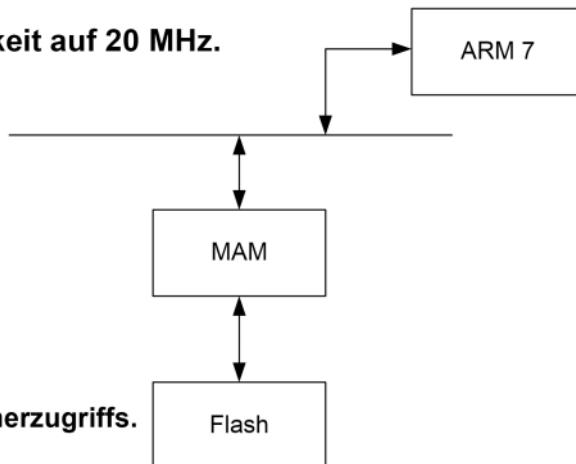
Übersicht



- ▶ Memory Accelerator Module
- ▶ Taktzeugung
- ▶ Stromverbrauch, Batteriebetrieb, Schlafzustände
- ▶ Reset
- ▶ Watchdog

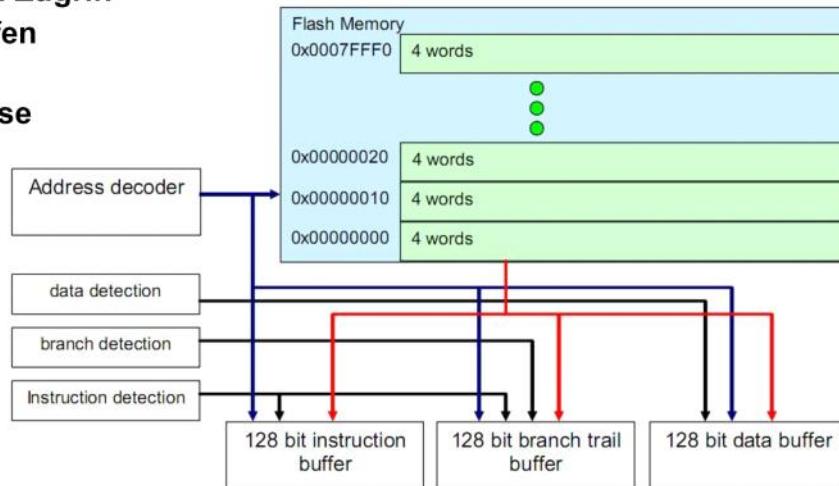
Memory Accelerator Module (MAM)

- ▶ Maschinencode ist im Flash gespeichert.
- ▶ Flash ist Flaschenhals bezüglich Programmausführungsgeschwindigkeit.
 - ▶ Zugriffszeit ca. 50 ns:
Begrenzung der Geschwindigkeit auf 20 MHz.
- ▶ Gängige Auswege:
 - ▶ Programm ins RAM kopieren.
 - ▶ Cache verwenden.
- ▶ LPC2000 verwendet MAM:
 - ▶ Kompromiss zwischen
 - Komplexität eines Caches und
 - Einfachheit des direkten Speicherzugriffs.



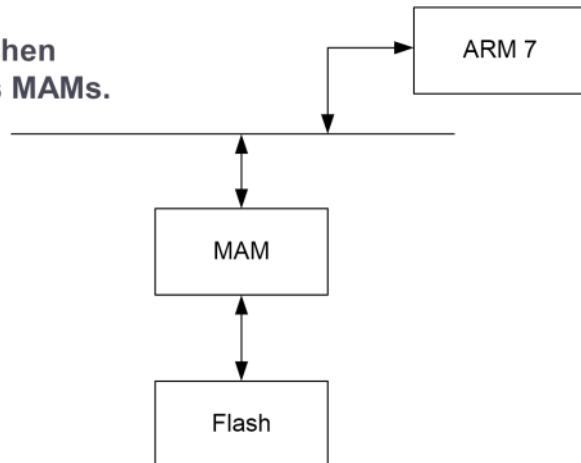
Memory Accelerator Module (MAM)

- ▶ Flash ist in 128-Bit Breite organisiert.
- ▶ Mit jedem Zugriff können 4 Instruktionen gelesen werden.
- ▶ Verbessert:
 - ▶ **Sequentiellen Zugriff**
 - ▶ **Kurze Schleifen**
 - ▶ **Sprünge zur selben Adresse**
- ▶ **Wichtig:**
Timing der Flashzugriffe muss beachtet werden.



Memory Accelerator Module (MAM)

- ▶ MAM befindet sich zwischen Flash und CPU
- ▶ MAM ist transparent für den Benutzer.
- ▶ Konfiguration über zwei Register:
 - ▶ **Timing Register** und
 - ▶ **Control Register**.
- ▶ Zusätzliche Register ermöglichen Statistiken zur Effektivität des MAMs.



Memory Accelerator Module (MAM)

- ▶ Drei Betriebsarten:
 - ▶ **Mode 0: Off**
 - Jeder Programmspeicherzugriff wirkt direkt auf das Flash
 - ▶ **Mode 1: Partially enabled**
 - Sequentielle Programmspeicherzugriffe werden vom Zwischenspeicher genommen.
 - Nichtsequentielle Zugriffe wirken direkt auf das Flash
 - ▶ **Mode 2: Fully enabled**
 - Alle Programmspeicherzugriffe werden nach Möglichkeit vom Zwischenspeicher genommen.

MAM Control Register (MAMCR - address 0xE01F C000) bit description

Bit	Symbol	Value	Description	Reset value
1:0	MAM_mode_control	00	These bits determine the operating mode of the MAM.	0
			MAM functions disabled	
			MAM functions partially enabled	
			MAM functions fully enabled	
		11	Reserved. Not to be used in the application.	

Memory Accelerator Module (MAM)

- ▶ Timing-Einstellung

Suggestions for MAM timing selection

system clock	Number of MAM fetch cycles in MAMTIM
< 20 MHz	1 CCLK
20 MHz to 40 MHz	2 CCLK
40 MHz to 60 MHz	3 CCLK
> 60 MHz	4 CCLK

MAM Timing register (MAMTIM - address 0xE01F C004) bit description

Bit	Symbol	Value	Description
7	MAM_fetch_cycle_timing	000	These bits set the duration of MAM fetch operations. 0 - Reserved
		001	1 - MAM fetch cycles are 1 processor clock (CCLK) in duration
		010	2 - MAM fetch cycles are 2 CCLKs in duration
		011	3 - MAM fetch cycles are 3 CCLKs in duration
		100	4 - MAM fetch cycles are 4 CCLKs in duration
		101	5 - MAM fetch cycles are 5 CCLKs in duration
		110	6 - MAM fetch cycles are 6 CCLKs in duration
		111	7 - MAM fetch cycles are 7 CCLKs in duration

MAM: Benchmarks

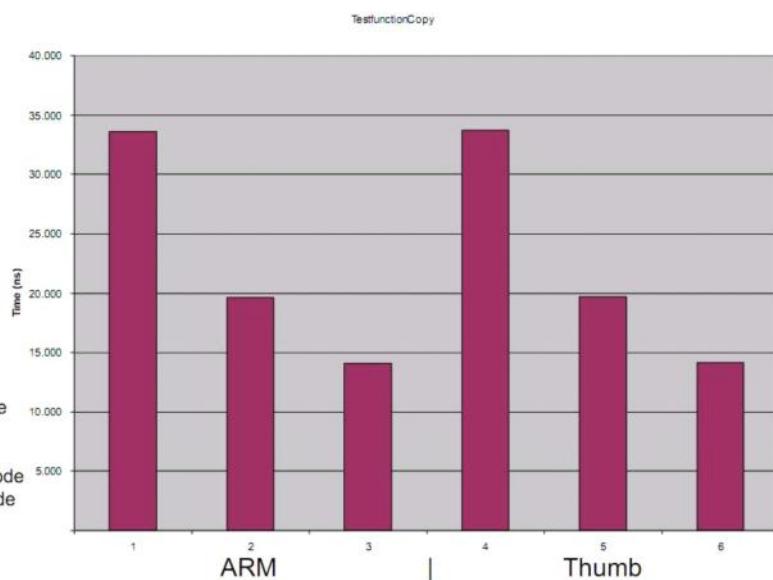
hitex ■
DEVELOPMENT TOOLS

TestFunctionCopy

- ◆ Copy Data (256 Words)
- ◆ RAM to RAM

Test Configurations

1. MAM Disabled, ARM Code
2. MAM Partly Enabled, ARM Code
3. MAM Fully Enabled, ARM Code
4. MAM Disabled, Thumb Code
5. MAM Partly Enabled, Thumb Code
6. MAM Fully Enabled, Thumb Code



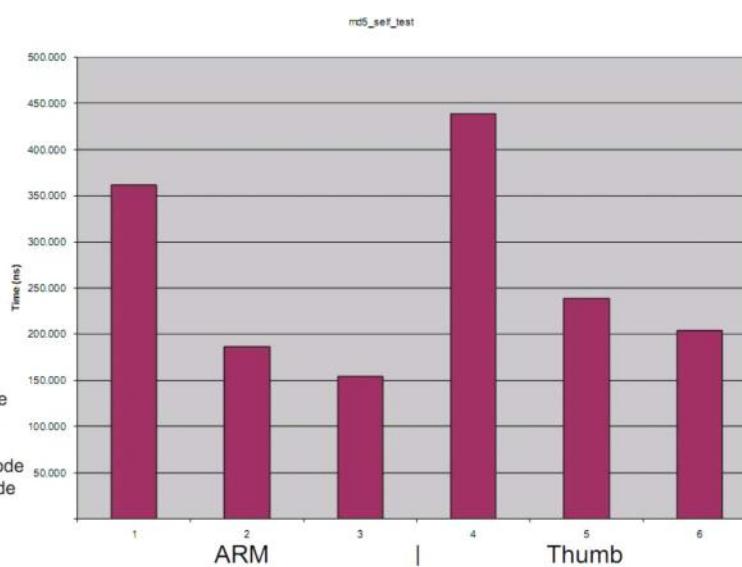
MAM: Benchmarks

md5_self_test

- ◆ Use the self test function which is included in the md5 package

Test Configurations

1. MAM Disabled, ARM Code
2. MAM Partly Enabled, ARM Code
3. MAM Fully Enabled, ARM Code
4. MAM Disabled, Thumb Code
5. MAM Partly Enabled, Thumb Code
6. MAM Fully Enabled, Thumb Code



Übersicht



- ▶ Memory Accelerator Module
- ▶ Taktzeugung
- ▶ Stromverbrauch, Batteriebetrieb, Schlafzustände
- ▶ Reset
- ▶ Watchdog

CE WS12

Technologien Taktquellen

- ▶ Externer Oszillator
- ▶ Externer Quarz
- ▶ Externer Keramik-Resonator
- ▶ Interner RC-Oszillatot

Auswahlkriterien

- ▶ Preis
- ▶ Genauigkeit
- ▶ Platzbedarf
- ▶ Strombedarf



Oszillatoren

12

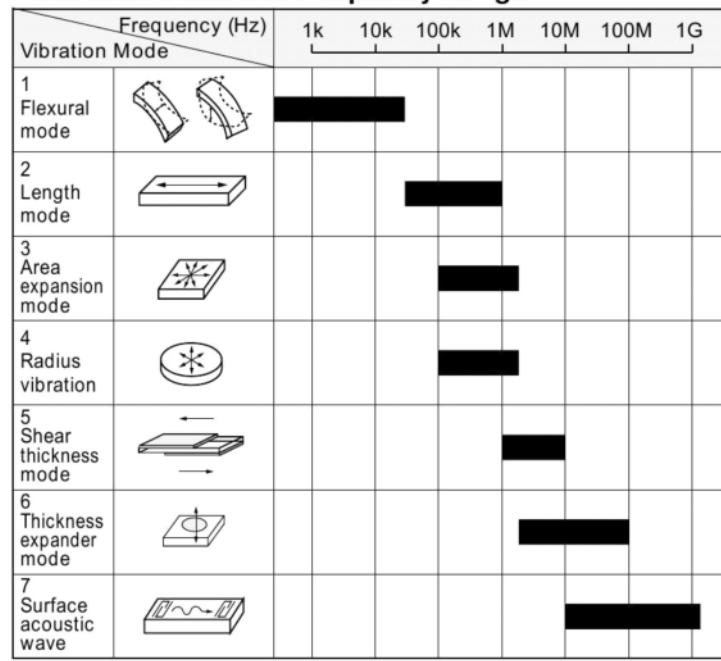
■ Characteristics of Various Oscillator Elements

Name	Symbol	Price	Size	Adjust- ment	Oscillation Frequency Initial Tolerance	Long-term Stability
LC		Inexpen- sive	Big	Required	±2.0%	Fair
CR		Inexpen- sive	Small	Required	±2.0%	Fair
Quartz Crystal		Expen- sive	Big	Not required	±0.001%	Excellent
Ceramic Resonator		Inexpen- sive	Small	Not required	±0.5%	Excellent

Schwingquarz

Systemeinstellungen

■ Vibration Mode and Frequency Range



[Note] :   show the direction of vibration

CE WS12

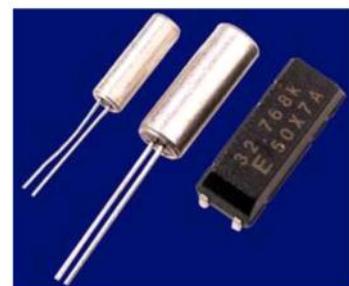
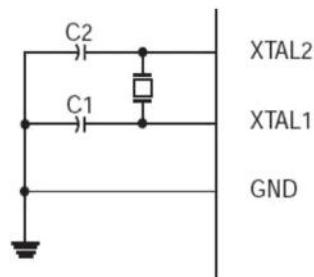
Externer Taktoszillator

	CXO-11	CXO-21
Package	14 Pin DIP	8 Pin DIP
Frequency Range	1.000 ~ 100.000 MHz	
Frequency Stability	A=±25ppm, B=± 50ppm, C=± 100ppm	
Operating Temperature Range	0°C - 70°C (-40°C - +85°C -> Option 'S')	
Storage Temperature Range	-55°C - 125°C	
Supply Voltage	5.0 VDC ±10%	
Aging (at 25°C)	±5ppm / year max.	
Supply Current	1.000MHz to 23.999MHz 24.000MHz to 70.000MHz 70.000MHz to 100.000MHz	20mA max. 30mA max. 40mA max.
Waveform Symmetry	40/60 % Normal, 45/ 55% Tight	
Rise/Fall Time	< 9 MHz < 32 MHz > 32 MHz	10ns max. 5ns max. 4ns max.
Output Voltage	Logic Low Logic High	0.4 V max. 2.4 V min.
Output Load	1 to 10 TTL Load	
Start-up Time	10 ms	



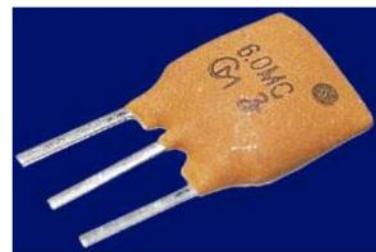
Externer Quarz

Frequenzbereich:	4 ... 40 MHz
Frequenztoleranz (25°C):	± 50 ppm
Shunt-Kapazität (max.):	5 pF
Betriebstemperaturbereich:	- 20 ... + 70 °C
Temperaturstabilität < 5,5 MHz:	± 50 ppm
Temperaturstabilität > 5,5 MHz:	± 30 ppm
Isolationswiderstand (min.):	500 MΩ
Belastung (empfohlen):	10 ... 100 µW
Alterung:	± 5 ppm / Jahr
Lötwärmebeständigkeit bis 3 Min.:	< 230 °C
bis 10 s:	> 260 °C
Schockbeständigkeit (max.):	± 10 ppm



Externer Keramik-Resonator

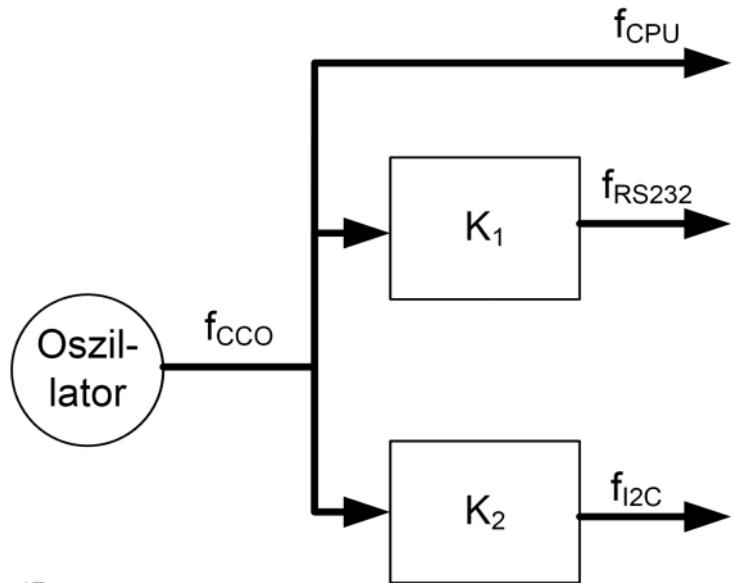
- ▶ 8,00 MHz Keramik-Resonator, 3-pin, für integrierte Ladekapazität.
- ▶ Frequenztoleranz: +/- 0,5%
- ▶ Frequenzstabilität bis 6 MHz: +/- 0,3%
- ▶ Frequenzstabilität ab 8 MHz: +/- 0,4%



Interner RC-Oszillator

- ▶ Feste Taktfrequenz: 8 MHz
- ▶ Genauigkeit:
- ▶ Standard: 10 % (bei 5 V und 25 °C)
- ▶ Kann kalibriert werden: 2%

Takterzeugung



17

CE WS12

Takterzeugung, Beispiel Atmel AVR

- ▶ Im Mikrocontroller verwendete Takte werden meist von einer gemeinsamen Quelle abgeleitet.
- ▶ Kann bei niedrigen Taktraten problematisch werden, z.B. bei gleichzeitiger Nutzung von I2C und RS232 (Beispiel AVR):
 - ▶ I2C mit Bitrate 400kHz

$$Teiler = \frac{f_{CLK}}{2 \cdot Bitrate} - 8 \quad Teiler \geq 10$$

- Mögliche Teiler sind 10, 11 und 12 bei 14.4 MHz, 15.2 MHz und 16 MHz

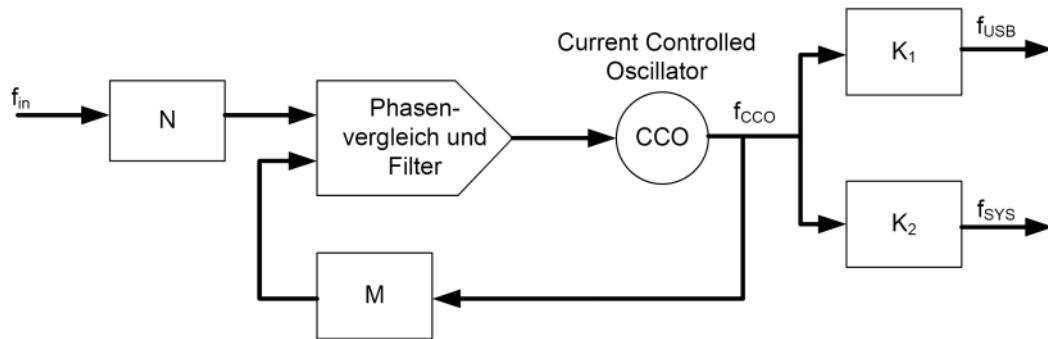
- ▶ RS232 mit Baudrate 115200

$$Teiler = \frac{f_{CLK}}{8 \cdot Baudrate} - 1$$

- Fehler bei bester Einstellung:
 - 2.6% bei 14.4 MHz, 3.2% bei 15.2 MHz, -2.2% bei 16 MHz,

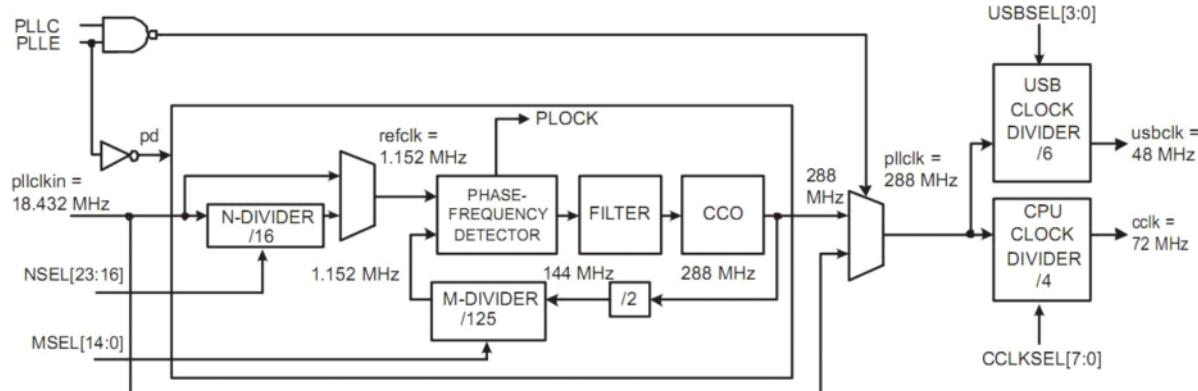
Takterzeugung

- ▶ Frequenzvervielfachung:
 - ▶ höhere Oszillatorfrequenz ermöglicht flexible Takteinstellung
 - ▶ Verwendung von Phase-locked loops (PLL)



Takterzeugung

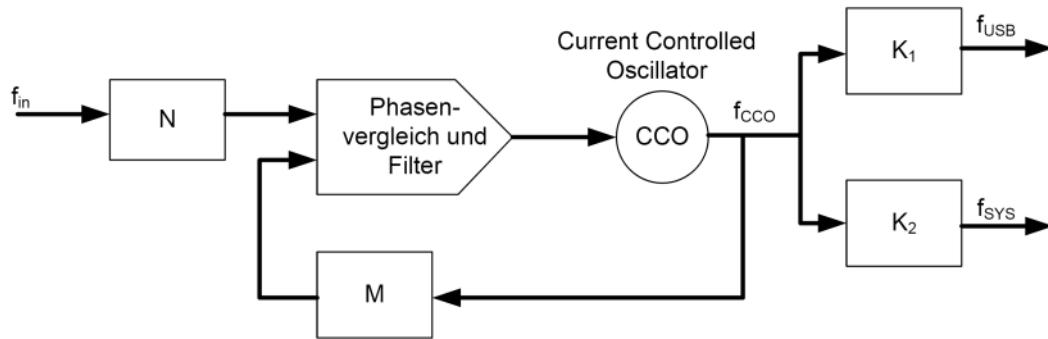
- ▶ Getrennter Takt für
 - ▶ System (cclk) und
 - ▶ Usb (usbclk)
- ▶ Takte werden mittels Frequenzvervielfacher von langsamer externer Taktquelle abgeleitet

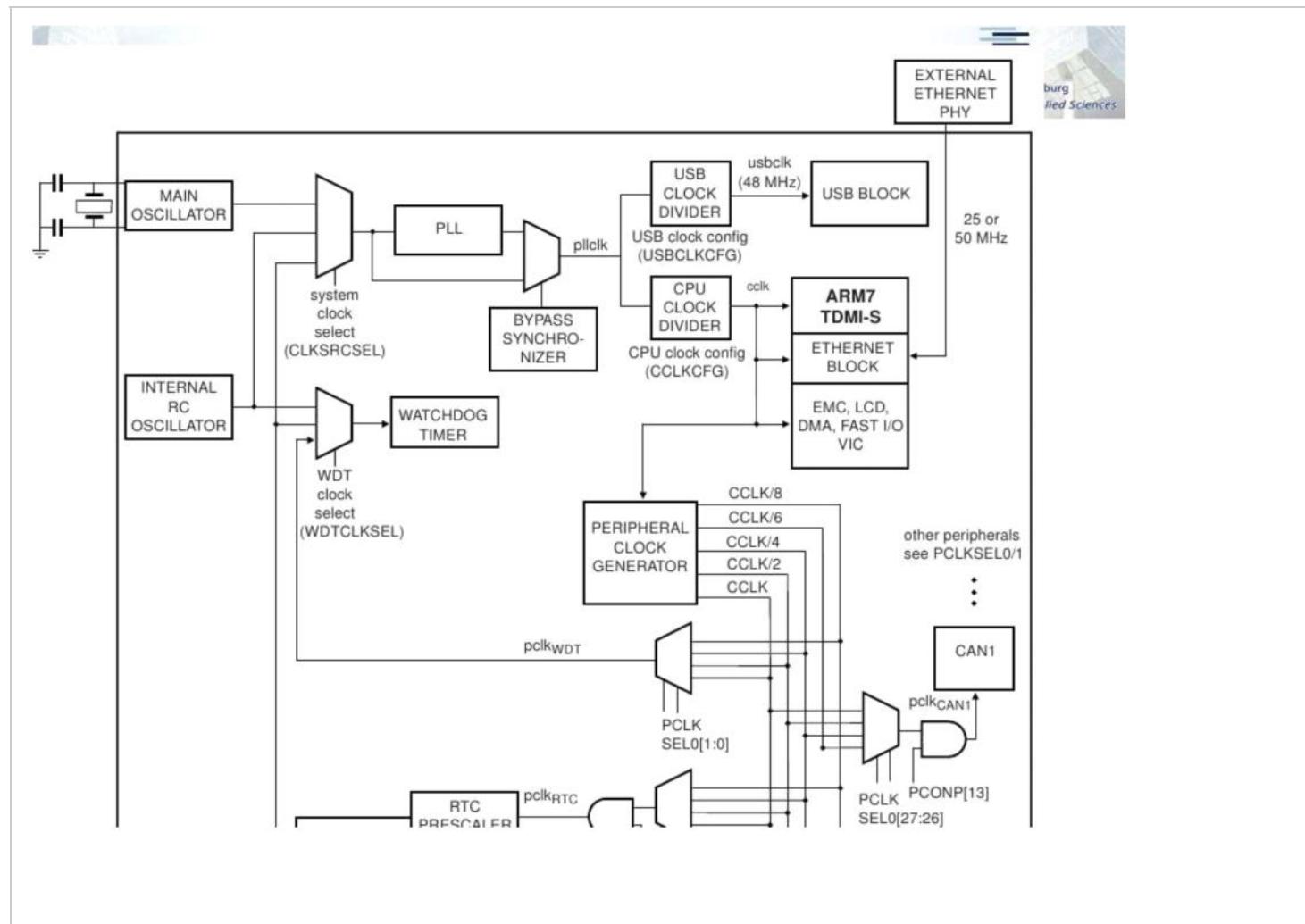


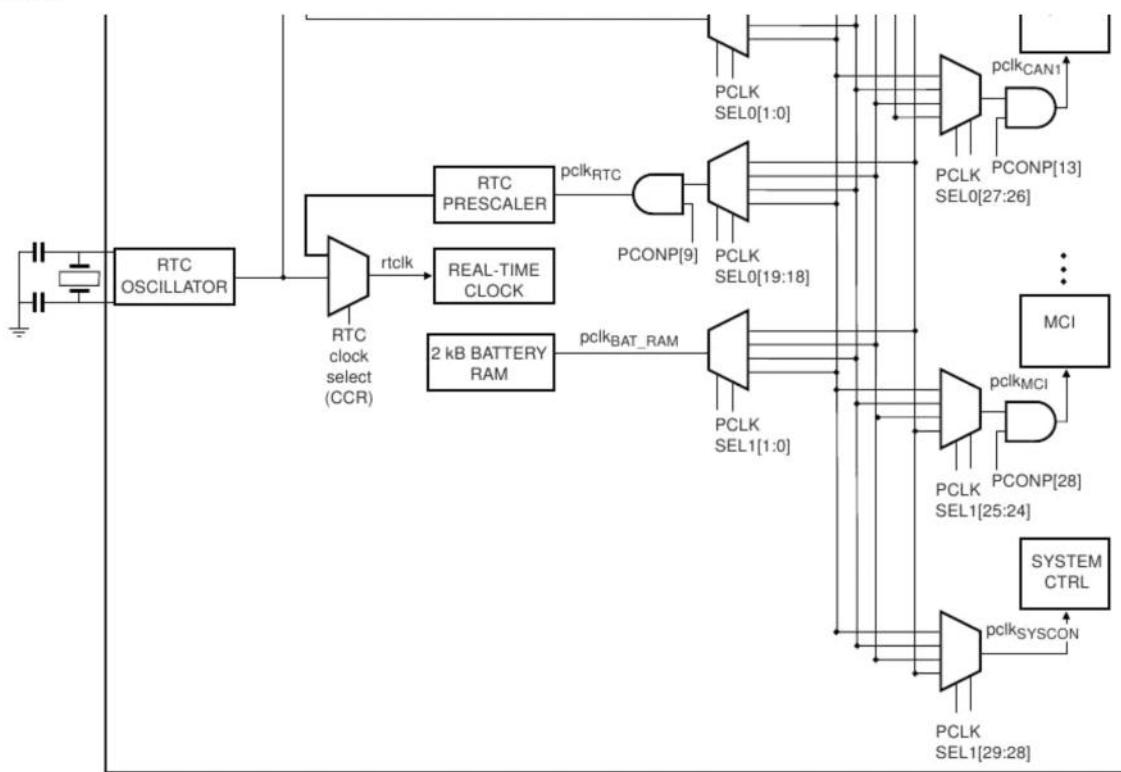
CE WS12

Takterzeugung LPC2468

- ▶ f_{in} : 32kHz bis 24 MHz
- ▶ f_{CCO} : 275 MHz bis 550 MHz
- ▶ f_{USB} : 48 MHz (4*12 MHz)
- ▶ f_{SYS} : maximal 72 MHz
- ▶ K_1, K_2 : gerade
1 bis 32
- ▶ N: 1 bis 32
- ▶ M: 6 bis 32768







Übersicht



- ▶ Memory Accelerator Module
- ▶ Taktzeugung
- ▶ Stromverbrauch, Batteriebetrieb, Schlafzustände
- ▶ Reset
- ▶ Watchdog

Stromverbrauch LPC2468

$T_{amb} = -40^{\circ}\text{C}$ to $+85^{\circ}\text{C}$ for commercial applications, unless otherwise specified.

Symbol	Parameter	Conditions	Min	Typ ^[1]	Max	Unit
$I_{DD(DCDC)act(3V3)}$	active mode DC-to-DC converter supply current (3.3 V)	$V_{DD(DCDC)(3V3)} = 3.3 \text{ V}$; $T_{amb} = 25^{\circ}\text{C}$; code while(1){} executed from flash; no peripherals enabled; PCLK = CCLK				
		CCLK = 10 MHz	-	15	-	mA
		CCLK = 72 MHz	-	63	-	mA
		all peripherals enabled; PCLK = CCLK / 8				
		CCLK = 10 MHz	-	21	-	mA
		CCLK = 72 MHz	-	92	-	mA
		all peripherals enabled; PCLK = CCLK				
		CCLK = 10 MHz	-	27	-	mA
		CCLK = 72 MHz	-	125	-	mA
$I_{DD(DCDC)pd(3V3)}$	power-down mode DC-to-DC converter supply current (3.3 V)	$V_{DD(DCDC)(3V3)} = 3.3 \text{ V}$; $T_{amb} = 25^{\circ}\text{C}$		150	-	μA
I_{BATact}	active mode battery supply current	DC-to-DC converter on	[10]	-	20	μA
		DC-to-DC converter off	[10]	-	28	μA

Betriebsarten der CPU, Schlafzustände

- › Zum Stromsparen hat die CPU 3 verschiedene Schlafzustände:
 - › **Idle**
 - › **Sleep**
 - › **Power-down**
- › Unterscheiden sich
 - › in den möglichen **Quellen zum Aufwecken**
 - › benötigte **Zeit zum Aufwecken**
 - › Höhe der **Stromersparnis**

Betriebsarten der CPU, Schlafzustände

› Idle Mode

- › Takt der CPU wird abgeschaltet.
- › Peripherie funktioniert weiter und kann Interrupts auslösen.
- › Stromersparnis dadurch,
dass CPU, Speicher und interne Busse nicht mehr getaktet werden.

Betriebsarten der CPU, Schlafzustände

- ▶ **Sleep Mode**
 - ▶ Ausschalten des Hauptoszillators.
 - ▶ Interner RC-Oszillator läuft weiter, ist aber nicht mit System verbunden
 - ermöglicht schnelles Aufwachen
 - ▶ Real-time Clock funktioniert weiter
 - Echtzeituhr kann CPU wieder aufwecken.
 - ▶ Aufwecken kann durch Echtzeituhr oder durch andere Interrupts, die keinen Takt benötigen, erfolgen.
 - ▶ Nach Aufwecken muss Hauptoszillator gestartet werden:
 - 4096 Takte Wartezeit
 - ▶ Zusätzlicher Zeitbedarf für das Neukonfigurieren der PLL:
 - 500µsec

Betriebsarten der CPU, Schlafzustände

- ▶ **Power-down Mode**
 - ▶ **Wie Sleep Mode, zusätzlich:**
 - Power-down des Flash
 - Ausschalten des RC-Oszillators.
 - ▶ **Real-time Clock funktioniert weiter**
 - Echtzeituhr kann CPU wieder aufwecken.
 - ▶ **Aufwecken kann durch Echtzeituhr oder durch andere Interrupts, die keinen Takt benötigen, erfolgen.**
 - ▶ **Nach Aufwecken:**

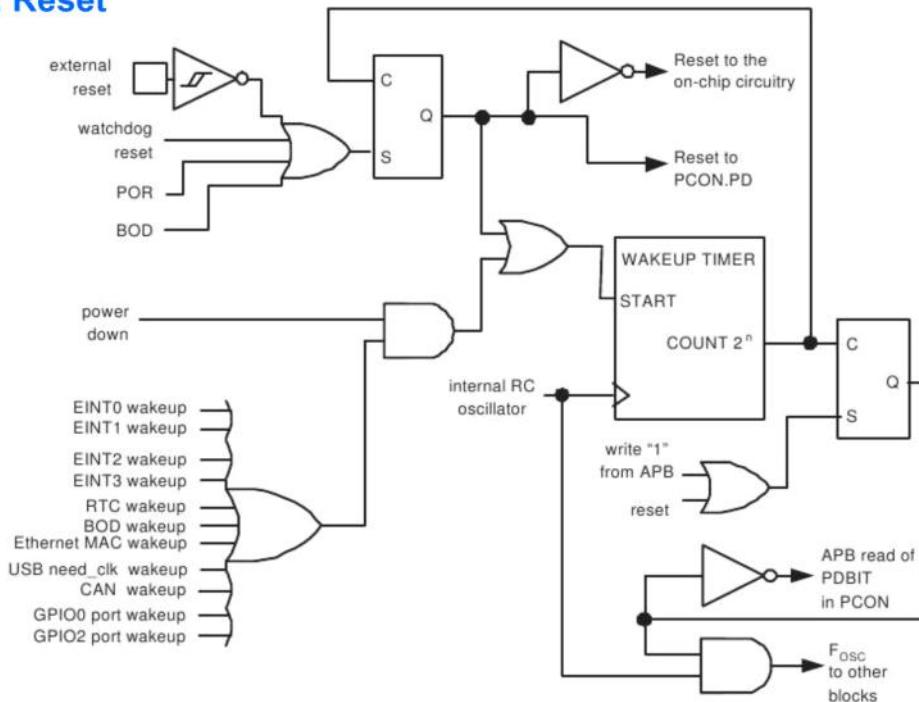
• Start des internen RC-Oszillators:	60 µsec
• Start des Flash:	100 µsec
• Start des Hauptoszillators:	4096 Takte
• Neukonfiguration der PLL	500 µsec

Übersicht

- ▶ Memory Accelerator Module
- ▶ Taktzeugung
- ▶ Stromverbrauch, Batteriebetrieb, Schlafzustände
- ▶ Reset
- ▶ Watchdog

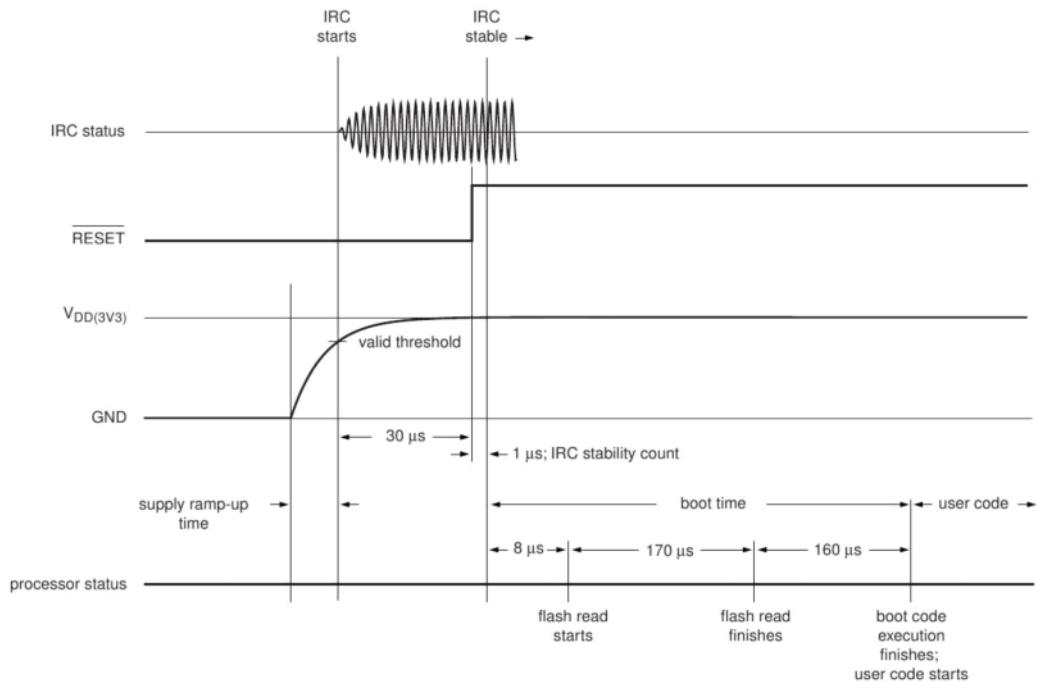


LPC2468: Reset



CE WS12

LPC2468: Reset

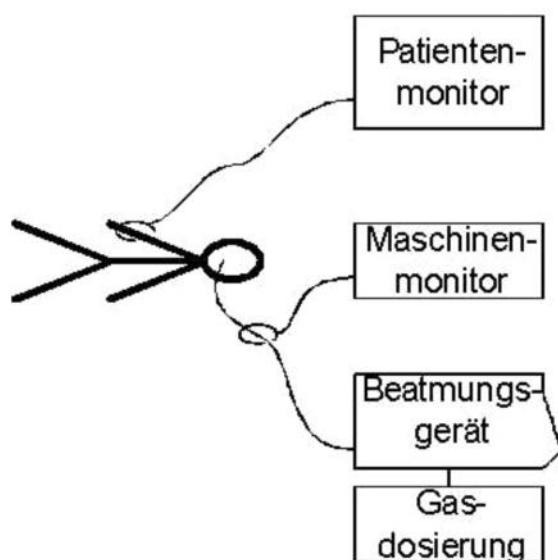


Übersicht

- ▶ Memory Accelerator Module
- ▶ Taktzeugung
- ▶ Stromverbrauch, Batteriebetrieb, Schlafzustände
- ▶ Reset
- ▶ Watchdog



Sicherheitsaspekte: Beispiel Medizintechnik

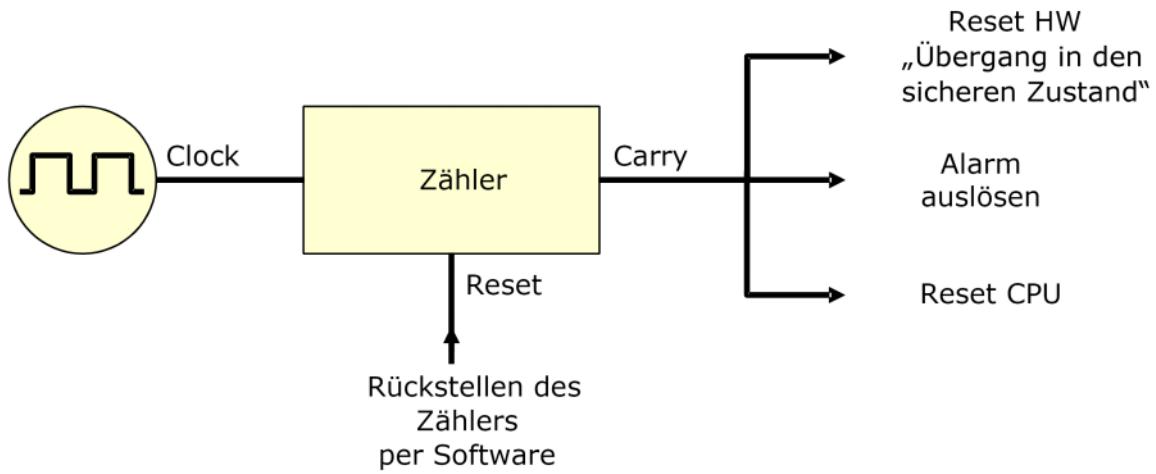


Überwachung:
Kreislauf (EKG, Herzfrequenz)
Sauerstoffsättigung des Blutes
Blutdruck

Überwachung:
Gaszusammensetzung
(O₂, CO₂, Narkosemittel)
Beatmungsdruck

Einstellung:
Beatmungsfrequenz, -volumen
Sauerstoffgehalt, Narkosemittel

Prinzipieller Aufbau



Erkennbare Fehler

- Programmpfad, welcher Zurücksetzen des Watchdogs beinhaltet, wird nicht mehr durchlaufen, z. B. wg.
 - CPU hängt in Endlosschleife
 - Interrupt-Hardware umprogrammiert
 - CPU ist im Sleep-Zustand
 - CPU ist aus dem Programm gesprungen

Nicht erkennbare Fehler

- Falsche Daten
- Zu häufiges Durchlaufen der Schleife
- versehentliches Triggern des Zählers
- Falscher Takt (wenn keine unabhängigen Taktquellen)

Watchdog

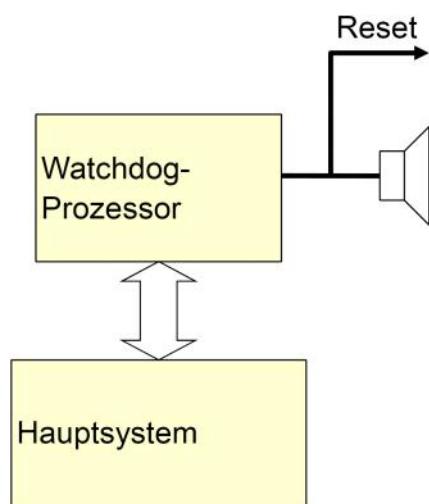
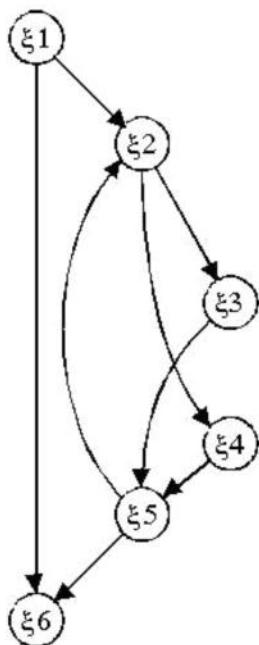
Design-Hinweise

- Zeitbasis muss unabhängig von CPU-Takt sein.
- Vorsicht bei programmierbarer Periode.
- Schutz gegen versehentliches Triggern,
z. B. mit spezieller Sequenz: (0x55, 0xAA)
- Besser: Verwendung komplexer Sequenzen, die z. B. den Programmablauf beschreiben.
- Unabhängige Überwachung von
 - Hauptschleife und
 - Interruptserviceroutinen.
- Watchdog-Funktion muss beim Einschalten des Systems überprüft werden.

Watchdog

Watchdog Prozessor

```
xi1
while( x > 0 )
{
  xi2
  x = x - 1;
  if( y < x )
  {
    xi3
  }
  else
  {
    xi4
  }
  xi5
  z = y + x;
}
xi6
```

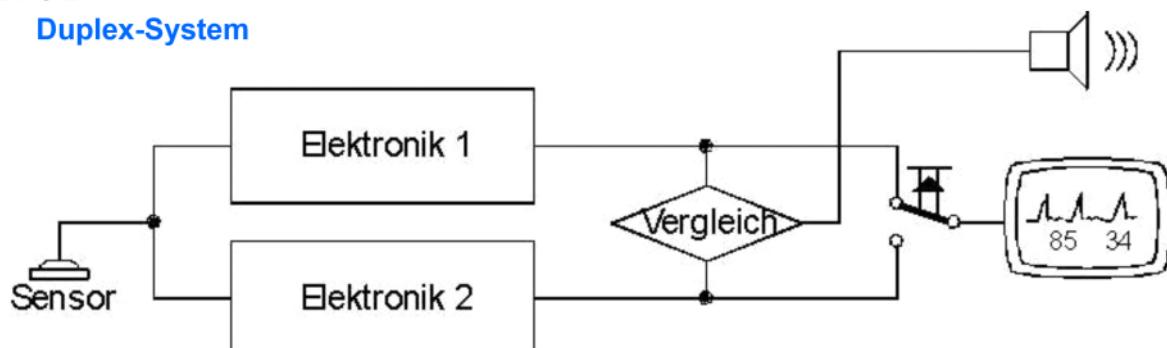


Watchdog Prozessor

- ▶ Überwachung des Programmflusses
- ▶ Darstellung des Programmablaufs als Graph
- ▶ Jedem Knoten wird eine Signatur zugeordnet:
z.B. mittels automatischem Verfahren
- ▶ Signaturen, erlaubte Pfade und Verweildauer sind im Watchdog-Prozessor gespeichert
- ▶ Während der Programmausführung werden Signaturen und Pfade mit den gespeicherten verglichen.
- ▶ Im Fehlerfall: CPU-Reset, Übergang in den "sicheren Zustand" und Alarmierung
- ▶ Zu beachten:
 - ▶ erlaubte, aber falsche Übergänge werden nicht entdeckt
 - ▶ Interruptroutinen können Kontrollfluß jederzeit unterbrechen
 - ▶ Besondere Behandlung von Bibliotheksfunktionen und Betriebssystemaufrufen (z.B. Multitasking)
 - ▶ Erkennung von fehlerhaften Daten nur wenn sie zu unerlaubten Übergängen führen

Watchdog

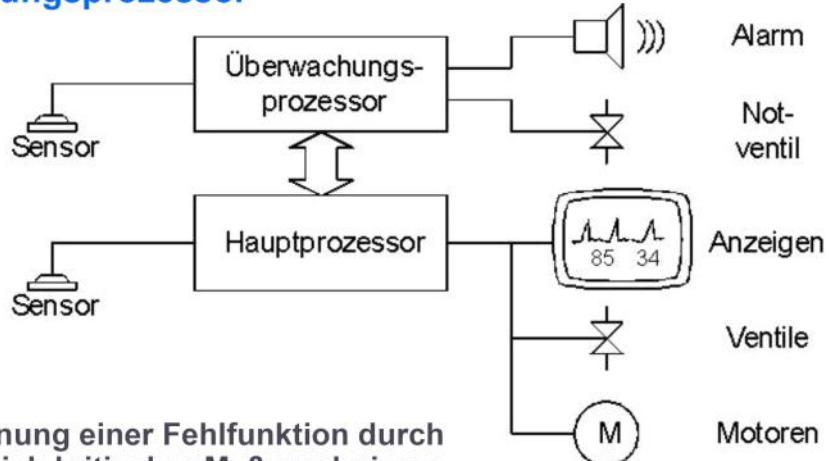
Duplex-System



- Zwei gleichwertige Kanäle zur Signalaufbereitung und –darstellung.
- Einfache Sensorik.
- Bei unterschiedlichen Ergebnissen erfolgt Alarmgebung.
- Bediener muss entscheiden, welche Ergebnisse richtig sind.
- **Sicherstellung der Funktion bei Ausfall einer Elektronik.**
- **Geringer Entwicklungsaufwand.**
- **Keine Sicherheit gegen Ausfall eines Sensors.**
- **Hohe Herstellkosten.**

Watchdog

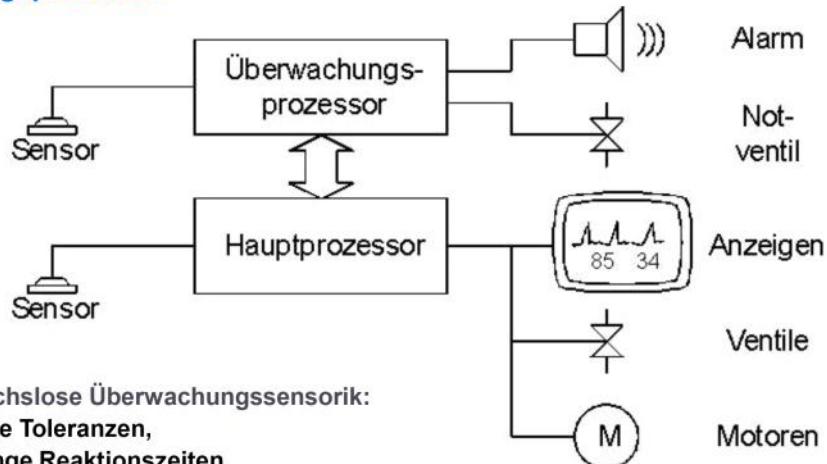
Überwachungsprozessor



- ▶ Erkennung einer Fehlfunktion durch Vergleich kritischer Meßergebnisse:
 - ▶ Übergang in den sicheren Zustand
 - ▶ Alarmierung
 - ▶ Aber keine Sicherstellung der Gerätefunktion

CE WS12

Überwachungsprozessor

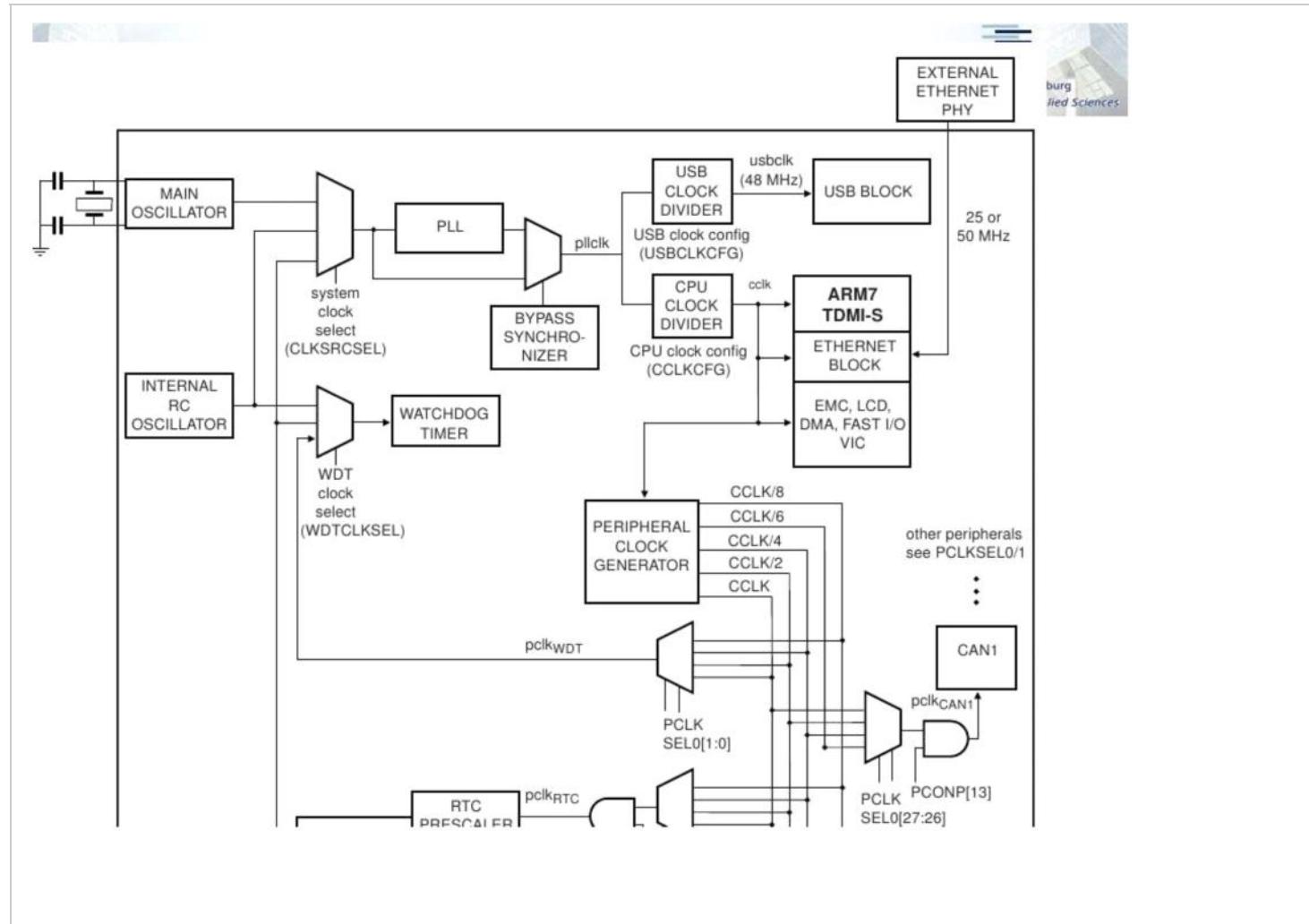


- ▶ Anspruchslose Überwachungssensorik:
 - ▶ große Toleranzen,
 - ▶ geringe Reaktionszeiten,
 - ▶ einfacher zu verifizieren.
- ▶ Geringer Softwareaufwand, einfache Algorithmen im Überwachungsprozessor
- ▶ Anforderungen:
 - ▶ Alle Überwachungseinrichtungen müssen überprüfbar sein (z.B. während des Einschalt-Selbsttest).
 - ▶ Hardwareausfall während des Betriebes kann toleriert werden (führt zum doppelten Fehlerfall).

42

LPC2468: Watchdog

- ▶ Initialisierung:
 - ▶ Auswahl der Taktquelle: Interner RC Oszillator.
 - Wichtig wegen Unabhängigkeit vom Haupttakt.
 - ▶ Programmierung der Betriebsart:
 - Interrupt oder Reset
 - ▶ Festlegung der Timeout-Zeit.
 - 4-facher Vorteiler und 32-Bit Zähler.
 - ▶ Aktivierung des Watchdog durch Schreiben der Sequenz:
 - 0xAA, 0x55 (darf nicht unterbrochen werden!)
- ▶ Danach regelmäßiges Nachtriggern der Watchdog-Timers
 - ▶ Schreiben der Sequenz 0xAA, 0x55.



Computer Engineering WS 2012

Embedded Systems

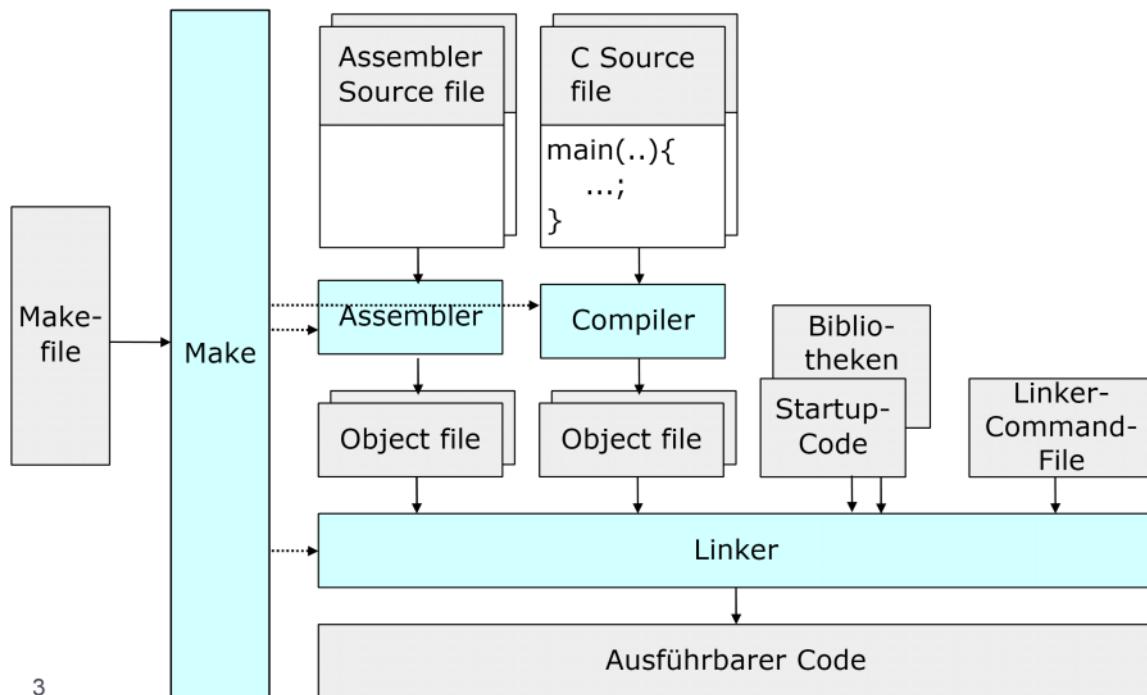
HTM – SHF - SWR

Übersicht



- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
 - ▶ Bootprogramm
 - ▶ Jtag-Interface
- ▶ Fehlersuche
 - ▶ Debugger
 - ▶ Logikanalyser
 - ▶ In-Circuit Emulator

Softwareentwicklung



CE WS12 **Speicherarten**

ROM
(Flash)

- Persistente Speicherung
- Löschbar (blockweise)
- N-mal wiederholbar (typ. 1000)
- Langsam
- Preiswert

- Programmspeicher

RAM

- Flüchtige Speicherung
- Schnell
- Teuer

- Arbeitsspeicher

EEPROM

- Persistente Speicherung
- Byteweise löschbar
- N-mal wiederholbar (typ. 10^6)
- Langsam
- Teuer

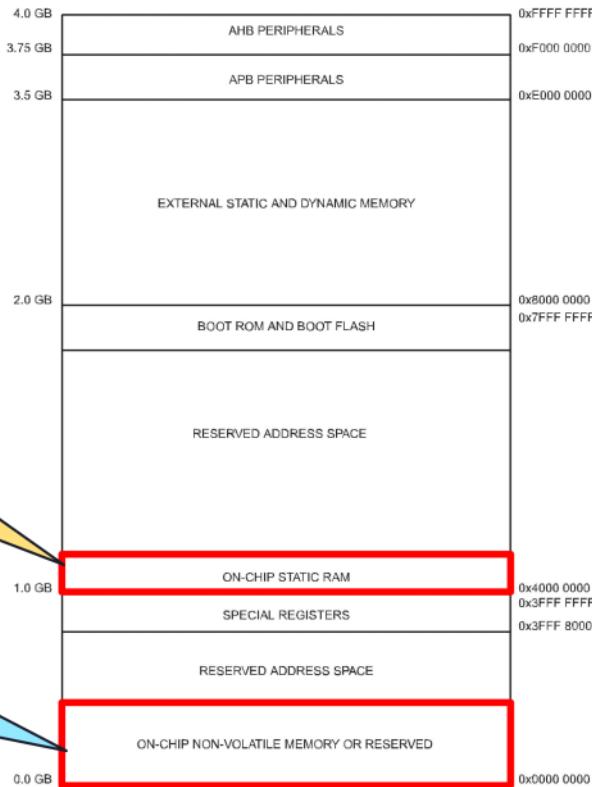
- Speicherung von Konfigurationsdaten

Speicherbelegung LPC2468

RAM (64kByte)
Start: 0x4000 0000
Ende: 0x4000 FFFF

Flash(512kByte)
Start: 0x0000 0000
Ende: 0x0007 FFFF

5



Quelle: LPC24XX User Manual

CE WS12

Sektionen

- ▶ Ausführbarer Code ist in Sektionen aufgeteilt
- ▶ Ermöglicht gezielte Platzierung des erzeugten Codes im Speicher
- ▶ Skript für Linker (Loader) gibt die Adressen der Sektionen vor

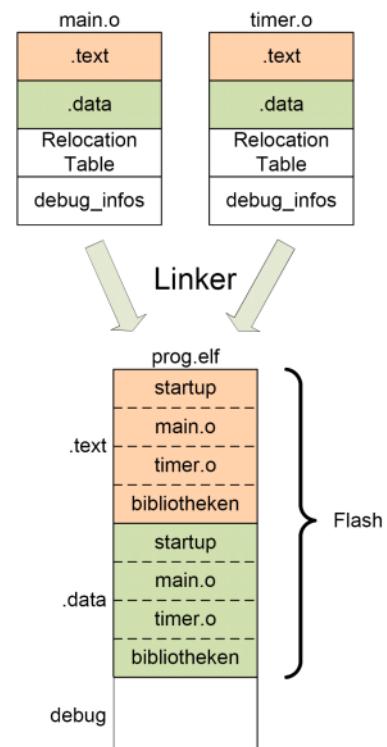
Wichtige Sektionen:

Name	Inhalt	Speicher-ort	Bemerkung
.text	Programmcode	FLASH	
.data	Initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen vom FLASH in das RAM kopiert
.bss	Nicht initialisierte globale Variable	RAM	Bei Programmstart wird Inhalt der Variablen gelöscht

Linker (loader, ld)

- ▶ Zusammenfügen der Sektionen
- ▶ Zuweisung der Adressen
 - ▶ Jede Sektion hat zwei Adressen:
 - Virtual Memory Address (VMA): Adresse der Sektion, wenn das Programm ausgeführt wird.
 - Load Memory Address (LMA): Adresse, unter der die Daten abgelegt werden.
- ▶ Anwendung der Relocation Table
 - ▶ Platzhalter für Adressen werden durch die tatsächlichen Adressen ersetzt.

7

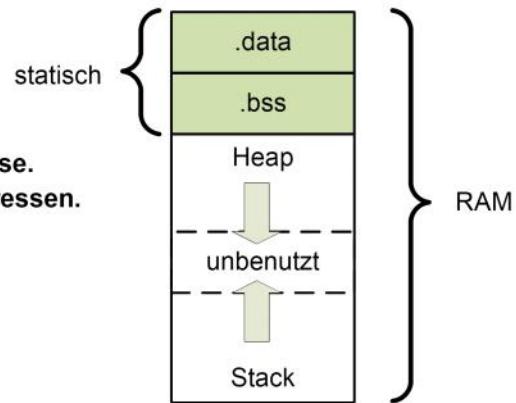


Programmiersprache C: Speicheraufteilung

- ▶ Verwendung des Speichers:
 - ▶ Konstante (Verwendung des Schlüsselworts **const**):
→ Spezielle Sektion **.rodata**.
 - ▶ Globale und statische Variable:
→ **.data** oder **.bss** (Abhängig von Initialisierung)
 - ▶ Lokale Variable:
→ **Stack**
 - ▶ Dynamische Variable:
→ **Heap**
- Dynamische Verwendung
Speicherbedarf schwer vorhersagbar.
Kritisch: Rekursive Funktionsaufrufe.

Programmiersprache C: Speicheraufteilung

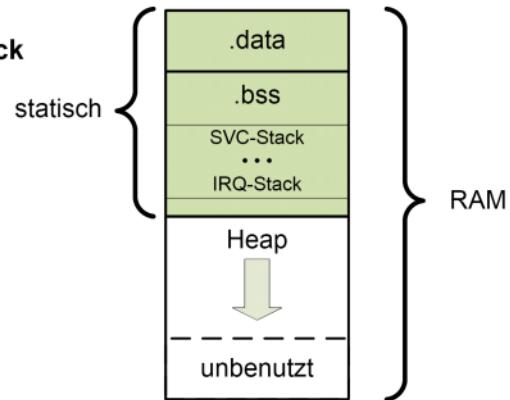
- Häufige Anordnung:
 - **Heap:**
 - Beginnt am Ende der .bss-Sektion.
 - Wächst von kleinen nach großen Adressen.
 - **Stack:**
 - Beginnt an der höchsten RAM-Adresse.
 - Wächst von großen nach kleinen Adressen.
- **Vorteil:** Sehr flexibel.
- **Nachteil:** Kein Schutz vor Stack- oder Heapüberlauf.
- In Embedded Systems muss der Heap- und Stackbedarf vorab abgeschätzt werden.
Heap und Stack sollten statisch eingerichtet werden.



Programmiersprache C: Speicheraufteilung HiTOP

▶ ARM 7 TDMI:

- ▶ **6 Stacks für 7 Betriebsarten**
 - USR/SYS haben gemeinsamen Stack
- ▶ **Größen der Stacks werden statisch in der .bss-Sektion festgelegt**
 - Stacks werden in `start.s` definiert
- ▶ **Heap nutzt den gesamten restlichen Speicher ab Ende der .bss-Sektion**
 - Definiert durch die Laufzeitbibliothek



Programmiersprache C: Laufzeitbibliothek

- ▶ Meist nicht Bestandteil des Compilers.
- ▶ Üblich:
Verbindung der Anwendung mit dem Betriebssystem.
 - ▶ **Betriebssystem wird über Software-Interrupt aufgerufen:**
 - Umschaltung in System-Betriebsmodus.
 - Höhere Privilegien.
 - ▶ **Wesentliche Aufgaben:**
 - Ein- und Ausgabe, Dateien und Geräte.
 - Prozessverwaltung, Starten und Stoppen von Prozessen.
 - Speicherverwaltung.

Programmiersprache C: Laufzeitbibliothek HiTOP

- ▶ Verwendung der **newlib**:
 - ▶ Open-source Projekt
 - ▶ Standard C Bibliothek für Embedded Systems
 - ▶ Wird gepflegt von Red Hat Entwicklern

- ▶ Kann einfach angepasst werden:
 - Unterstützung einer Vielzahl von CPU-Typen
 - Verwendung von insgesamt 17 Stub-Funktionen (System Calls).
 - Optimierung des Speicherbedarfs (printf versus iprintf).
 - Kann reentrant-fähig übersetzt werden.

Übersicht



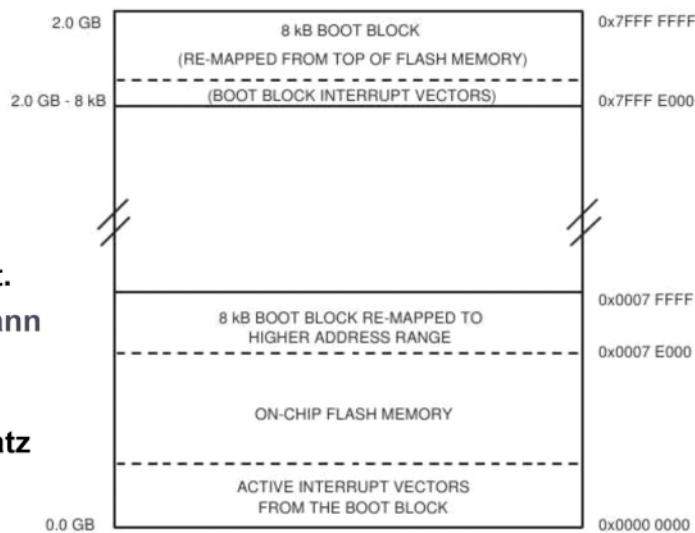
- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
- ▶ Bootprogramm
- ▶ Jtag-Interface
- ▶ Fehlersuche
 - ▶ Debugger
 - ▶ Logikanalyser
 - ▶ In-Circuit Emulator

Chip-Programmierung, Software-Download

- ▶ Programmiergerät
 - ▶ Chip wird in ein externes Programmiergerät gesteckt.
 - ▶ Meist sehr schnell.
 - Gut für Massenproduktion
- ▶ In-System-Programmierung (ISP)
 - ▶ Programmieren direkt im Zielsystem.
 - Chip muss nicht ausgebaut werden, kann fest eingelötet sein.
 - Ermöglicht späteres Update des Programms.
 - ▶ Erfordert zusätzliche Programmierschnittstelle:
 - RS232, SPI
 - JTAG
 - USB, Ethernet,
 - ▶ Erfordert wohldefiniertes Datenformat:
 - Üblich Motorola S2 Hexformat, Intel Hexformat
 - ▶ Programmierlogik ist
 - Teil des Mikrocontrollers oder
 - befindet sich im Flash des Mikrocontrollers (Boot-Programm)

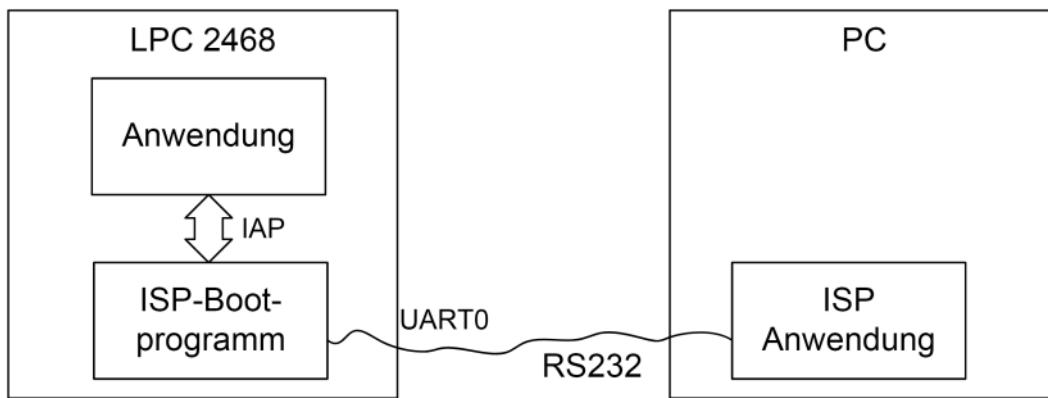
LPC2468: ISP

- ▶ Implementiert im Boot Block
- ▶ Bootprogramm wird gestartet,
 - ▶ wenn Pin P2.10 Low ist oder
 - ▶ wenn kein gültiges Programm geladen ist.
- ▶ Geladenes Programm kann geschützt werden:
 - ▶ Erfordert bestimmtes Pattern in Speicherplatz **0x1FC**



LPC2468: ISP

- ISP-Bootprogramm verwendet UART0
- Programmieren des Controllers ohne zusätzliche Hardware möglich!
- Zusätzlich bietet das Bootprogramm eine einfache Schnittstelle (**In Application Programming, IAP**), die das Programmieren des Mikrocontrollers aus einer eigenen Anwendung heraus ermöglicht.



Übersicht



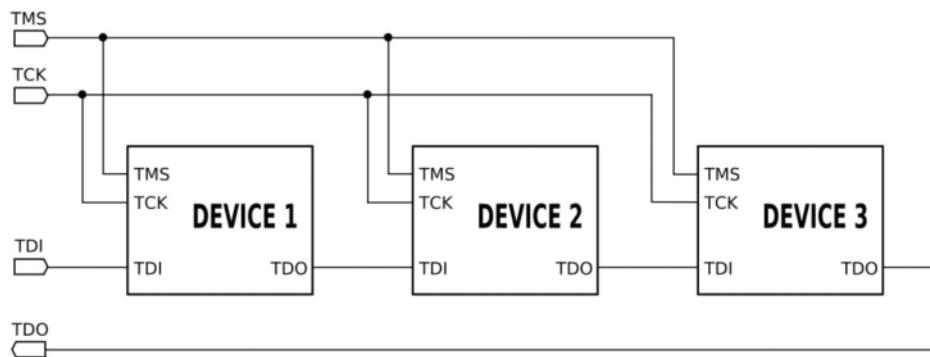
- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
- ▶ Bootprogramm
- ▶ Jtag-Interface
- ▶ Fehlersuche
- ▶ Debugger
- ▶ Logikanalyser
- ▶ In-Circuit Emulator

JTAG (Joint Test Action Group)

- ▶ Standard zum Testen und Debuggen von elektronischen Komponenten direkt in der Schaltung
- ▶ Ursprünglicher Zweck:
 - ▶ Testen der Funktion von integrierten Schaltungen.
 - ▶ Ermöglicht für jeden Anschluss des ICs:
 - Trennung des Anschlusses von der internen Funktion.
 - Beobachten des internen und/oder externen Zustands.
 - Steuern des internen und/oder externen Zustands.
- ▶ Erweiterungen:
 - ▶ Programmieren von Speicher.
 - ▶ Debuggen von Mikrocontroller.
- ▶ Implementierung:
 - ▶ Serieller Port
 - Erlaubt die Verkettung mehrere ICs.
 - ▶ Intern: diverse Schieberegister.

JTAG (Joint Test Action Group)

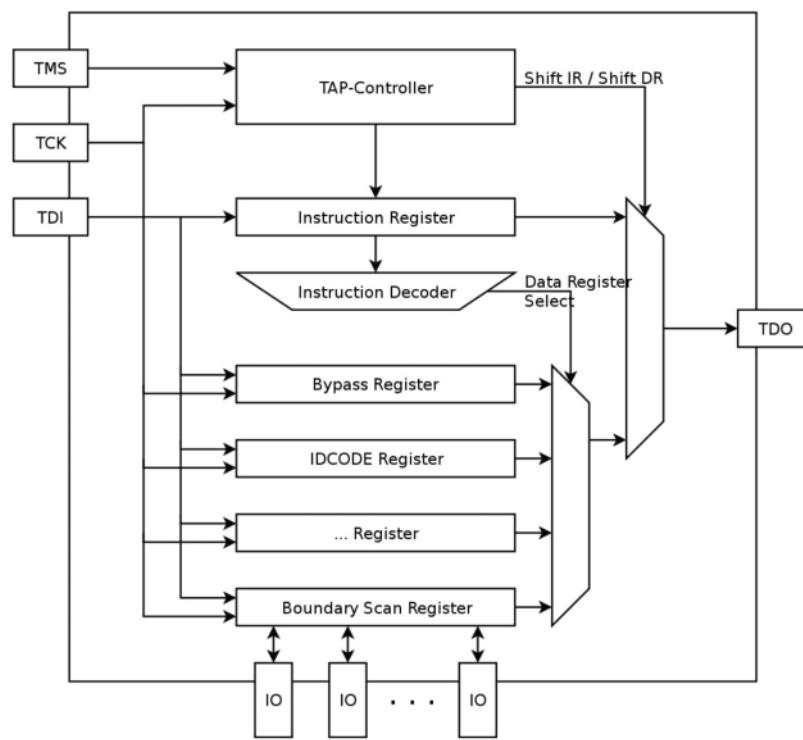
- Verkettung von Komponenten





JTAG

- ▶ Aufbau, Datenregister

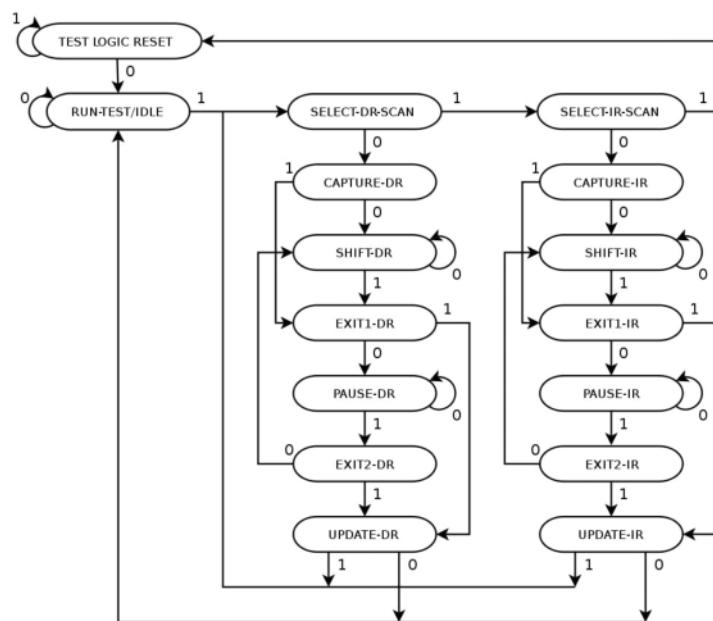


<http://de.wikipedia.org/wiki/JTAG>

CE WS12

JTAG

► Zustandsautomat

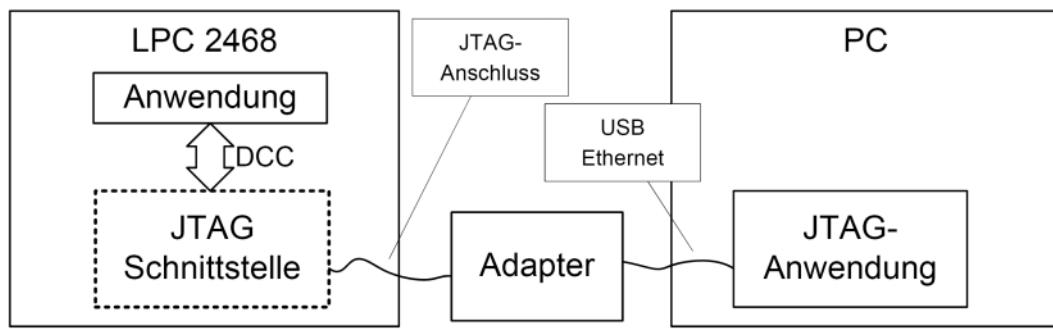


21

<http://de.wikipedia.org/wiki/JTAG>

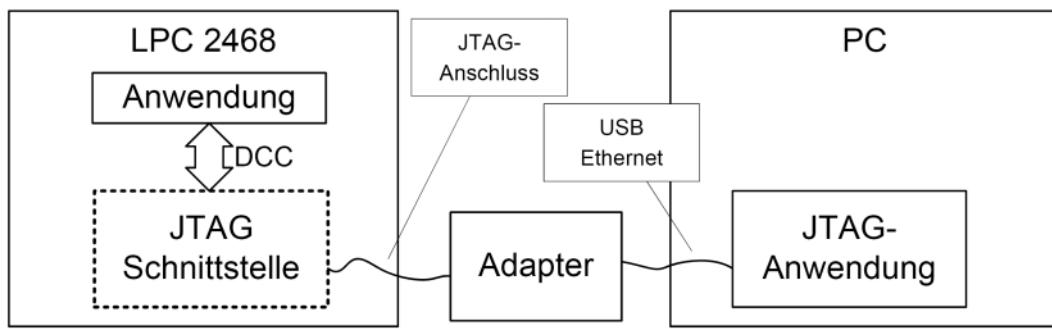
LPC2468: JTAG

- ▶ Debug-Interface Bestandteil des ARM7TDMI-S.
- ▶ Enthält:
 - ▶ Debug-Kommunikationskanal (per SW bedienbar)
 - ▶ Lese- und Schreibzugriff auf Datenbus der CPU
 - CPU wird isoliert.
 - Zugriffe der CPU auf den Datenbus werden auf das JTAG-Interface „umgeleitet“.
 - Erlaubt das „Einspeisen“ eigener Instruktionen.



LPC2468: JTAG

- ▶ Möglicher Ablauf beim Programmieren:
 1. Retten des RAM-Speicherinhaltes .
 2. Übertragen eines kleinen Hilfsprogramm zum „Flashen“.
 3. Übertragen der Flashdaten mittels DCC (Debug Communication Channel).
 4. Hilfsprogramm führt „Flashen“ durch.
 5. Restaurieren des RAM-Speicherinhaltes.

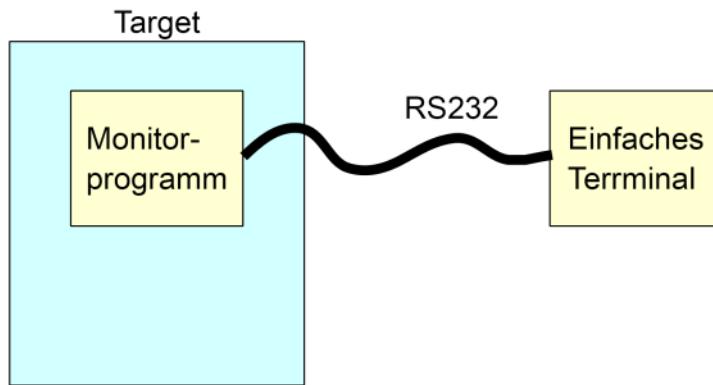


Übersicht

- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
 - ▶ Bootprogramm
 - ▶ Jtag-Interface
- ▶ Fehlersuche
 - ▶ Debugger
 - ▶ Logikanalyser
 - ▶ In-Circuit Emulator

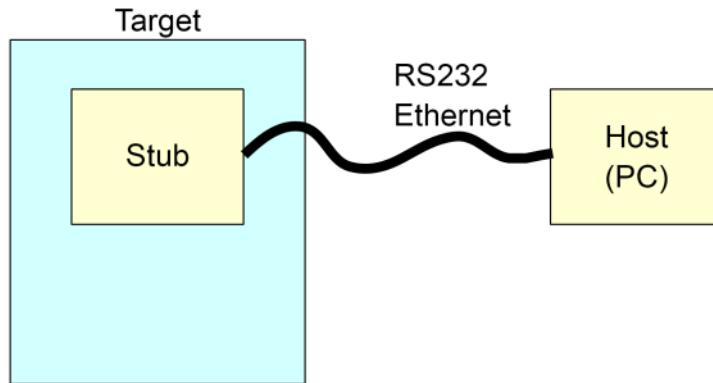


Debugging, ROM-Monitor



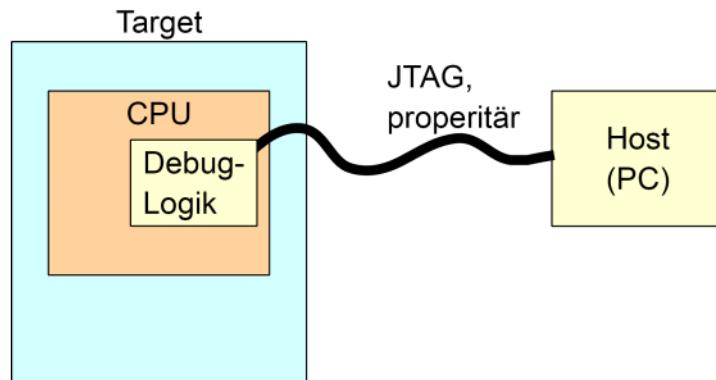
- + Einfache Handhabung
- Ressourcenverbrauch im Target
- Wenig komfortabel:
 - ▶ Kein Sourcecode
 - ▶ Keine Symbole

Debugging, Host-basierter Debugger (Remote Debugging)



- + Geringer Ressourcenverbrauch im Target
- + Geeignet für Sourcecode-Debugging
- Problematisch: Unterbrechung des laufenden Programms (Breakpoints)

Debugging, Host-basierter Debugger (Remote Debugging)



- + Keine Software notwendig im Target
- + Kein Ressourcenverbrauch
- + Breakpoint-Logik integriert
- Allgemeines Problem:
 - ▶ Debuggen beeinflusst zeitlichen Ablauf des Programms
 - ▶ Schlecht geeignet zur Fehlersuche in Echtzeitprogrammen

LPC2468: EmbeddedICE (In-Circuit Emulator?)

- ▶ On-chip Debugger Unterstützung
- ▶ Enthält zwei Watchpoints:
 - ▶ Kombination von **Watchpoint** (Unterbrechung bei Datenzugriff) und **Breakpoint** (Unterbrechung bei Programmzugriff).
 - ▶ Beide Watchpoints können kombiniert werden, z.B.:
 - „Halte, wenn nach Zugriff auf Peripherie ein Taskumschalten erfolgt“
 - „Halte, wenn der Zugriff innerhalb der unteren 256 Bytes aber nicht in den untersten 32 Bytes erfolgt“
- ▶ Debug Communication Channel
 - ▶ Ermöglicht Datenaustausch zwischen Target und Host.
- ▶ Per JTAG können in den Datenbus beliebige ARM-Instruktionen eingefügt werden.
 - ▶ Ermöglicht das Lesen und Schreiben von
 - CPU-Register
 - Speicher, Peripherie

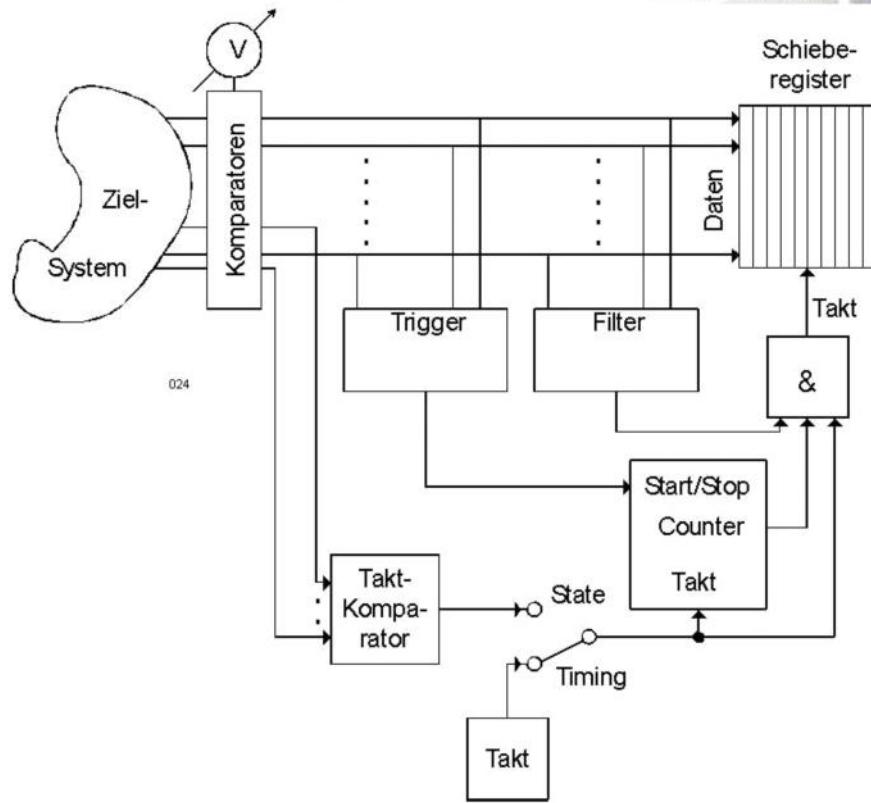
Übersicht

- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
 - ▶ Bootprogramm
 - ▶ Jtag-Interface
- ▶ Fehlersuche
 - ▶ Debugger
 - ▶ Logikanalyser
 - ▶ In-Circuit Emulator

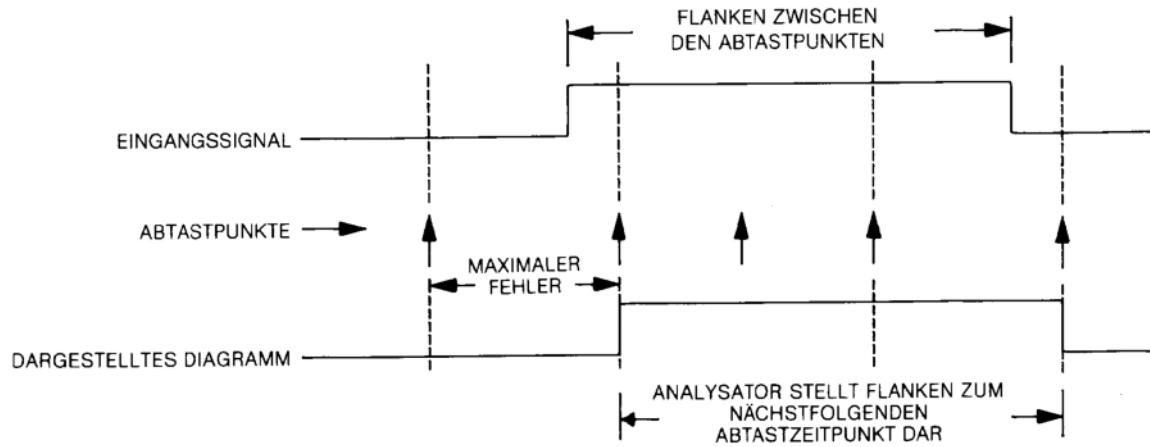


Logikanalysator, Signalaufnahme- Mode

Embedded Systems

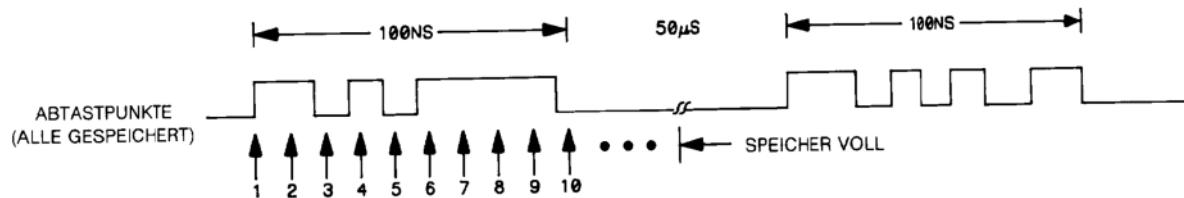


Logikanalysator, Signalaufnahme-Mode

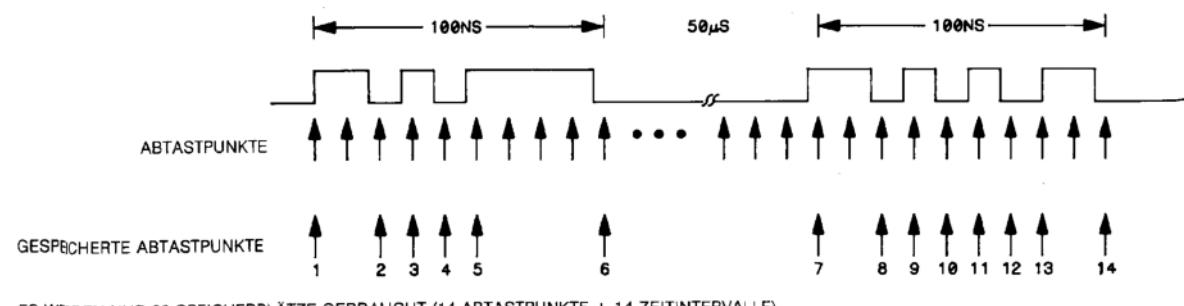


Logikanalysator, Signalaufnahme-Mode

- ▶ Zeitanalyse



- ▶ Zustandsanalyse



Logikanalysator

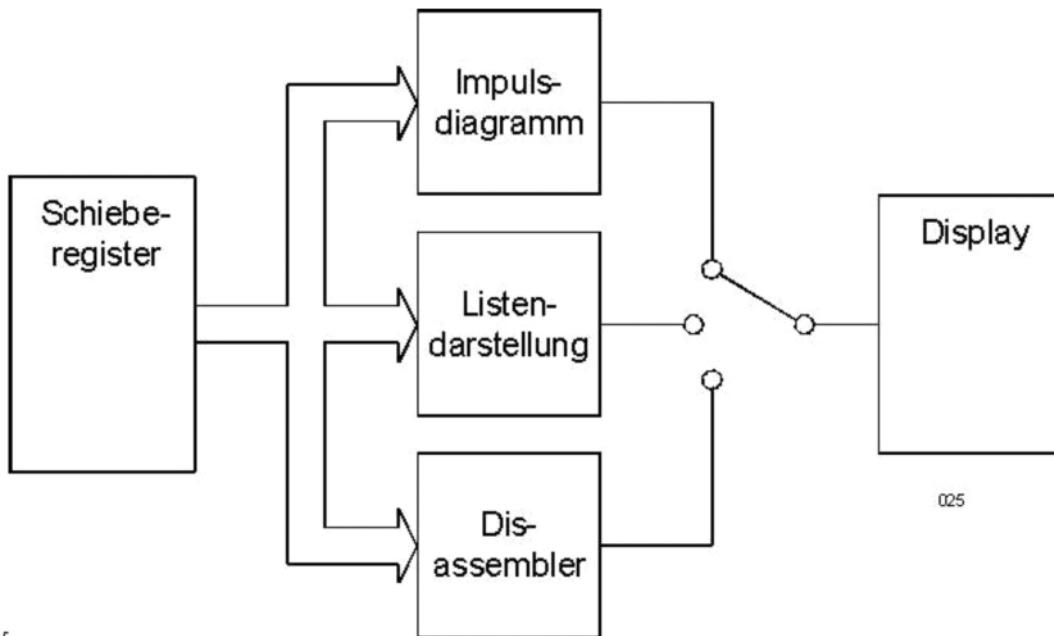
- ▶ **Funktionsprinzip Zeitanalyse**
 - ▶ **Jedes Eingangssignal wird mittels Komparatoren in Rechtecksignale umgewandelt.**
 - ▶ **Signale werden in regelmäßigen Zeitabständen abgetastet.**
 - ▶ **Aufzeichnung erfolgt kontinuierlich und wird erst nach dem Auftreten eines Triggerereignisses beendet. So kann auch der Zeitraum vor dem Triggerereignis erfaßt werden.**
- ▶ **Anwendung der Zeitanalyse**
 - ▶ **Untersuchung der Zeitbeziehungen zwischen vielen verschiedenen Signalen.**
 - ▶ **Keine Analyse der Signalparameter wie z.B. Anstiegszeit. Dafür Oszilloskop benutzen.**
 - ▶ **Meßfehler beträgt maximal eine Abtastperiode.**
 - ▶ **Verringerung des Meßfehlers durch Erhöhung der Taktrate. Dies bedeutet aber eine kürzere Aufzeichnungslänge.**

Logikanalysator

- ▶ Funktionsprinzip Zustandsanalyse
- ▶ Aufzeichnung erfolgt mit einem aus dem zu testenden System abgeleiteten Takt.
Der Zustandsanalysator arbeitet synchron mit dem getesteten System.
- ▶ Zeitinformationen gehen verloren, man erhält Auskunft darüber, was passiert und nicht wann etwas passiert.
- ▶ Es kann so ein deutlich längerer Zeitraum erfasst werden!

- ▶ Anwendung der Zustandsanalyse
 - ▶ Untersuchung der Vorgänge auf einem Mikroprozessor-Bus. Ergebnisse können in Listenform dargestellt werden. Darstellung als Impulsdiagramm zwar möglich aber nicht sinnvoll.
 - ▶ Aus den aufgezeichneten Daten können mittels eines speziellen Disassemblers die abgearbeiteten Mikroprozessor-Befehle entschlüsselt werden.
 - ▶ Mittels eines speziellen Filters können die aufgezeichneten Daten weiter eingegrenzt werden. So kann man z.B. nur die Daten aufzeichnen, die an eine bestimmte Portadresse übertragen wurden.

Logikanalysator, Signalwiedergabe-Mode

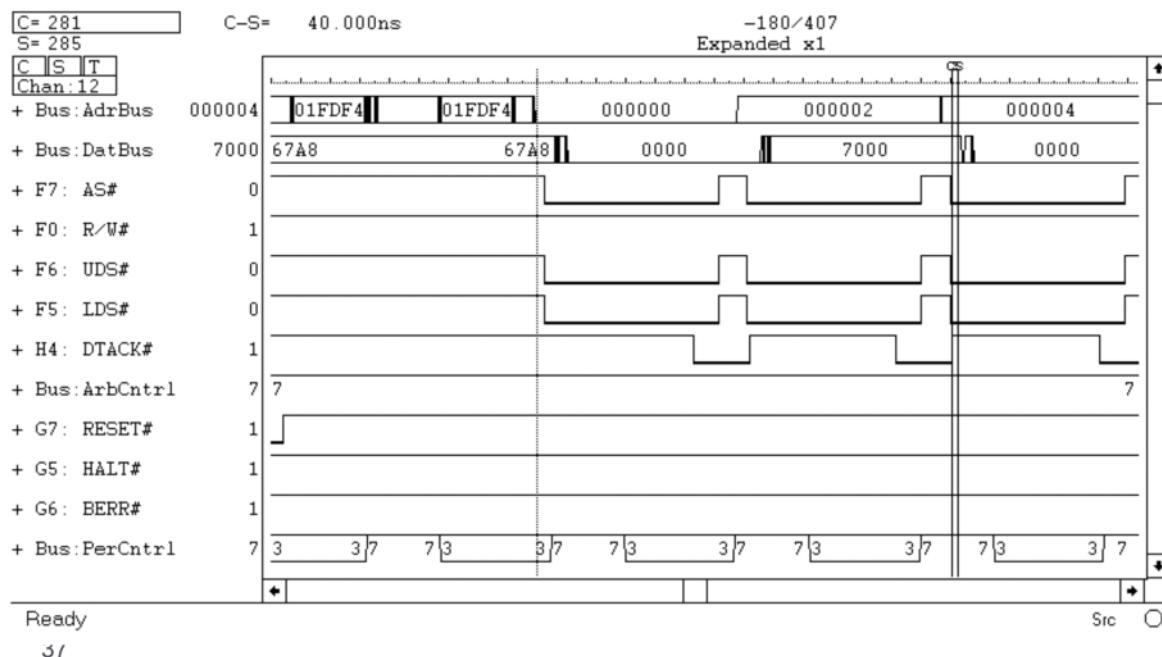


Logikanalysator, Signalwiedergabe-Mode

- ▶ Impulsdigramm
 - ▶ **Hauptsächliche Verwendung im Timing-Mode.**
 - ▶ Logic Analyser funktioniert wie ein Digital-Scope mit sehr vielen Kanälen und komplexen Triggermöglichkeiten.
- ▶ Listendarstellung
 - ▶ **Verwendung hauptsächlich im State-Mode.**
 - ▶ Darstellung der Daten in Tabellenform.
 - ▶ Es können verschiedene Formate gewählt werden: z.B. binär, dezimal, hexadezimal, ASCII-Zeichen.
- ▶ Disassembler
 - ▶ Zusatzsoftware für die Auswertung der gespeicherten Daten.
 - ▶ Setzt eine feste Zuordnung der Kanäle zu den Signalen voraus.
 - ▶ Nur möglich im State-Mode, bei ganz bestimmter Einstellung des Taktkomparators.
 - ▶ Filter darf nicht wirksam sein.

CE WS12

Logikanalysator, Impulsdiagramm

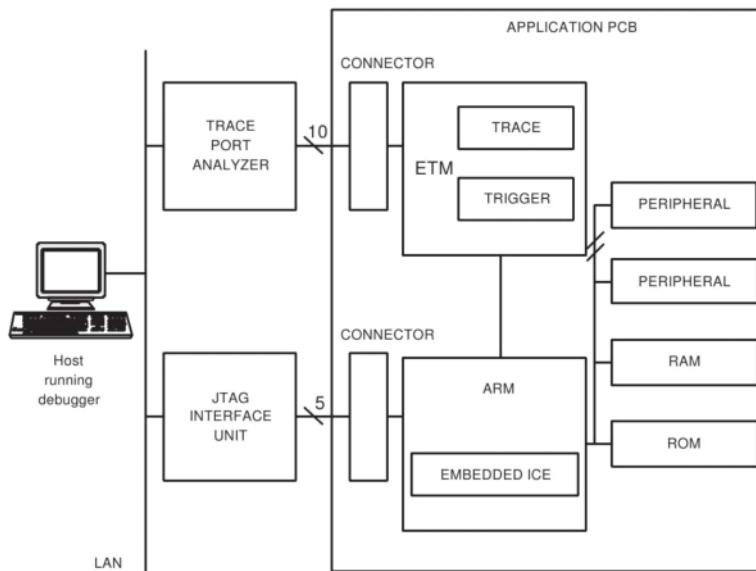


Logikanalysator, Disassembler

Loc	Addr	Data	Mnemonic - 68000 - [Bus] V 4.40
-C----	000004	0000	Supervisor Program Read (Reset:PC)
3	000006	0400	Supervisor Program Read (Reset:PC)
4	000400	61000278	s BSR.W \$0000067A
--S---	006FFC	0000	Supervisor Data Write
7	006FFE	0404	Supervisor Data Write
8	00067A	13FC00070000C001	s MOVE.B #\$07,\$0000C001
12	000682	13FC00070000C041	s MOVE.B #\$07,\$0000C041
13	00C000	--07	Supervisor Data Write
17	00068A	13FC00380000C003	s MOVE.B #\$38,\$0000C003
18	00C040	--07	Supervisor Data Write
22	000692	13FC00380000C043	s MOVE.B #\$38,\$0000C043
23	00C002	--38	Supervisor Data Write
27	00069A	13FC00730000C009	s MOVE.B #\$73,\$0000C009
28	00C042	--38	Supervisor Data Write
32	0006A2	13FC00730000C049	s MOVE.B #\$73,\$0000C049
33	00C008	--73	Supervisor Data Write
37	0006AA	13FC003D0000C00B	s MOVE.B #\$3D,\$0000C00B
38	00C048	--73	Supervisor Data Write
42	0006B2	13FC003D0000C04B	s MOVE.B #\$3D,\$0000C04B
43	00C00A	--3D	Supervisor Data Write
47	0006BA	13FC00030000C00D	s MOVE.B #\$03,\$0000C00D
48	00C04A	--3D	Supervisor Data Write
52	0006C2	13FC00030000C04D	s MOVE.B #\$03,\$0000C04D
53	00C00C	--03	Supervisor Data Write
57	0006CA	13FC002D0000C00F	s MOVE.B #\$2D,\$0000C00F
58	00C04C	--03	Supervisor Data Write
62	0006D2	13FC002D0000C04F	s MOVE.B #\$2D,\$0000C04F
63	00C00E	--2D	Supervisor Data Write

LPC2468: Embedded Trace Module (ETM)

- ▶ Erfasst die Instruktionen, die die CPU ausführt.
- ▶ Infos werden komprimiert über Traceport in Echtzeit ausgegeben.
- ▶ Erfassung mittels externen Trace Port Analyser.



Übersicht

- ▶ Softwareerstellung
- ▶ Download zum Zielsystem
 - ▶ Bootprogramm
 - ▶ Jtag-Interface
- ▶ Fehlersuche
 - ▶ Debugger
 - ▶ Logikanalyser
- ▶ In-Circuit Emulator





In-Circuit Emulator



- + **Integrierter Logik-Analysator**
- + **Eigener Speicher**
 - ▶ **Programm und Daten**
 - ▶ **Dual Ported (kann zur Laufzeit eingesehen werden)**
- **Sehr teuer**
- **Nicht alle CPU-Aktivitäten nach außen sichtbar:**
 - ▶ **Interner Bus Mikrocontroller**
 - ▶ **Cache für Programm und Daten**