

**Informatik**  
HAW Hamburg

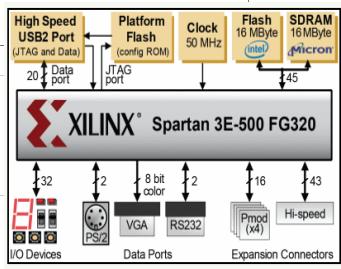
Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences



# Computer Engineering WS 2012

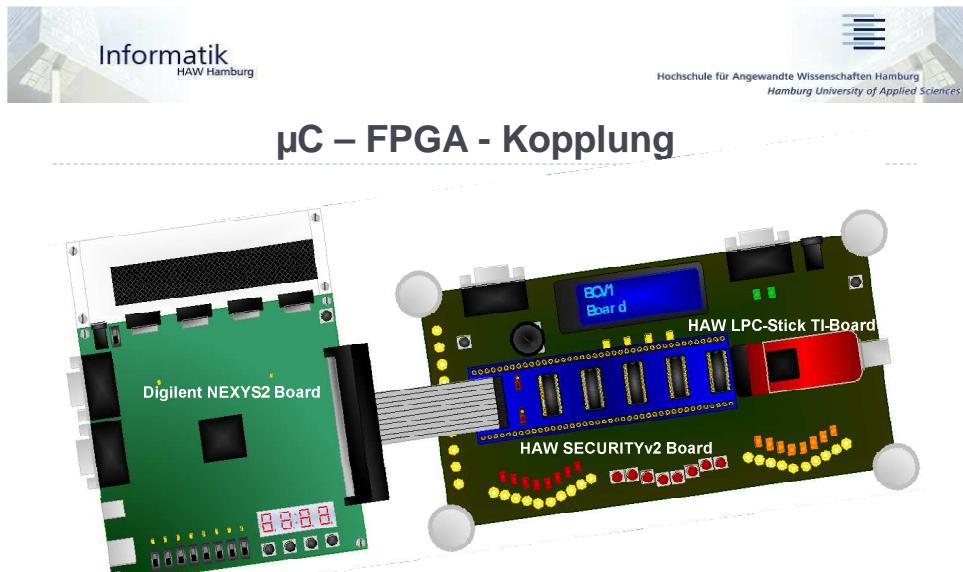
## Inhaltsübersicht

**Prof. Dr. Heitmann - Prof. Dr. Schwarz**



```

graph LR
    HSP[High Speed USB2 Port] <--> JTAG[JTAG port]
    JTAG --> ROM[Platform Flash config ROM]
    ROM --> C[Clock 50 MHz]
    C --> F[Flash 16 MByte Intel]
    F --> S[SDRAM 16 MByte Micron]
    ROM <--> S
    ROM <--> IO[32 I/O Devices]
    ROM <--> DP[2 Data Ports]
    ROM <--> VGA[VGA]
    ROM <--> RS232[RS232]
    ROM <--> Pmod[Pmod (x4)]
    ROM <--> HS[Hi-speed]
    ROM <--> EC[43 Expansion Connectors]
  
```



## Zusammenfassung CE, DS, RS

► Schwerpunkte:

- µController: Interrupts - Peripherie
- Prozessorelement: Daten- u. Steuerpfad im FPGA
- Kombination von SW-HW
- Rechnerstrukturen: RISC - Caches

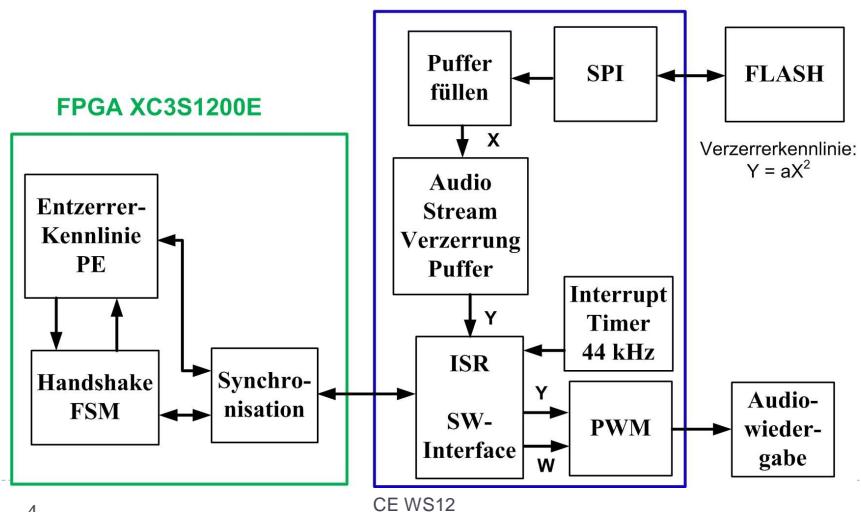
► 4 LVS + 8 Labortermine

3

CE WS12

## CE-Labor-System

$\mu$ C LPC 2468



4

CE WS12



## Laboraufgaben

### **FPGA**

- Schieberegister
- FSM-Timer-Konzept ; Aufzugsteuerung
- Prozessorelement; Multizyklus-DP rechnet nichtlineare Kennlinie

### **μController**

- UART, Interrupts und ISR
- SW-Interface: SPI Interface zu einem EEPROM/Flash-Speicher
- Timer und PWM 44 kHz

### **Integration des Audio-Laborsystems als eine Aufgabe**

- Synchronisation, Handshake-FSM, PE-Ankopplung zur Entzerrung
- SW-Interface; MPR3 Decodierung, Verzerrung; entzerrte Daten über PWM ausgeben



## Vorlesungsinhalte

### **FSMs**

- Vergleich von synchronen Automatenstrukturen
- Entwurfsmethodik; Automatenbeschreibungen
- Entkopplung von Zustandsautomaten

### **Architekturentwurf für RTL-Entwürfe**

- ASM-Diagramme für Prozessorelemente
- Ressource Sharing für PE-Funktionseinheiten
- Datenpfad in Pipelinestruktur

### **Synchronisation**

- Clock-Konzepte; Multiratensysteme
- Synchronisation asynchroner Eingangssignale
- Kommunikation zwischen asynchronen Clock-Bereichen



## Vorlesungsinhalte

### Aufbau Mikroprozessor

- Prozessorbus, Adressdekoder speziell Zeitverhalten
- Interruptverarbeitung
- DMA

### Mikroprozessor-Peripherieeinheiten

- Parallele Ein- und Ausgabe
- Timer, Systemkonfiguration, Watchdog, Serielle Schnittstelle

### Programmierung Mikrocontroller

- Treiberprogrammierung
- Startup-Code
- Laufzeitumgebung, Aufteilung ROM/RAM, Sektionen, Systemtests

### Externe Beschaltung

- SPI



## Vorlesungsinhalte

### ALU

- RISC mit Pipeline; Phasendiagramm; Daten-, Kontroll- u. Struktur-Hazards
- Implikationen von Pipelines; Performancesteigerung, HW-Bedarf
- Prozessorklassifikation; Load-Store/Register-Speicher; Harvard-v.Neuman; Pseudo-Harvard

### Speicher

- Speicherhierarchie
- Aufbau und Funktion von Caches;
- Direct-Mapped, n-Wege/Teilassoziativ, Vollassoziativ
- Einfluss auf Antwortzeiten; Cache-Einsatz steuern



## CE-Sequenz WS 12/13

W1 KW39	W2	W3	W4	W5 KW43	W6	W7	W8	W9	W10	W11 KW49	W12	W13 KW51	W14 KW2	W15 KW3
VD VD	VDP	VP VD	VDP	VDP	VDP	VP	VDP	VDP	VDP	VDP	VDP	Übung	VR	KLV
---	---	---	L1	L1/ L2	L2	L3	L3/ L4	L4	L5	L5/ L6	L6/ L7	L7	L8	L8

D: Digitale Systeme; P: µController; R: Rechnerstrukturen; KLV: Klausurvorbereitung; L: Labortermin

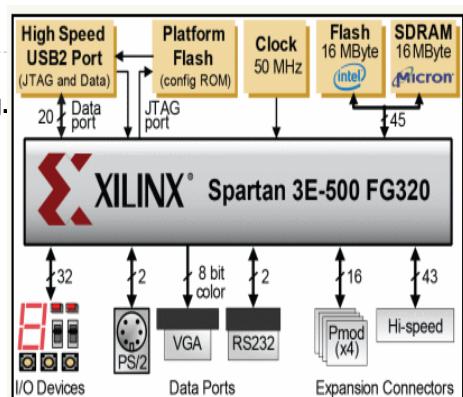
9

CE WS12



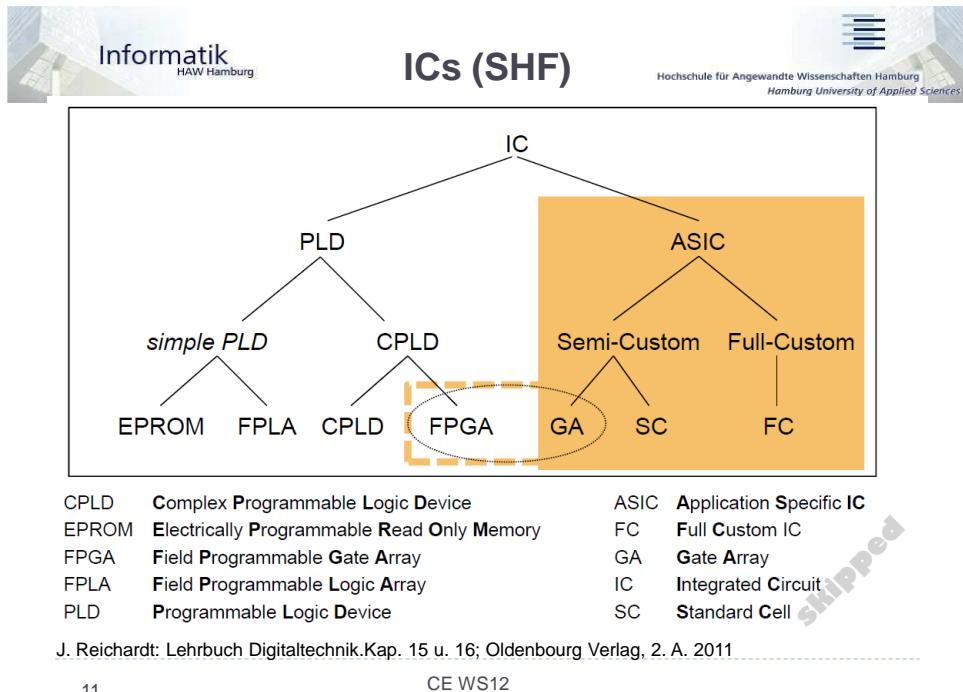
## Digilent Nexys2

- ▶ Xilinx XC3S1200E
- 19500 LEs, 63kB RAM, 28 MUL
- ▶ USB2: board power, device config. and high-speed data transfers
- ▶ 16MB fast Micron SDRAM
- ▶ 16MB Intel Flash Flash ROM
- ▶ Xilinx Platform Flash ROM
- ▶ 50MHz Oscillator + Socket
- ▶ 59 GPIOs
- ▶ On-board I/O includes 8 LEDs, 4 7-segment display, 4 pushbuttons, 8 slide switches
- ▶ 119\$=91€
- ▶ ADC-, DAC-, Video-Decoder-, RS232-, 10Mbit Ethernet-Module



10

CE WS12



Informatik  
HAW Hamburg

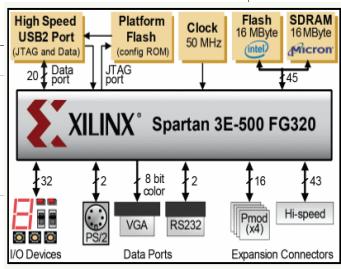


Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Computer Engineering WS 2012

### Digitale Systeme Ziele

Prof. Dr. B. Schwarz



```

    graph TD
        HSP[High Speed USB2 Port] <--> JTAG[JTAG port]
        JTAG --> ROM[Platform Flash config ROM]
        ROM --> C[Clock 50 MHz]
        C --> F[Flash 16 MByte Intel]
        C --> S[SDRAM 16 MByte Micron]
        F <--> S
        IOD[I/O Devices] <--> PS2[PS/2]
        IOD <--> VGA[VGA]
        IOD <--> RS232[RS232]
        IOD <--> Pmod[Pmod (x4)]
        IOD <--> HS[Hi-speed]
        DS[Data Ports] <--> PS2
        DS <--> VGA
        DS <--> RS232
        DS <--> Pmod
        DS <--> HS
    
```

Informatik  
HAW Hamburg

### Digitale Systeme

Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

- 1. Ziele für die Anwendung in SoC-FPGAs**
- 2. Entwurf und Verhalten synchroner Automaten**  
**Vergleich der Automatenstrukturen: Mealy, Moore**  
**Entwurfsmethodik; Beispiel Sequenzerkennung**  
**VHDL-Model, Timing, Synthese**  
**Zwei-Prozess VHDL-Automatenbeschreibungen**  
**Entkopplung von Zustandsautomaten**  
**Entwurf eines sequentiellen Addierers**

### 3. Architektursynthese

**Prozessorelement für eine S-Kurvenapproximation**  
**ASM-Diagramm für das**  
**Multizyklus-Prozessorelement**  
**Gemeinsame Nutzung von Arithmetik-**  
**Funktionseinheiten: Ressource Sharing**  
**Gemeinsame Register- / Speicher-Nutzung**  
**Datenpfad mit Fixed-Point Arithmetik im Q-Format**  
**Datenpfad in Pipelinestruktur**  
**Rechensysteme mit Ausgangsrückführung**

3

CE - DS Ziele

### 4. Synchrone Systeme

**Clock-Konzepte**  
**Asynchrone Eingangssignale**  
**Synchronisationsschaltung für lange Impulse**  
**Synchronisationsschaltung für kurze Impulse**  
**MTBF bei Flipflops mit metastabilen Zuständen**  
**Kommunikation zwischen asynchronen**  
**Clock-Bereichen**  
**Vier-Phasen Handshake**

4

CE - DS Ziele

## Literatur

- [1] T. J. Wakerly: Digital Design Principles & Practices; Prentice Hall 2006; 4<sup>th</sup> edition
- [2] D. D. Gajski: Principles of Digital Design; Prentice Hall 1997
- [3] P. J. Ashenden: Digital Design. An Embedded Systems Approach Using VHDL. Morgan Kaufmann 2008
- [4] R. Sass, A.G. Schmidt: Embedded Systems Design with Platform FPGAs. Morgan Kaufmann 2010
- [5] J. Reichardt; B. Schwarz: VHDL Synthese. Entwurf digitaler Schaltungen und Systeme; Oldenbourg Nov. .2012, 6. Auflage
- [6] W. Wolf: Computers as Components: Principles of Embedded Computing System Design. Morgan Kaufmann 2008, 2<sup>nd</sup> edition
- [7] W. Wolf: High-Performance Embedded Computing; Morgan Kaufmann 2007
- [8] A. Sloss, D. Symes, Ch. Wright: ARM System Developer's Guide; Morgan Kaufmann 2004
- [9] U. Brinkschulte, Th. Ungerer: Mikrocontroller und Mikroprozessoren; 2. Auflage Springer 2007
- [10] D. A. Patterson, J. L. Hennessy: Computer Organisation & Design; The Hardware/Software Interface; Morgan Kaufmann 2008, 4<sup>th</sup> edition

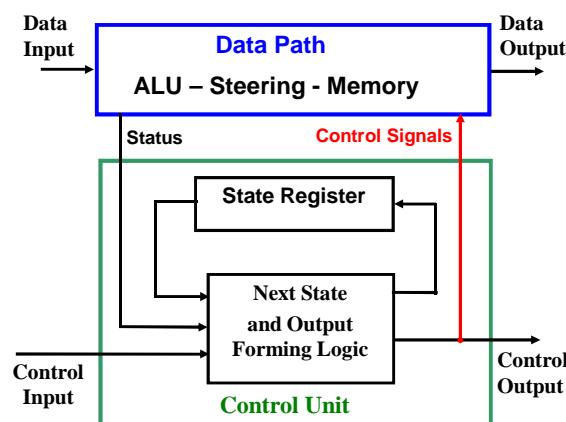
5

CE - DS Ziele

## 1. Ziele

### Entwurf von Prozessorelementen für SoC-Anwendungen

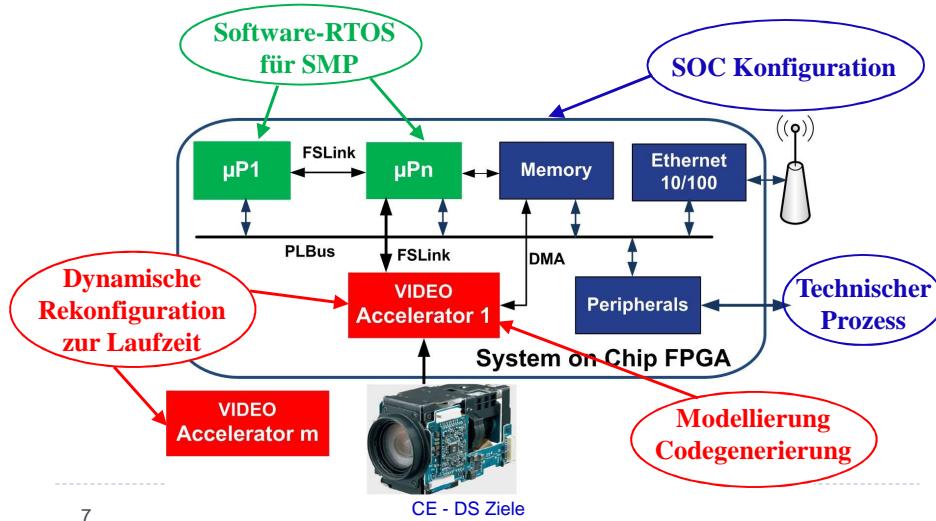
#### in der Audiosignal- und Bildverarbeitung



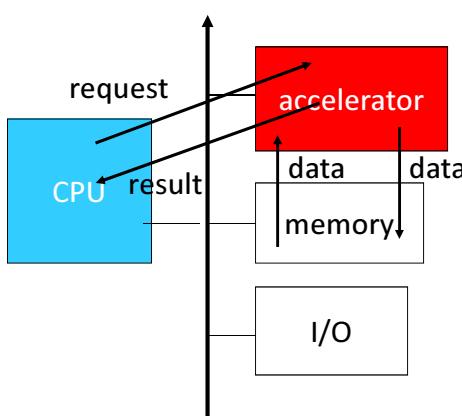
6

CE - DS Ziele

## System on Chip für eingebettete Anwendungen



## µC mit Beschleuniger

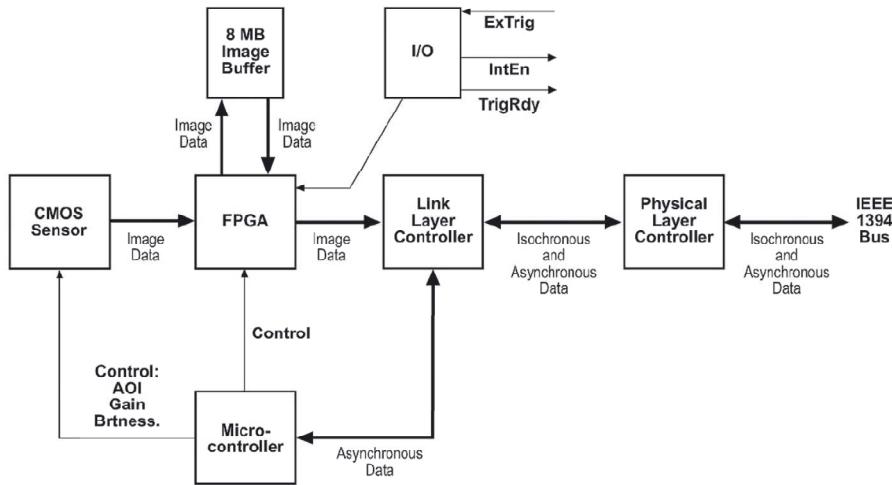


CE - DS Ziele

8

- **Co-Prozessoren** führen Instruktionen aus, die von der CPU angestoßen werden; keine Nebenläufigkeit
- **Beschleuniger** arbeiten wie µP-Peripherie als Bus-Teilnehmer;
  - Steuerung über Register.
  - Realisierung mit rekonfigurierbarer HW: FPGAs
  - Modellierung paralleler Funktionen mit HDLs: VHDL
  - Vorteile gegenüber µC mit höherer Frequenz:
    - I/O in Echtzeit
    - Geringerer Energieverbrauch
    - Datenflussanwendungen mit hohen Datenraten

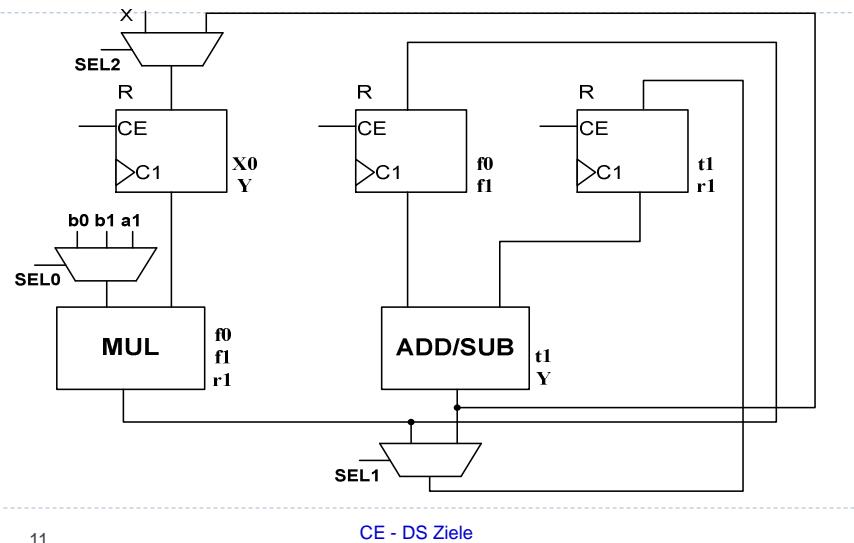
## **μC-FPGA-Konfiguration: CCD-Kamera Basler A602f mit 491x656 Pixel, 105 Bilder/sek**



## **Architekturentwurf**

- **Strukturierung eines digitalen Systems als Prozessorelement** in einen **Datenpfad** und einen **Steuerpfad**.
- **It is appropriate to separate both parts because to each part different design strategies and optimisation methods can be applied.**
- Der **Datenpfad** enthält alle Operationen zur Datenmanipulation. Er wird aus Schaltnetzketten (z.B. ALU) mit Registern zur Datenzwischenspeicherung gebildet. Seine Operationen werden durch Steuersignale kontrolliert. Er selbst erzeugt Statussignale.
- Der **Steuerpfad** ist als taktsynchroner Zustandsautomat aufgebaut. Abhängig von den Eingangs- und Statussignalen des Datenpfads werden die Aktionen im Datenpfad über **Steuersignale** initiiert. CE - DS Ziele

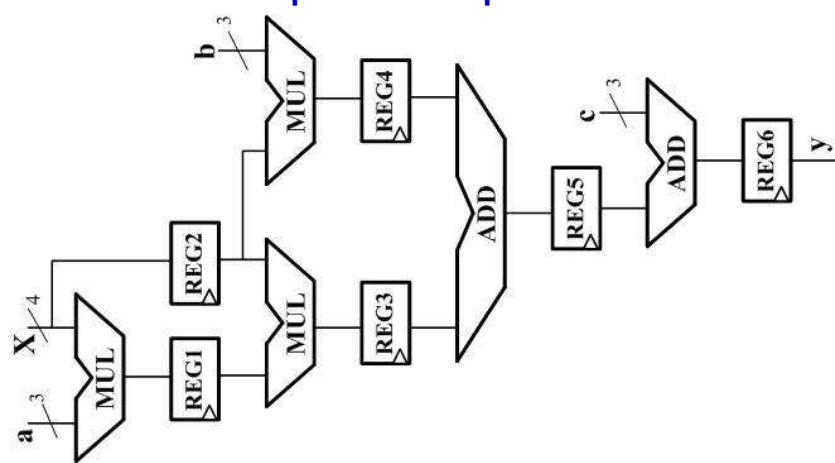
### Multizyklus-Datenpfad



11

CE - DS Ziele

### Pipeline-Datenpfad



12

CE - DS Ziele

- **Technik: Register Transfer Level**
- **Entwürfe auf der Grundlage von Architekturübersichtsbildern**
  
- **Coding Style: synthesegerechtes VHDL**
- 1076-2004 - IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis** P. Ashenden: The VHDL Designers Guide. 3rd edition 2008 Morgan Kaufmann; chap. 21, p. 633 - 665
- Subset des 1076-2008 IEEE Standard VHDL Language Reference Manual**
  
- **ModelSim PE VHDL-Simulation: 1076-2008**
- **Xilinx XST Synthese: partially implemented 1076-2002**
- **Synthesis and Simulation Design Guide Guide UG626 (v 14.1)**  
**May 8th, 2012; 171 pages**

13

CE - DS Ziele

## Firmen in Hamburg mit innovativen Produkten

- **Basler** in Ahrensburg: Industriekameras; 30% Kamera Entwicklungsabteilung mit HAW Absolventen
  
- **Allied Vision Technologies** in Ahrensburg:  
Herr R. German BA; Herr N. Andreae MA
  
- **Dermalog Identification System** in HH: Biometrische Erkennungsmethoden  
Herr R. Weber BA; Herr D. Blauthut BA; Herr C. Schulz MA  
**Master-Forschungsprojekt : Biometric Graphic Acceleration**
  
- **Ibeo Automotiv Systems** in Rahlstedt: Laserscanner Systeme  
Herr Ö. N. Püsküll BA

14

CE - DS Ziele

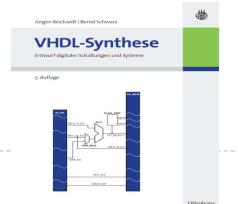


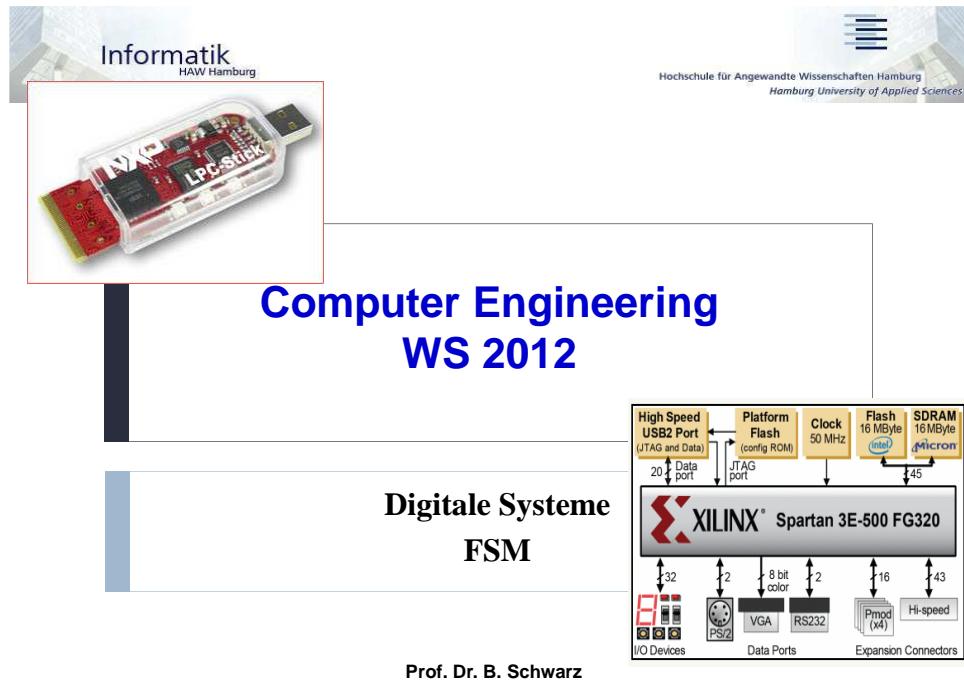
## Aktuelle Referenzen

- **B. Schwarz:** Ein FPGA-basiertes System-on-Chip in der Echtzeitbildverarbeitung.  
Informatik Aktuell; Eingebettete Systeme: Echtzeit 2010; Springer Verlag  
08.11.2010
- **H. Wilken, M. Kirschke, B. Schwarz:** Configuring a Dual-MicroBlaze-Xilkernel System. Memory segmentation and data exchange strategies take a significant share on the implementation of deeply coupled master/slave multiprocessor systems.  
Xilinx Xcell Journal issue 77 2011.
- **F. Opitz, E. Sahak, B. Schwarz:** Accelerating Distributed Computing with FPGAs. Development of a Distributed SoC Network by employing the Xilinx Dynamic Partial Reconfiguration Technology. Xilinx Xcell Journal issue 79 2012.
- **Xilinx Donation-Program 2010-2012:** PRMs controlled as RTOS Tasks.  
Partial-Reconfiguration und 50 ISE-DS Lizenzen für das HPEC-Team.  
PR 4x900€; ISE-DS 4x900€ (Lehre); Boards 2000€
- **J. Reichardt, B. Schwarz:** VHDL-Synthese.  
Oldenbourg Wissenschaftsverlag, 5. Auflage 2009  
6. Auflage November 2012

15

CE - DS Ziele

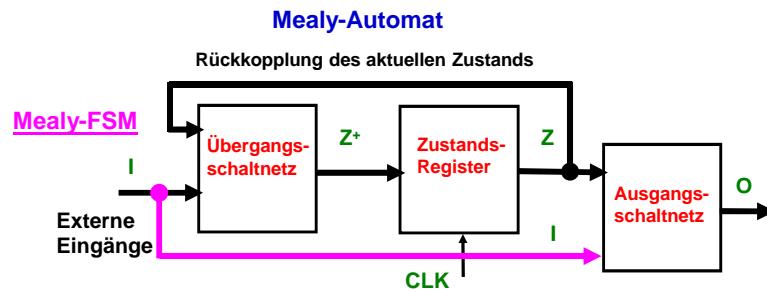




- 2. Entwurf und Verhalten synchroner Automaten**
- Automatenstrukturen**
- Entwurfsmethodik; Beispiel Sequenzerkennung**
- VHDL-Model, Timing, Synthese**
- Zwei-Prozess VHDL-Automatenbeschreibungen**
- Entkopplung von Zustandsautomaten**
- Entwurf eines sequentiellen Addierers (Mealy, Moore)**

## 2. Entwurf und Verhalten synchroner Automaten

### 2.1 Automatenstrukturen



Beim Mealy-Automaten gilt:

$$Z^+ = f(E, Z) \quad \text{und} \quad O = f_{\text{Mealy}}(I, Z)$$

3

CE - DS 2 FSM

## Moore- und Medvedev-Automaten

Beim Moore-Automaten gilt:

$$Z^+ = f(E, Z)$$

und

$$O = f_{\text{Moore}}(Z)$$

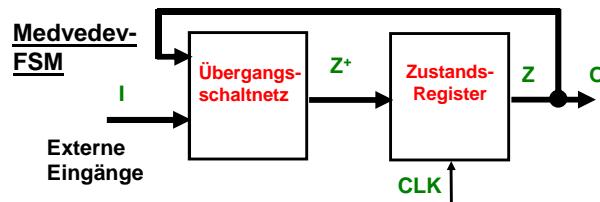
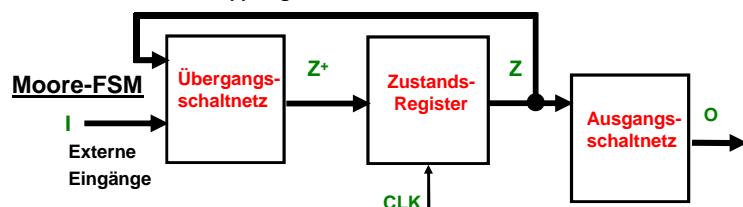
Der Medvedev-Automat ist ein Moore-Automat, bei dem die Ausgänge den Zustands-Flipflops entsprechen:

$$Z^+ = f(E, Z)$$

und

$$O = Z$$

Rückkopplung des aktuellen Zustands



4

CE - DS 2 FSM



## 2.2 Enwurfsmethodik 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 =...,01,11,10,...** erkannt werden und am Ausgang **A** des taktsynchronen 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.

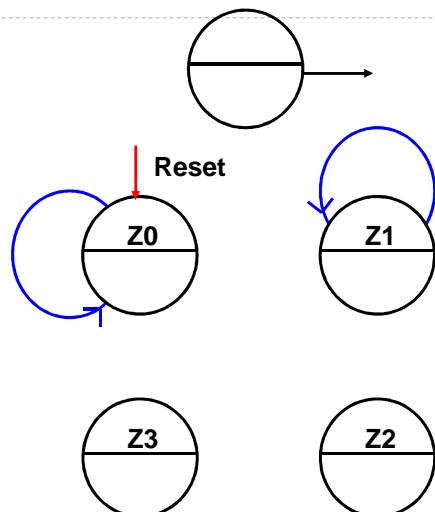
6

CE - DS 2 FSM

## Moore-Automat

### Zustände:

- Z0 : Anfangszustand , warte auf E="01". Dies ist der Zustand nach RESET.
- Z1: E="01" wurde erkannt, warte auf "11"
- Z2: E="11" wurde erkannt, warte auf "10"
- Z3: E="10" wurde erkannt, gebe A='1' aus.



7

CE - DS 2 FSM

## Zustandscodierung, Folgezustands- u. Ausgangstabellen zur Sequenzerkennung

Heuristischer Ansatz für die Zustandscodierung:

Zustand	Z <sub>1</sub>	Z <sub>0</sub>
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

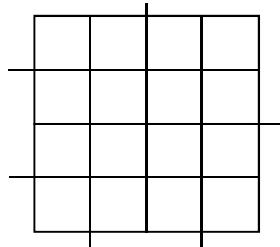
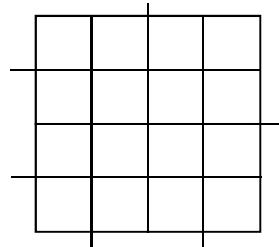
No.	Z <sub>3</sub>	Z <sub>2</sub>	Z <sub>1</sub>	Z <sub>0</sub>
1	00	01	10	11
2	00	01	11	10
3	00	10	01	11
4	00	10	11	01
...	...	...	...	...

8

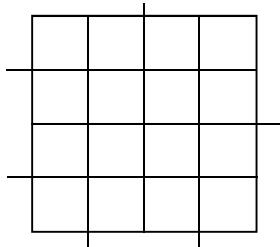
CE - DS 2 FSM

	Z <sub>1</sub>	Z <sub>0</sub>	E <sub>1</sub>	E <sub>0</sub>	Z <sub>1</sub> <sup>+</sup>	Z <sub>0</sub> <sup>+</sup>	A
Z0	0	0	0	0			
	0	0	0	1			
	0	0	1	0			
	0	0	1	1			
Z1	0	1	0	0			
	0	1	0	1			
	0	1	1	0			
	0	1	1	1			
Z2	1	0	0	0			
	1	0	0	1			
	1	0	1	0			
	1	0	1	1			
Z3	1	1	0	0			
	1	1	0	1			
	1	1	1	0			
	1	1	1	1			

## KV-Minimierung der Übergangs- und Ausgangsschaltnetze

 $Z_0^+$ : $Z_0^+ = \dots$  $Z_1^+$ : $Z_1^+ = \dots$ 

A:

 $A = \dots$ 

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 ZUSTAENDE is (Z0, Z1, Z2, Z3);           -- Aufzählungstyp
signal ZUSTAND, FOLGE_Z: ZUSTAENDE;           -- 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

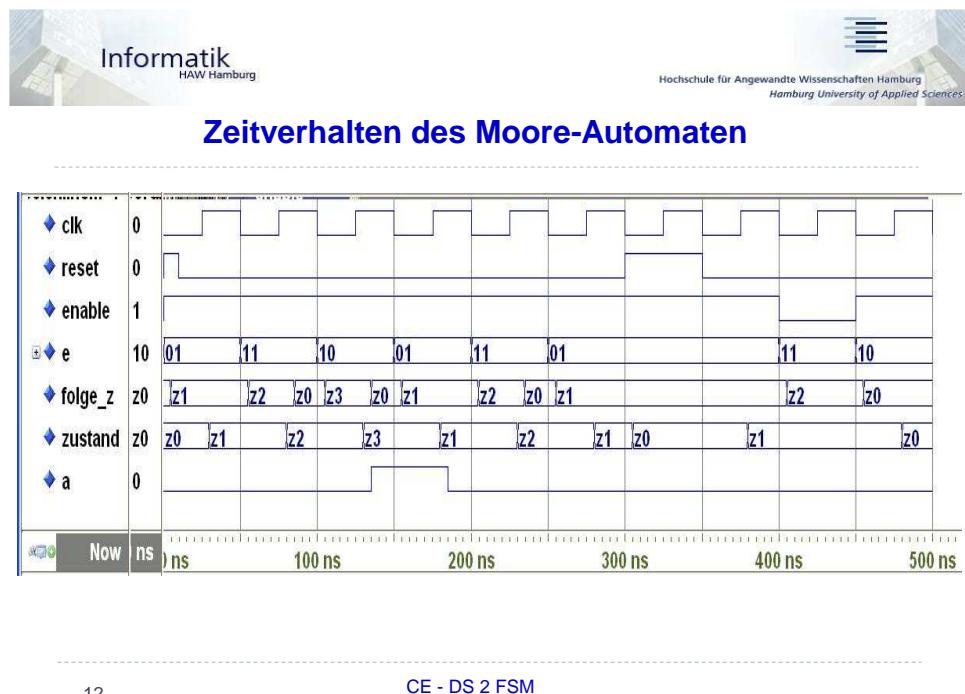
**Informatik**  
HAW Hamburg

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 = "11" 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 = "10" 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 = "01" 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;
-----
```

11 CE - DS 2 FSM



## 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));

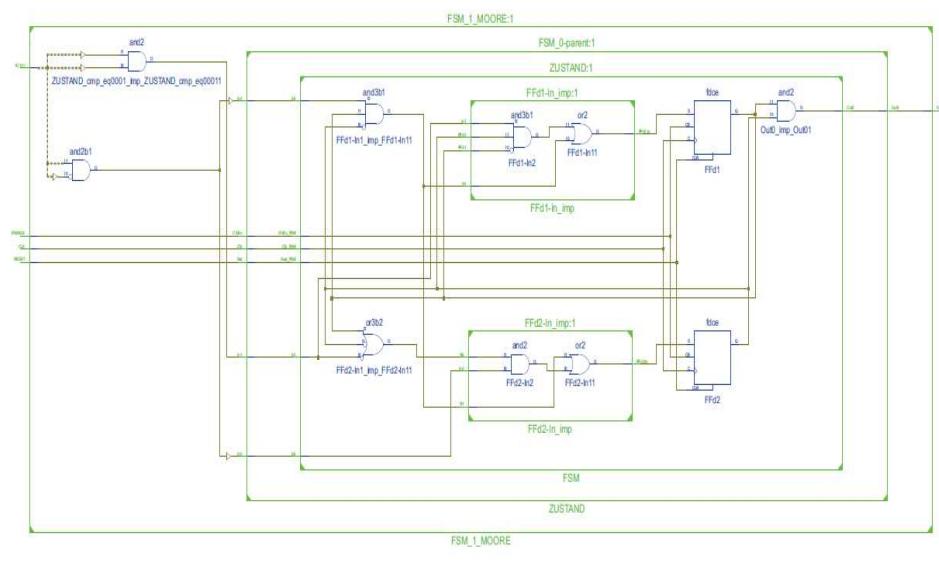
```

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

13

CE - DS 2 FSM

## RTL Schematic




**Informatik**  
 Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

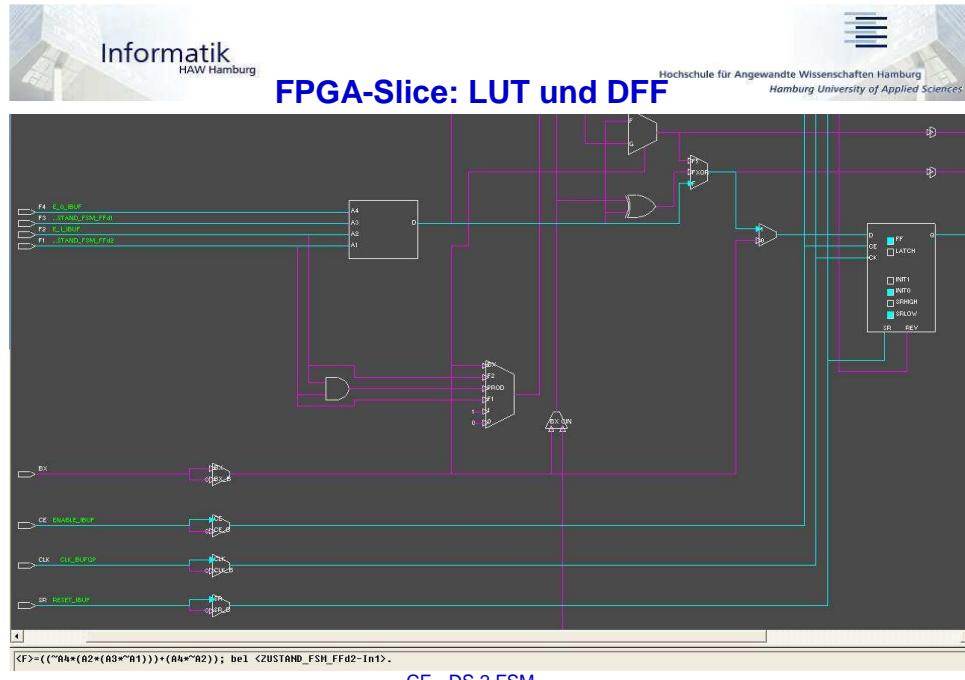
## Synthese-Reportauszug

```
-- Synthesizing Unit <FSM_1_MOORE>.
  Related source file is "C:/Users/Dr. B.
  Schwarz/Documents/A_MSI_ISE_work/ce/fsm_1_moore/seq_moore.vhd".
  Found finite state machine <FSM_0> for signal <ZUSTAND>.

  -----
  | States          | 4
  | Transitions    | 10
  | Inputs          | 3
  | Outputs         | 1
  | Clock           | CLK
  | Clock enable   | ENABLE      (rising_edge)
  | Reset           | RESET       (positive)
  | Reset type     | asynchronous
  | Reset State    | z0
  | Power Up State | z0
  | Encoding        | sequential
  | Implementation  | LUT
  -----
  Summary:
  inferred 1 Finite State Machine(s).
  Unit <FSM_1_MOORE> synthesized.
  * Advanced HDL Synthesis
  =====
  Optimizing FSM <ZUSTAND/FSM> on signal <ZUSTAND[1:2> with sequential encoding.
  -----
  State | Encoding
  -----
  z0   | 00
  z1   | 01
  z2   | 10
  z3   | 11
  -----
```

15

CE - DS 2 FSM

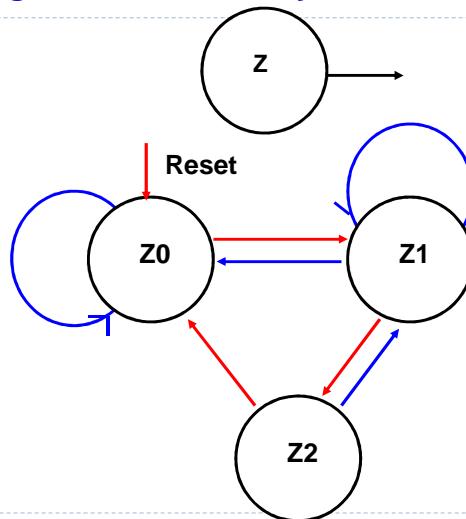


16

CE - DS 2 FSM

## 2.4 Impulsfolgeerkennung durch einen Mealy-Automaten

- **Impulsfolge:**  
 $E = \dots 01, 11, 10, \dots$
- Da sich beim Mealy-Automaten das Eingangssignal  $E$  direkt auf das Ausgangssignal  $A$  auswirken kann, wird ein Zustand eingespart.
- Allerdings muss deshalb damit gerechnet werden, dass beim Ausgangssignal  $A$  des Mealy-Automaten Hazards auftreten!



17

CE - DS 2 FSM

## 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 SEQUENZ of FSM_1_MEALY is
  type ZUSTANDE is (Z0, Z1, Z2);           -- Aufzählungstyp
  signal ZUSTAND, FOLGE_Z: ZUSTANDE := Z2;
  attribute SAFE_RECOVERY_STATE: STRING;
  attribute SAFE_RECOVERY_STATE of ZUSTAND: signal is "Z1";
begin
  Z_SPEICHER: process(CLK, RESET) -- Zustandsaktualisierung
  begin
    Z_SN: process(E, ZUSTAND)          -- Folgezustandsberechnung
    begin
      FOLGE_Z <= Z0 after 5 ns;       -- Defaultzuweisung
      case ZUSTAND is
        when Z0 => if E = "01" then FOLGE_Z <= Z1 after 5 ns;
        when Z1 => if E = "11" then FOLGE_Z <= Z2 after 5 ns;
        elsif E = "01" then FOLGE_Z <= Z1 after 5 ns;
        when Z2 => if E = "01" then FOLGE_Z <= Z1 after 5 ns;
        when others => null;
      end case;
    end process Z_SN;
    A_SN: process(E, ZUSTAND)          -- Ausgangssignalberechnung
    begin
      A <= '0' after 5 ns;            -- Defaultzuweisung
      if (ZUSTAND = Z2 and E = "10") then A <='1' after 5 ns;
      end if;
    end process A_SN;
  end process Z_SPEICHER;
end SEQUENZ;
  
```

18

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;
-- Periodisches Taktsignal 20 MHz

```

19

CE - DS 2 FSM



## 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 30 ns;
E_I <= "10";   wait for 20 ns;--A_I<= '1'
E_I <= "00";   wait for 15 ns;-- Hazard
E_I <= "10";   wait for 85 ns;--A_I <= '1'
E_I <= "01";   wait for 50 ns;
ENABLE_I <= '0';
ENABLE_I <= '1';
E_I <= "11";   wait for 50 ns;-- Z1 fest
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;

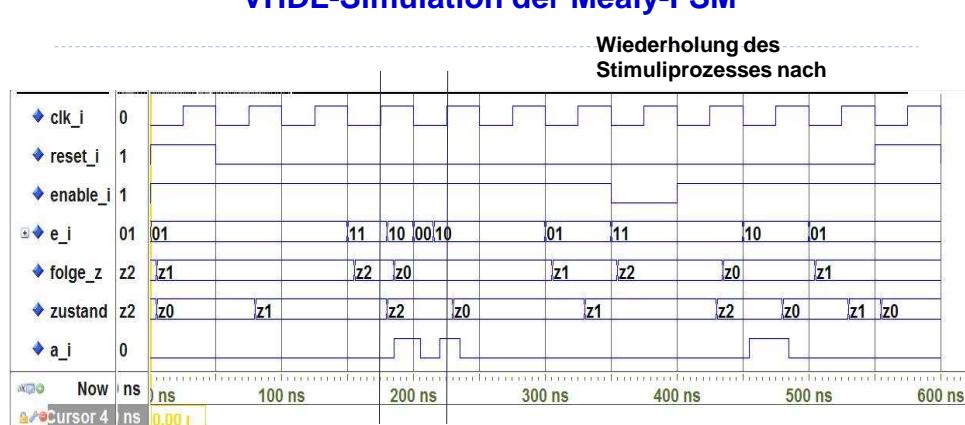
```

Wiederholung der Stimulisequenz ab:

20

CE - DS 2 FSM

**HAW Hamburg** Hochschule für Angewandte Wissenschaften Hamburg  
University of Applied Sciences



Hazard in E = 10,00,10

21

CE - DS 2 FSM

## Synthese-Reportauszug

```

Synthesizing Unit <FSM_1_MEALY>.
Related source file is "C:/Users/.../Documents/A_MSI_ISE_work/ModelSim_source/Lectures/cel/
seq_mealy.vhd".
Found finite state machine <FSM_0> for signal <ZUSTAND>.

-----
| States           | 3
| Transitions     | 7
| Inputs          | 2
| Outputs         | 1
| Clock           | CLK          (rising_edge)
| Clock enable    | ENABLE       (positive)
| Reset           | RESET        (positive)
| Reset type      | asynchronous
| Reset State     | z0
| Power Up State  | z2
| Recovery State  | z1
| 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
-----
z0   | 10
z1   | 01
z2   | 00
--22--
```

CE - DS 2 FSM

## Mealy-FSM: Implemented Equations

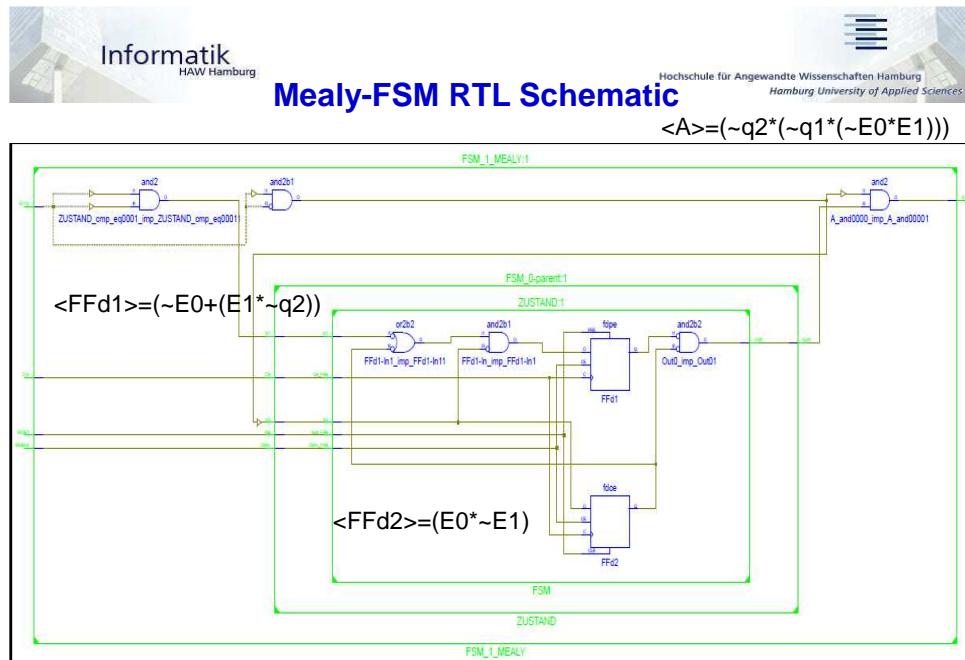
$A \leq (E(1) \text{ AND NOT } E(0) \text{ AND NOT } \text{ZUSTAND\_FSM\_FFd2.LFBK} \text{ AND}$   
 $\text{NOT } \text{ZUSTAND\_FSM\_FFd1.LFBK});$

```
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));
```

```
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



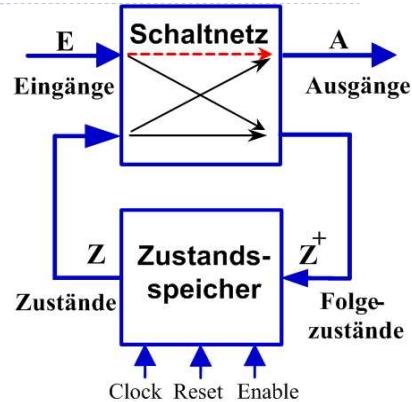
24

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.



## Kombiniertes ÜSN und ASN

```

UE_A_SN: process(E, ZUSTAND)-- Folgezustands- 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 SEQUENZ;

```

## 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

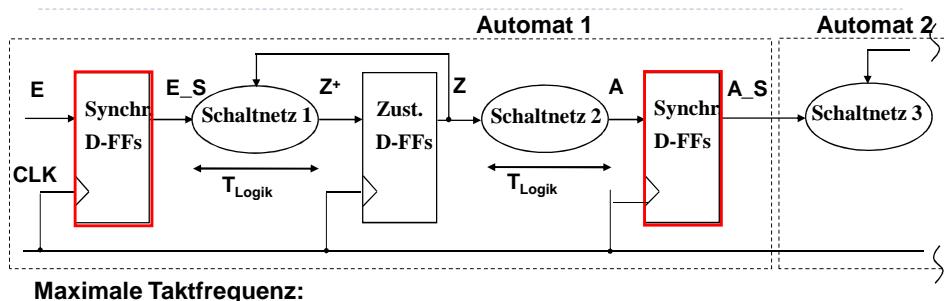


- Bei gekoppelten Automaten ergibt sich die Laufzeit durch den kombinatorischen Pfad als Summe der Laufzeiten durch das Ausgangsschaltnetz des ersten Automaten und der durch das Übergangsschaltnetz des zweiten Automaten!

27

CE - DS 2 FSM

## Ein- und Ausgangssignal synchronisation



Maximale Taktfrequenz:

$T_{PD}$ : D-Flipflop Verzögerung ( $CLK \rightarrow$  Ausgang Q)

$T_{Logik}$ : Signallaufzeit auf dem längsten kombinatorischen Pfad incl. Verdrahtungspfade.

$T_S$ : Einzuhaltende Setup-Zeit der Flipflop-Dateneingänge

28

CE - DS 2 FSM



## 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 ZUSTAEND 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

CE - DS 2 FSM



## Huffman-Normalform mit synchronisierten Eingängen

```

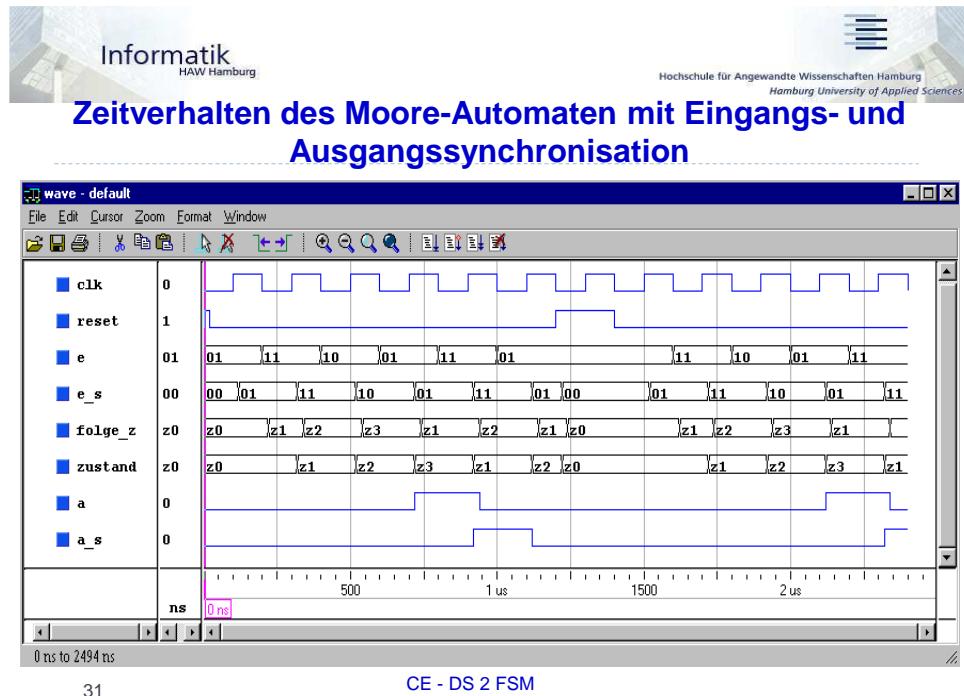
Z_SPEICHER: process(CLK, RESET)      -- Zustandsaktualisierung
  begin
    if RESET = '1' then
      ZUSTAND <= Z0 after 5 ns;
    elsif CLK = '1' and CLK'event then
      ZUSTAND <= FOLGE_Z after 5 ns;
    end if;
  end process Z_SPEICHER;

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

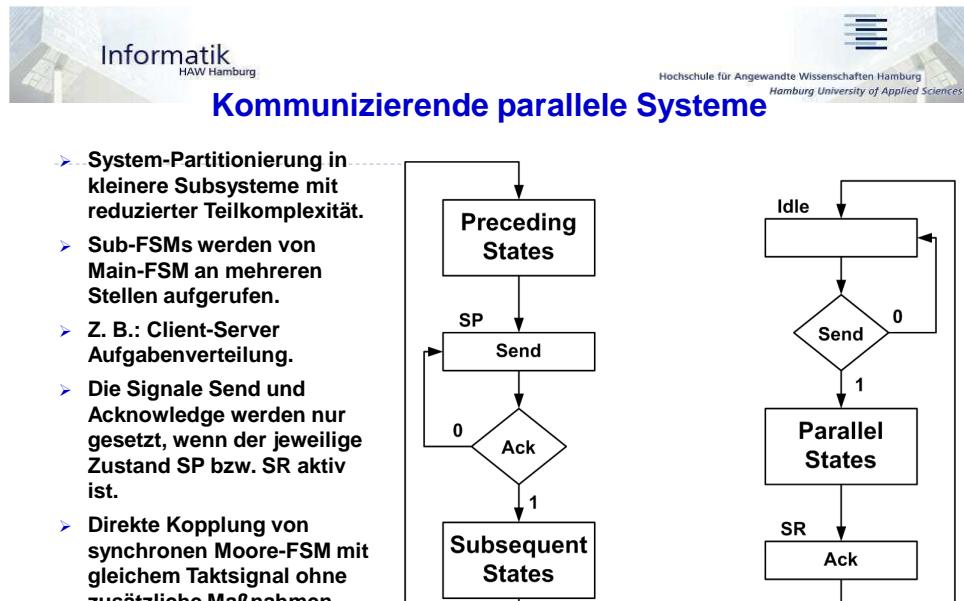
```

30

CE - DS 2 FSM



31

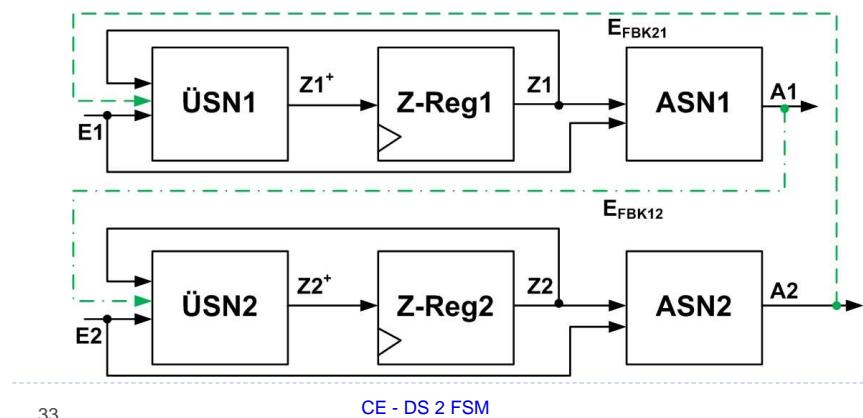


32

CE - DS 2 FSM

## Gekoppelte Mealy-FSM

- Wenn bei der Kopplung von Mealy-Automaten ein Ausgang des 2. Automaten auf den Eingang des 1. Automaten direkt zurück gekoppelt wird, so entsteht eine kombinatorische Schleife → die Ausgänge können schwingen!

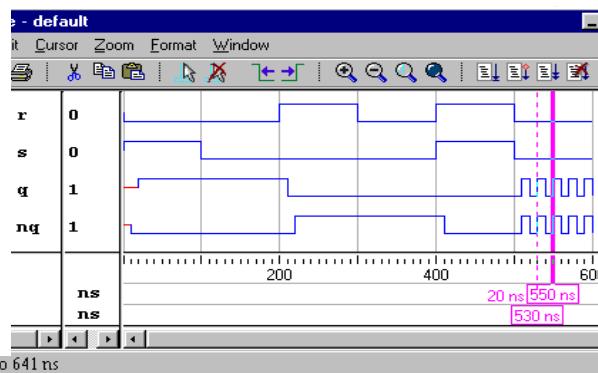
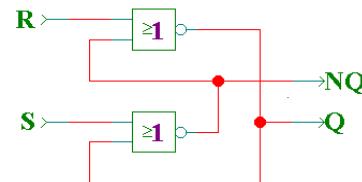


33

CE - DS 2 FSM

## Einfache kombinatorische Schleife

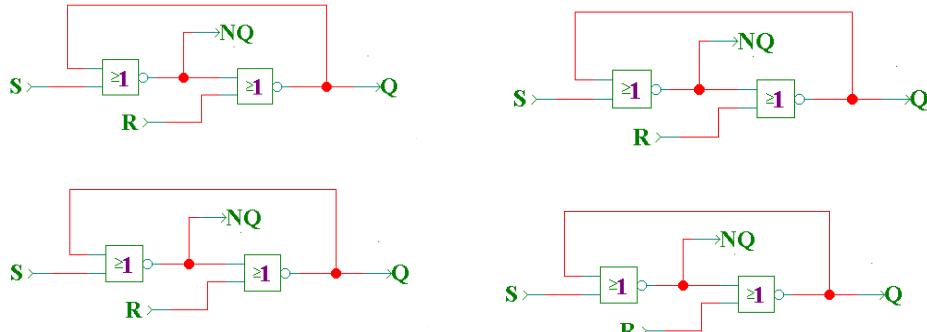
- Im Übergang vom irregulären in den Halten-Zustand treten bei einem NOR-RS-Latch Schwingungen auf, die eine Periode mit  $2 \cdot \text{Gatterlaufzeit}$  haben.
- Irregulär Reset=Setzen=1:  $Q=NQ=0$ ,  $R=S=1$  dominiert
- Halten  $R=S=0$  u.  $Q=NQ=0$ :  
 $\Rightarrow Q=NQ=1$   
 $\Rightarrow$  Rückführung dominiert  
 $\Rightarrow Q=NQ=0$



34

## Analyse der Zustandsübergänge im Basis RS-Latch

NOR-Gatter: Ein beliebiger '1' Eingang erzwingt eine '0' am Ausgang  
Alle Eingänge '0' bewirken eine '1' am Ausgang



35

CE - DS 2 FSM

## 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

## Einzellschrittaddition: Parallel-Seriell-Umsetzer + FSM

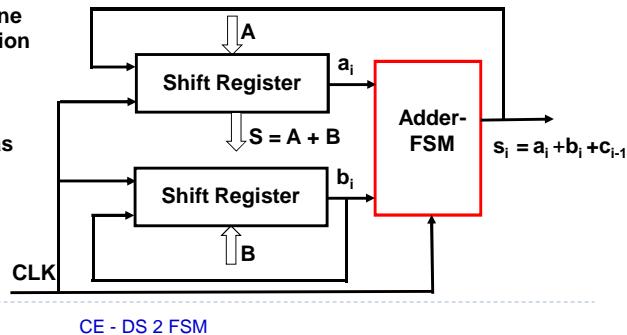
- Der Addierer besteht aus zwei Schieberegistern, in denen die Operandenbits  $a_i$  und  $b_i$  takt synchron nach rechts geschoben werden. In der FSM erfolgt eine 1-Bit Addition der jeweils beiden niederwertigen Operanden  $a_i$ ,  $b_i$  und des Carry-Bits  $c_{i-1}$  der vorangegangenen Addition. Das Summationsbit  $s_i$  wird im Ergebnis-Schieberegister von links nach rechts geschoben.

**Vorteile:**

Mit einem 1-Bit Volladdierer kann eine platzsparende Addition auch für große Bitbreiten erfolgen. Die Addierer-FSM speichert implizit das Carry-Bit der 1-Bit-Additionen.

**Nachteil:**

Eine n-Bit Addition erfordert n-Takte.



37

CE - DS 2 FSM

## Entwurf eines Mealy-Automaten

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:

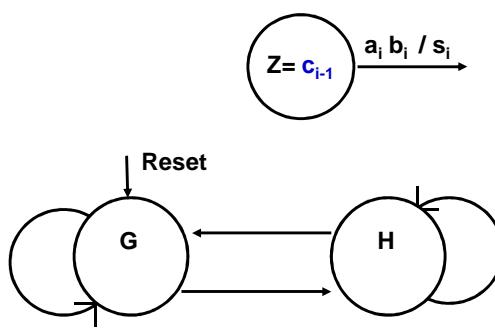
**G:** Carry-In = '0'

**H:** Carry-In = '1'

Zustandsfolge- und Ausgangstabelle:

$Z =$	$a_i \ b_i$	$Z^+ =$	$s_i$
G	0 0		
G	0 1		
G	1 0		
G	1 1		
H	0 0		
H	0 1		
H	1 0		
H	1 1		

CE - DS 2 FSM



38



Informatik  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

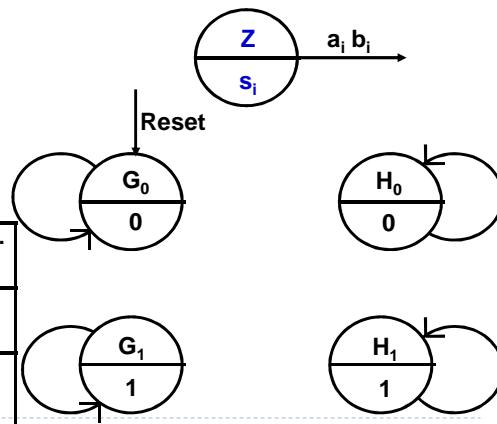
## Entwurf eines Moore-Automaten

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 Z	Folgezustand $Z^+$				Ausg. $s_i$
	$a_i b_i = 00$	$01$	$10$	$11$	
$G_0$					
$G_1$					
$H_0$					
$H_1$					



39

CE - DS 2 FSM

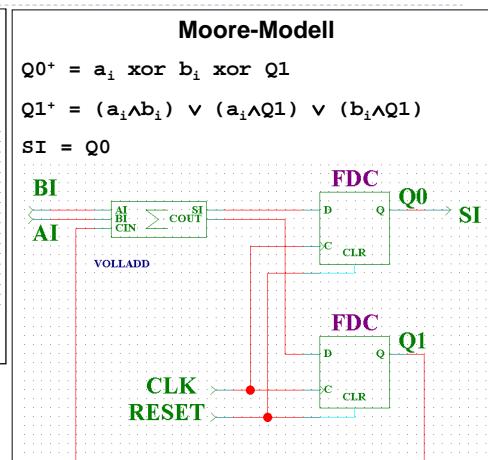
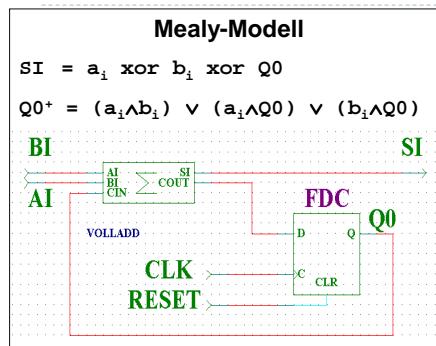


Informatik  
HAW Hamburg



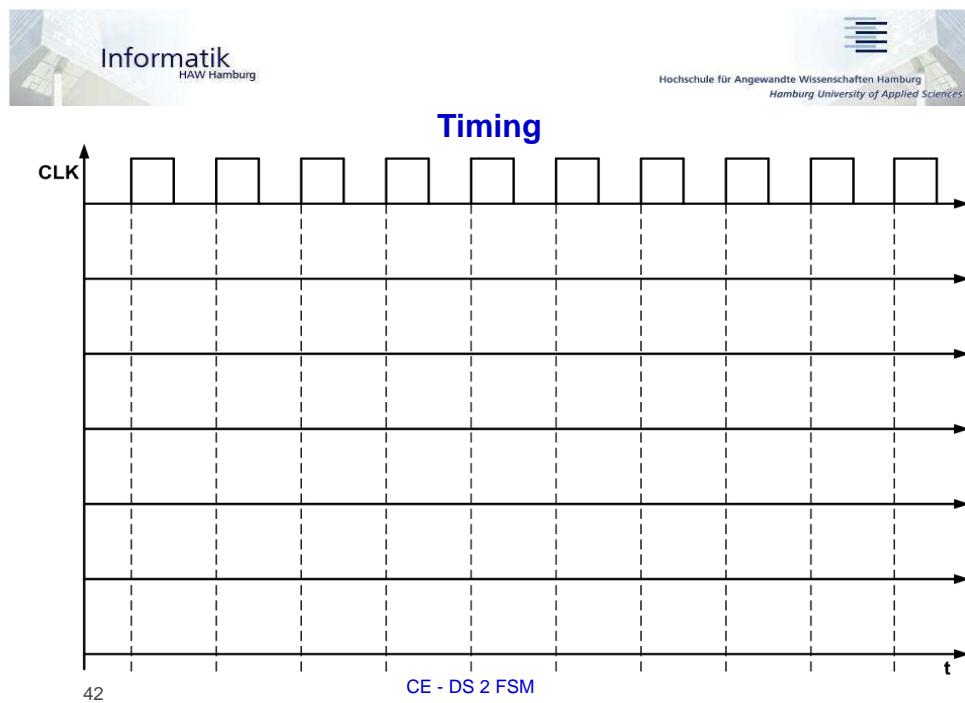
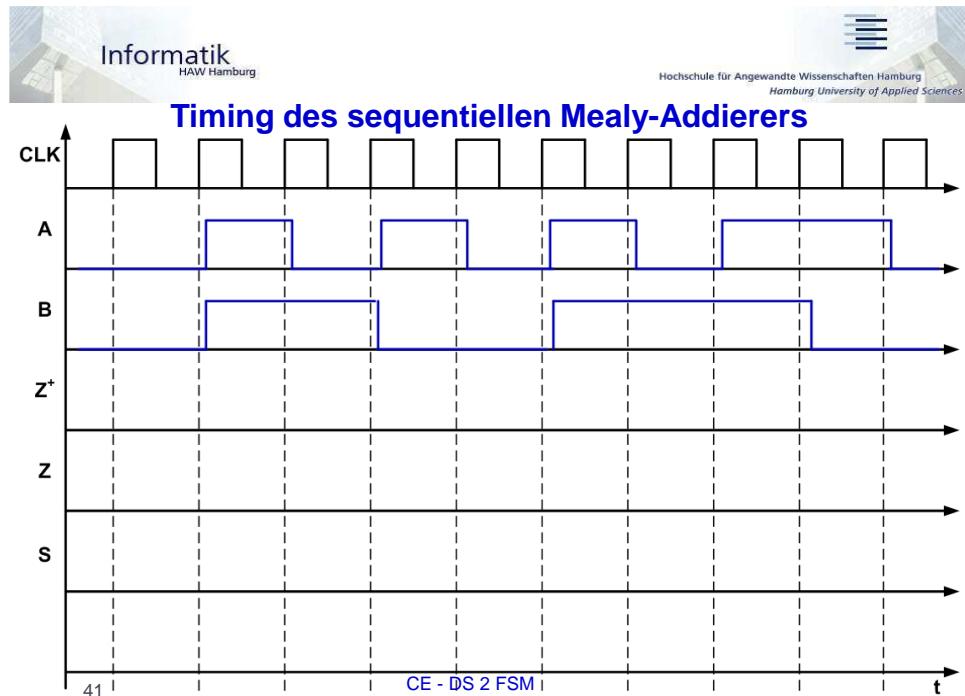
Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

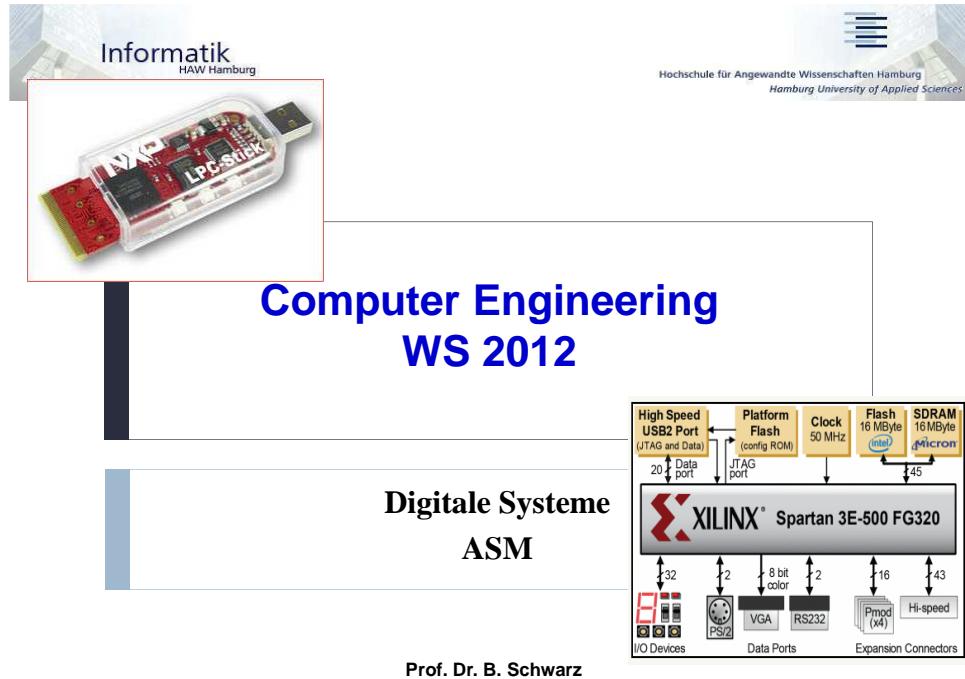
## Syntheseergebnis der sequentiellen Addierer



40

CE - DS 2 FSM





### 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



### 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 Eingangswerte.
- 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  $\ll f_{\text{clk}}$

3

CE - DS 3.1 ASM



### ASMs

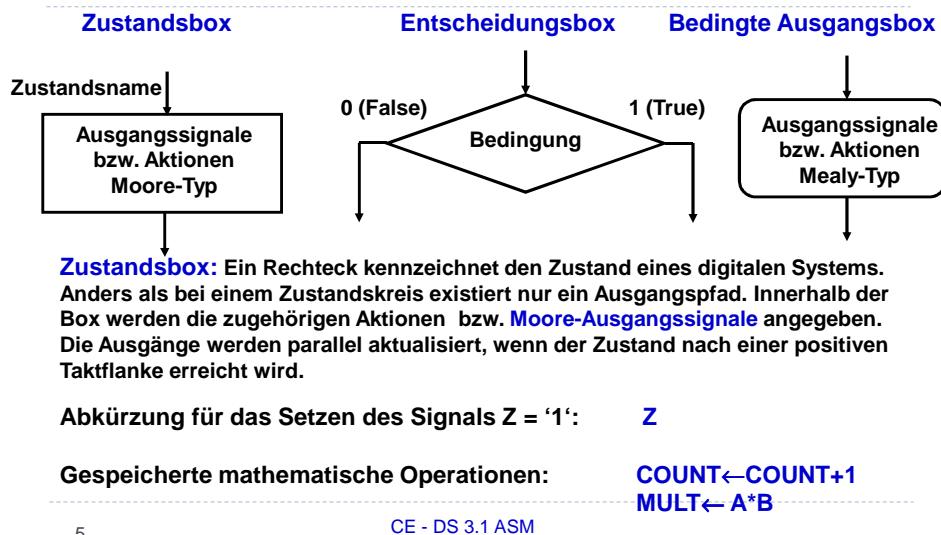
- ASM Diagramme ähneln den Flussdiagrammen, die in den Anfängen der Computerprogrammierung genutzt wurden. Im Gegensatz zu den Flussdiagrammen beschreibt eine ASM ein zeitlich getriggertes Verhalten, da die Übergänge zwischen den Zuständen durch die Flanken eines Taktsignals ausgelöst werden.
- ASM-Diagramme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

4

CE - DS 3.1 ASM



## Grafische ASM-Notation



5

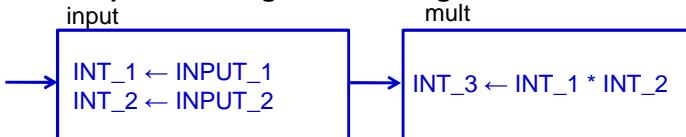
CE - DS 3.1 ASM



## RTL-Notation

- Gemäß der RTL-Notation (RTN) bedeutet in diesem Zusammenhang die Zuweisung mit dem Pfeil  $\leftarrow$ , dass von einer FSM ein hier nicht dargestelltes Clock-Enable-Signal gesetzt wird, mit dem die Übernahme des Ergebnisses der Rechten-Seite in ein Register bei der nächsten positiven Taktflanke initiiert wird!

- Gespeicherte Signalzuweisungen:



Note that the content of the Moore state boxes describes the values which are available in the subsequent state:  
The action which is described in a Moore state box goes effective after the next pos. clock edge.

6

CE - DS 3.1 ASM

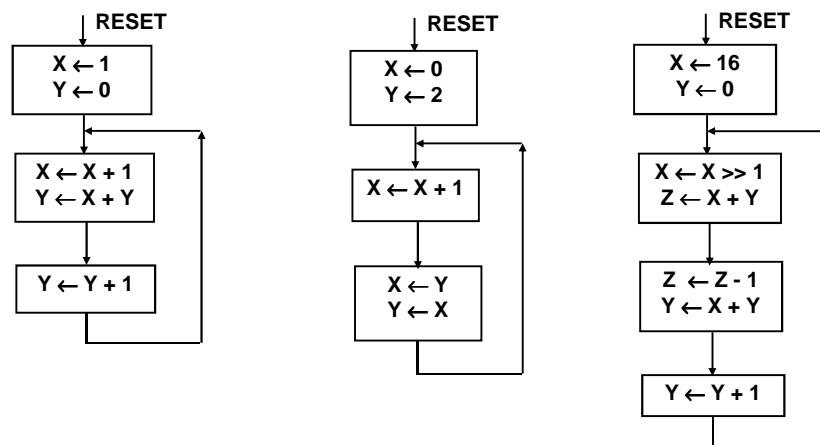
## Bedingte Verzweigungen – Mealy Ausgänge

- **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

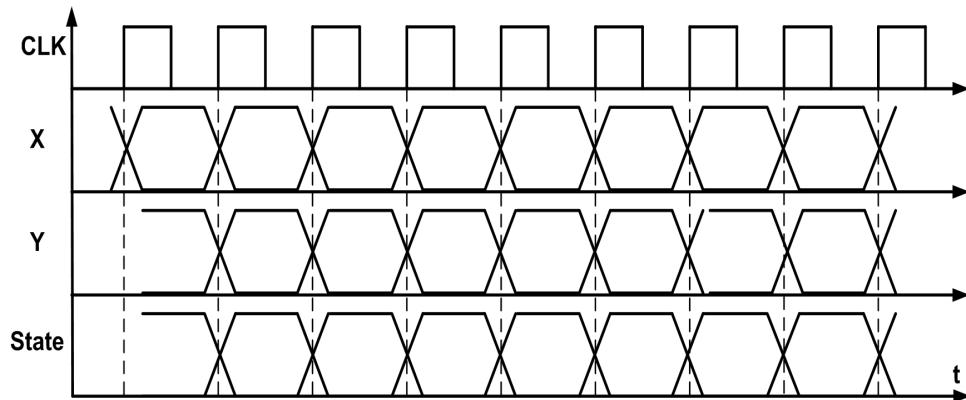


8

CE - DS 3.1 ASM



### Timing Diagramm zur ASM-Chart



9

CE - DS 3.1 ASM



### Behavioural description of an algorithm with an ASM-chart

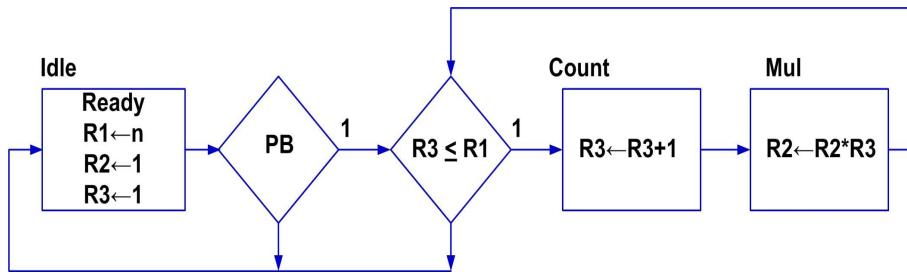
- According to the following C-code of a factorial  $n!$  calculation an ASM-chart has to be developed.
  - With a push button signal **PB = 1** the algorithm will be started otherwise it is idling. A **READY** signal is asserted if the algorithm is idling and **READY** will be zero if the algorithm is running.
- The signal assignment operations should be scheduled to different states.

```
R1 = n;          /* input */
R2 = ?;          /* result */
R3 = ?;          /* counter */
while ( R3 < R1)
{
    R3 = R3 + 1; /* order is important*/
    R2 = R2 * R3;
}
```

10

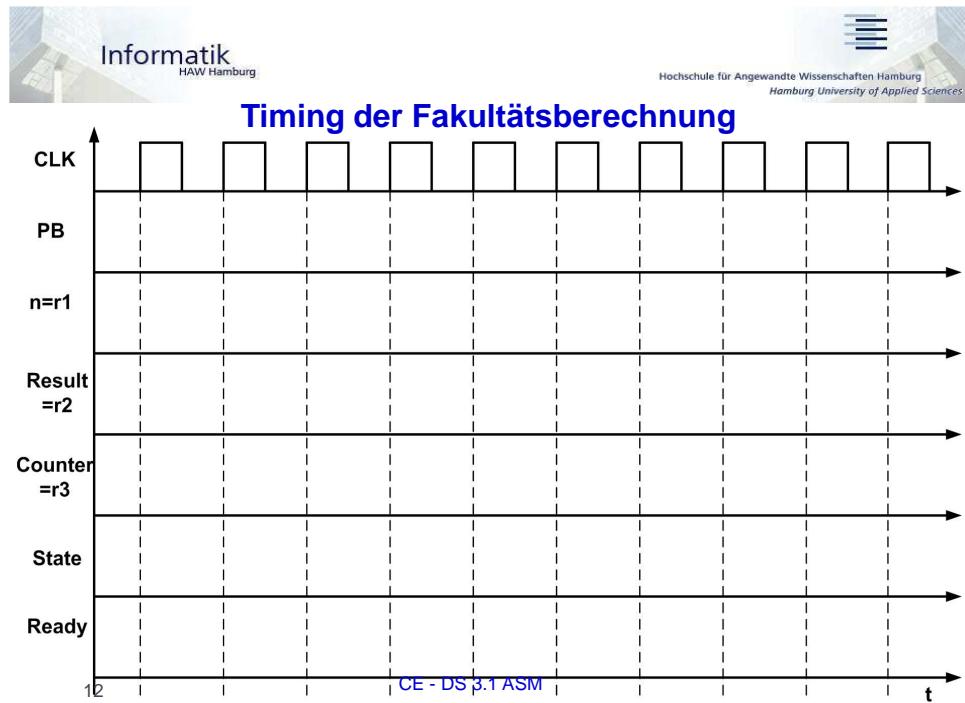
CE - DS 3.1 ASM

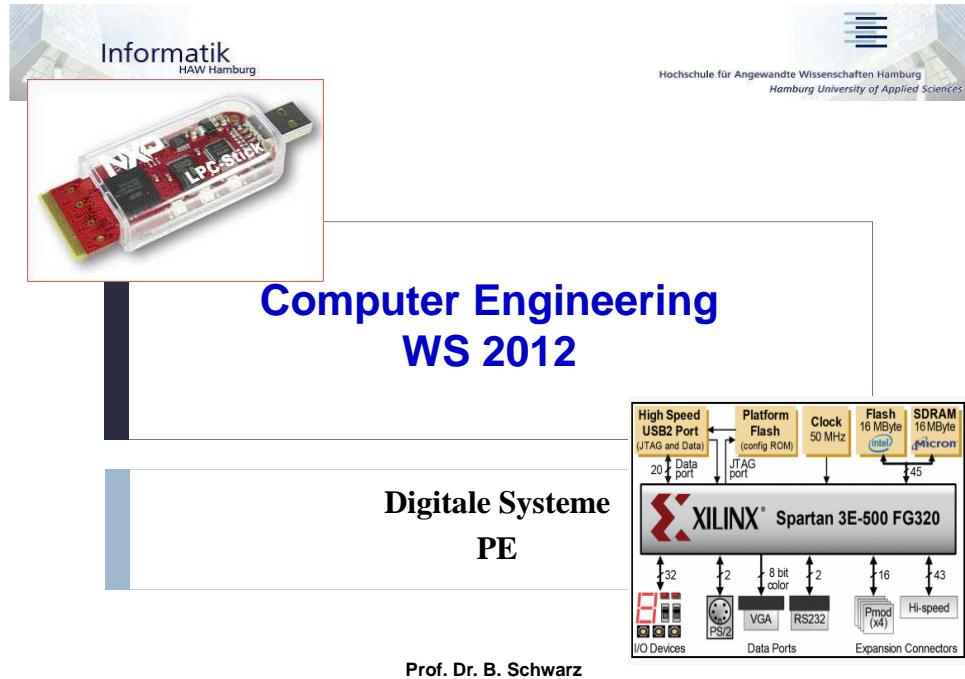
### ASM Chart der Fakultätsberechnung



11

CE - DS 3.1 ASM





### 3. Architektursynthese ASM-Diagramme

**Prozessorelement für eine S-Kurvenapproximation  
Gemeinsame Nutzung von Arithmetik-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**



### 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

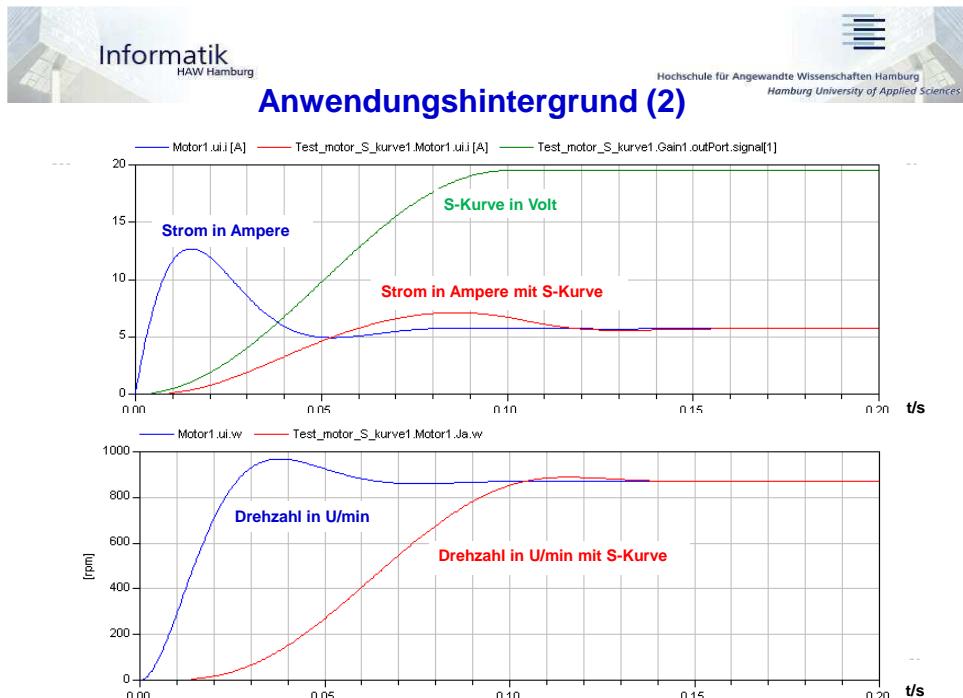


## Anwendungshintergrund (1)

- 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





## 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  $-1 \leq x_Q < 1$

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

9

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.

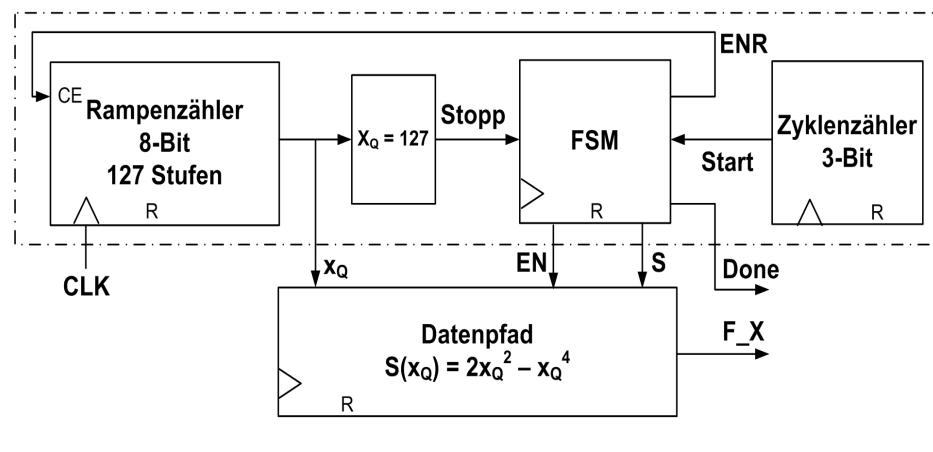
- Mit jedem wrap-around des Zyklenzählers erhält der Rampenzähler die Freigabe für das Inkrement einer Rampenstufe.

- 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

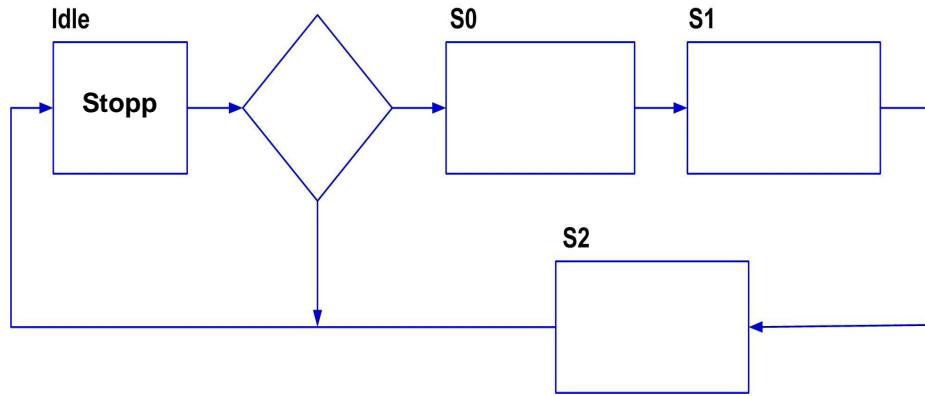
- 8-Bit Rampenzähler mit 128 Stufen:  $x_{Q\max} = 127$
- Anlaufintervall 100ms
- Effektive Taktfrequenz des Rampenzählers:  
 $f_{Rclk} = 1/(100ms/128) = 1280 \text{ Hz}$
- 3-Bit Zyklenzähler:  
 8 Zählzustände müssen das Intervall  $1/f_{Rclk}$  abdecken  
 $f_{clk} = 8 * 1280 \text{ Hz} = 10,24 \text{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

12

CE - DS 2

## ASM des PEs zur Approximation der S-Kurve

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



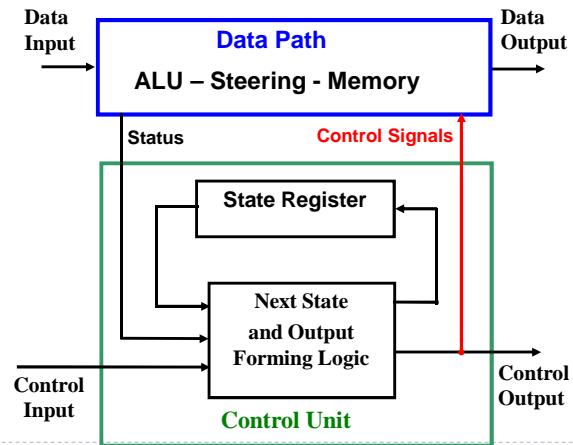
13

CE - DS 2

## 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 <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	Reg
t <sub>1</sub>					
t <sub>2</sub>					
y					

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

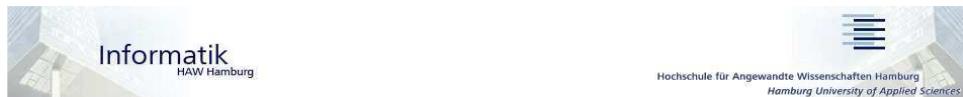
CE - DS 2



## 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:

$x_{\text{Bin}} = b_B \# b_{B-1} \dots b_1 b_0$  mit  $b \in (0,1)$  und  
einem Vorzeichenbit  $b_B = 1$  für  $x_{\text{Dez}} < 0$ .  
Dieses QB-Format repräsentiert Zahlen im Intervall  $-1 \leq x_{\text{Dez}} < 1$



## 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
```



## 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<sub>1</sub>-Multiplikanden und einem QB<sub>2</sub>-Multiplikator ergibt ein QB<sub>3</sub>-Ergebnis.  
Darin stehen B<sub>3</sub>=B<sub>2</sub>+B<sub>1</sub> Bits rechts vom impliziten Binärpunkt und der gesamte Vektor enthält B<sub>3</sub>+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

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



## 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 Vektdimensionierung 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

27

CE - DS 2



### 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 Überschwingen überlauffrei aufnehmen.
- [5] Kapitel 9.2 IIR-Filter

28

CE - DS 2



## 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.

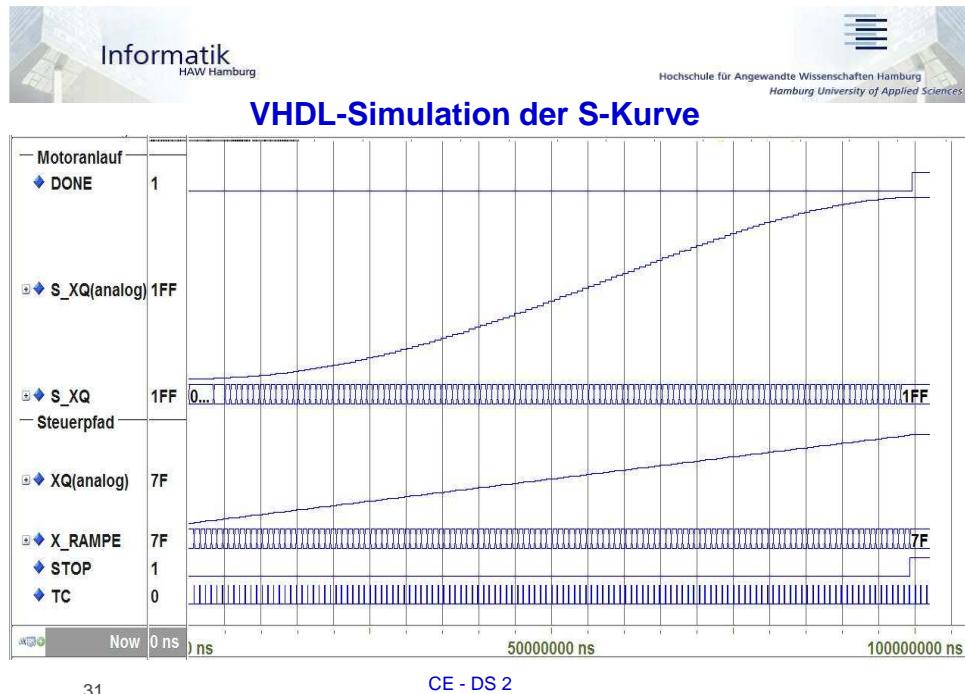
▪ Abgriff des Multiplikationsergebnisses Q18 mit 2 VZ:

RESULT(18 : 9); abschneiden der unteren 9 Bit ohne Rundung.

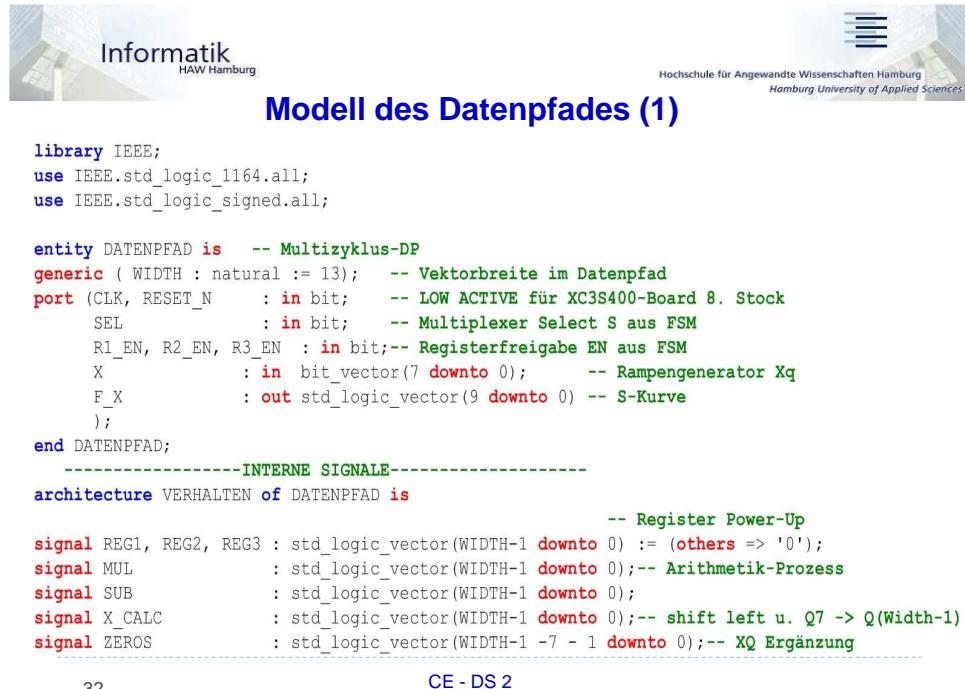
- Erweiterung der Subtraktionsoperanden, sodass Faktor 2 für  $x^2$  durch SHL erreicht wird und Sign-Extension bei  $x^4$ , damit die Vektorlängen auf WIDTH+1 angeglichen werden.

- 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



32



## 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;

```

33

CE - DS 2



## 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;

```

34

CE - DS 2



**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## 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
```

35

CE - DS 2



**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## 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;
```

36

CE - DS 2



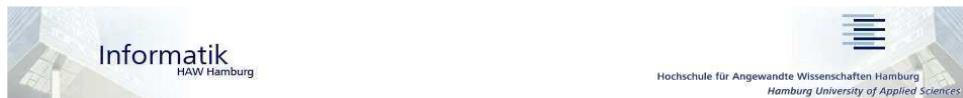
### Modell des Steuerpfades (3)

```

UE_A_SN: process(ZUSTAND, TC, STOP)
begin
    SEL <= '0' after 5 ns; -- Rampe an MUL
    R1_EN <= '0' after 5 ns;-- kein Register Update
    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 =>
        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.



## 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;
-----
```

39

CE - DS 2



## 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  -- Instanzierung
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;
```

40

CE - DS 2

**Informatik**  
HAW Hamburg

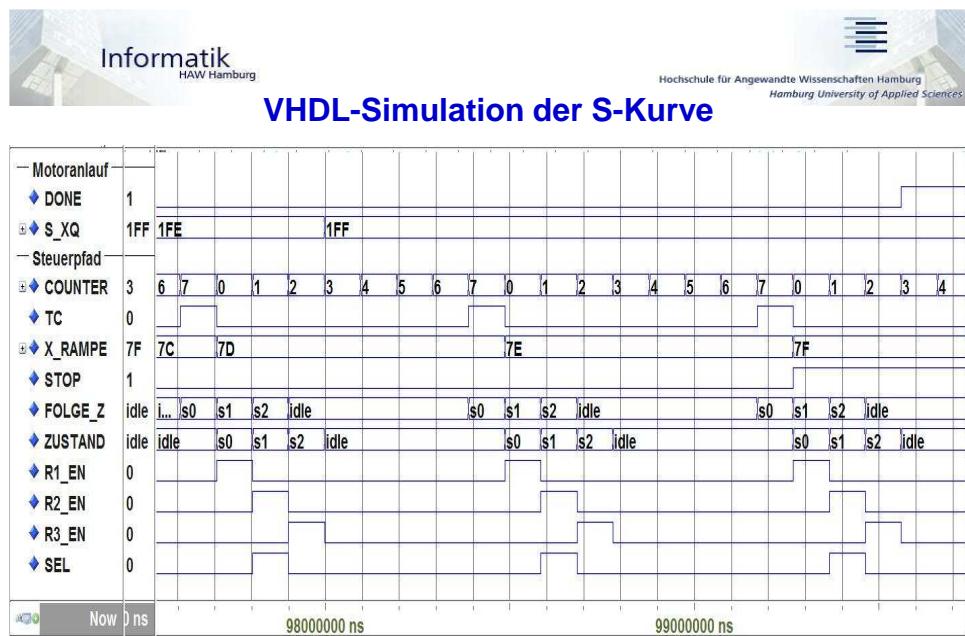
**Hochschule für Angewandte Wissenschaften Hamburg**  
*Hamburg University of Applied Sciences*

## 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;

```




**Informatik**  
 Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

### Auszug aus .do File für die Signalauswahl

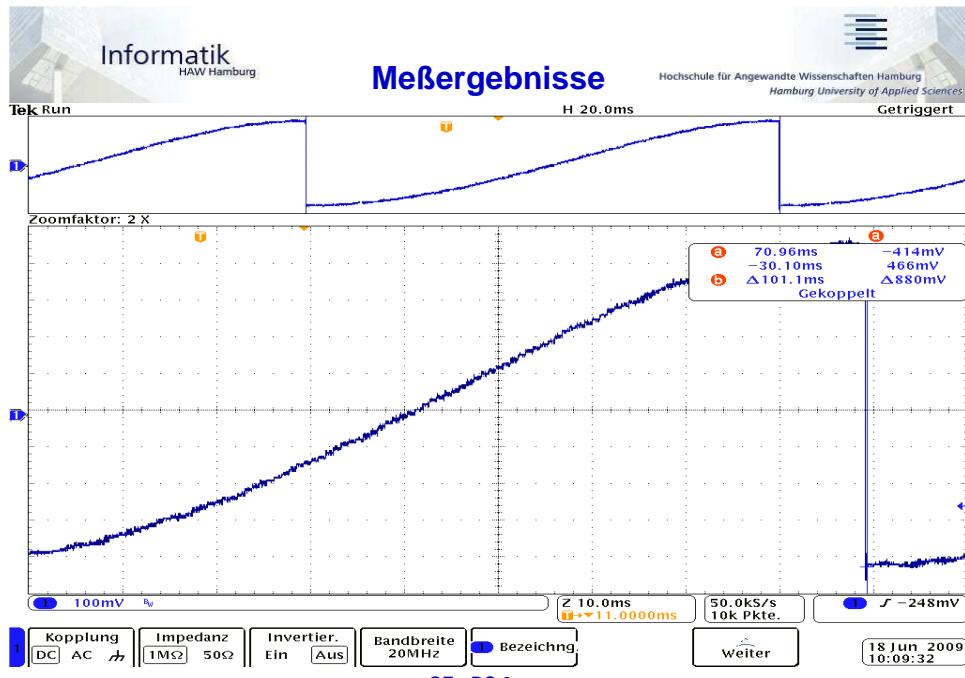
```

# 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 -divider 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

CE - DS 2



**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Längster Laufzeitpfad (1)

<b>Delay</b> (setup path): 10.496 ns (data path - clock path skew + uncertainty)					
<b>Source:</b>	ST_I/X_RAMPE_7 (FF)				
<b>Destination:</b>	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				
<b>Maximum Data Path:</b>	<b>ST_I/X_RAMPE_7 to DP_I/REG1_5</b>				
<u>Location</u>	<u>Delay type</u>	<u>Delay(ns)</u>	<u>Physical Resource</u>		
			Logical Resource(s)		
SLICE_X13Y107.YQ	<b>Tck0</b>	0.511	ST_I/X_RAMPE<6>		
SLICE_X13Y96.G4	net (fanout=3)	0.772	ST_I/X_RAMPE<7>		
SLICE_X13Y96.Y	<b>Tilo</b>	0.612	DP_I/TMP_mux0000<0>		
MULT18X18_X0Y12.B15	net (fanout=12)	2.499	DP_I/TMP_mux0000<12>		
MULT18X18_X0Y12.P17	<b>Tmult</b>	4.331	DP_I/Mmult_RESULT_mult0000		
SLICE_X2Y97.BX	net (fanout=2)	1.458	DP_I/Mmult_RESULT_mult0000		
SLICE_X2Y97.CLK	<b>Tdick</b>	0.313	DP_I/MUL<5>		
			DP_I/REG1<5>		
			DP_I/REG1_5		
Total		10.496ns (5.767ns logic, 4.729ns route) (54.9% logic, 45.1% route)			

45

CE - DS 2



**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Spartan 3E Switching Characteristics (1)

Configurable Logic Block (CLB) Timing

Table 98: CLB (SLICEM) Timing

Symbol	Description	Speed Grade				Units
		-5	-4	Min	Max	
<b>Clock-to-Output Times</b>						
<b>T<sub>CK0</sub></b>	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
<b>Setup Times</b>						
T <sub>AS</sub>	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 <sub>DICK</sub>	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
<b>Hold Times</b>						
T <sub>AH</sub>	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
<b>T<sub>CKDI</sub></b>	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

46

CE - DS 2



## Spartan 3E Switching Characteristics (2)

Propagation Times						
T <sub>ILO</sub>	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

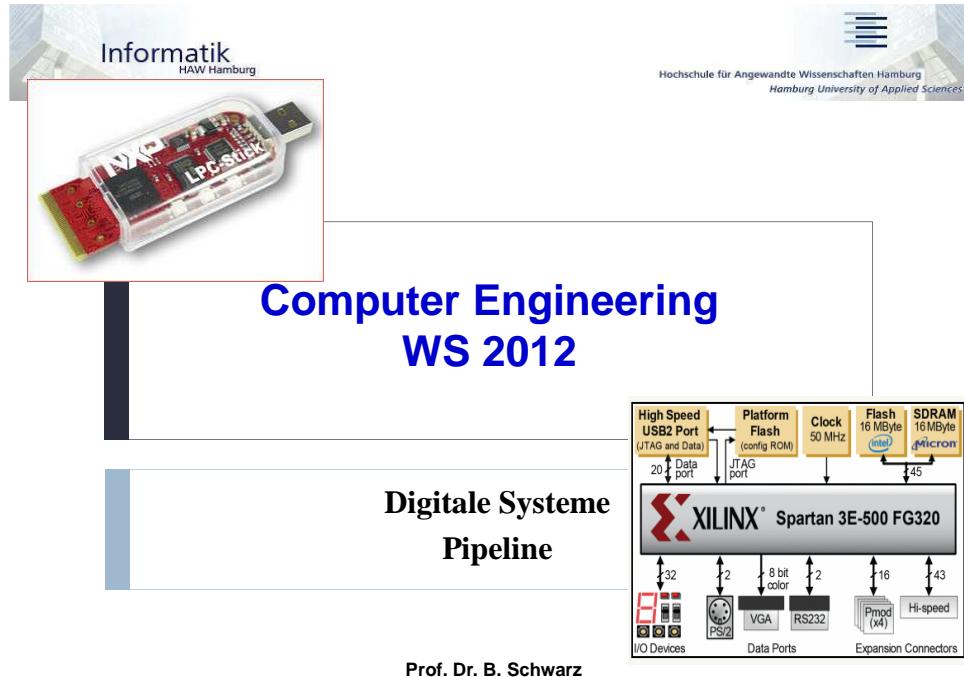
Table 102: 18 x 18 Embedded Multiplier Timing

Symbol	Description	Speed Grade				Units	
		-5		-4			
		Min	Max	Min	Max		
Combinatorial Delay							
T <sub>MULT</sub>	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	



## 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		
<b>Maximum Data Path:</b>	<b>ST_I/ZUSTAND_FSM_FFd2_1 to DP_I/REG1_5</b>		
Location	Delay type	Delay(ns)	Physical Resource Logical Resource(s)
SLICE_X13Y103.YQ	T <sub>cko</sub>	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 <sub>ilo</sub>	0.612	DP_I/TMP_mux0000<0>
MULT18X18_X0Y12.A17	net (fanout=12)	2.499	DP_I/TMP_mux0000<12>1
MULT18X18_X0Y12.P17	T <sub>mult</sub>	4.331	DP_I/TMP_mux0000<12>2
SLICE_X2Y97.BX	net (fanout=2)	1.458	DP_I/Mmult_RESULT_mult0000
SLICE_X2Y97.CLK	T <sub>clock</sub>	0.313	DP_I/Mmult_RESULT_mult0000
<b>Total</b>	<b>10.332ns (5.767ns logic, 4.565ns route) (55.8% logic, 44.2% route)</b>		

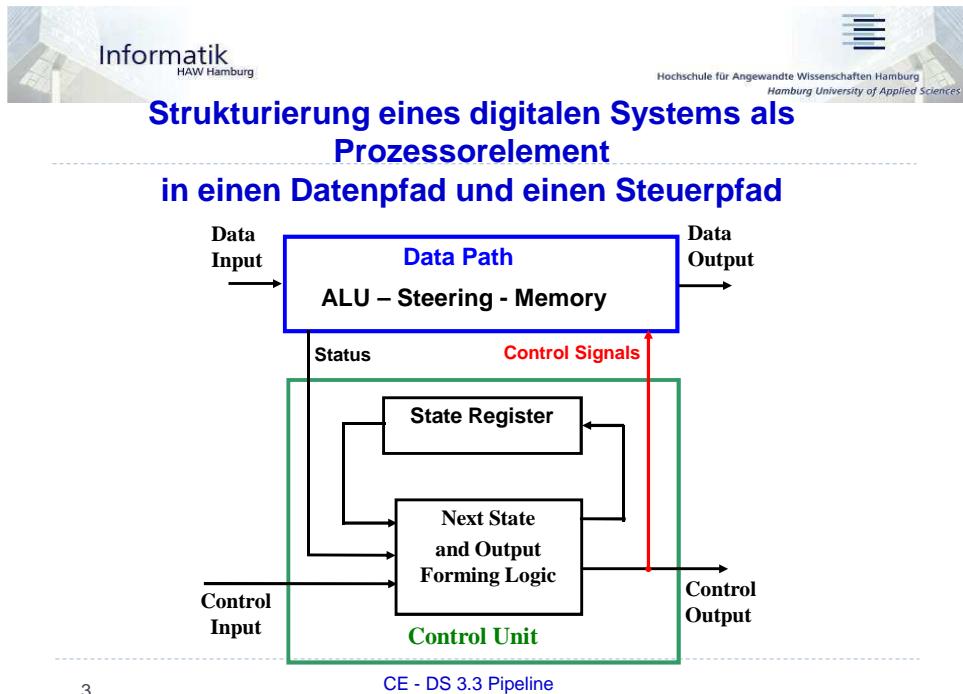


### 3. Architekturentwurf ASM-Diagramme

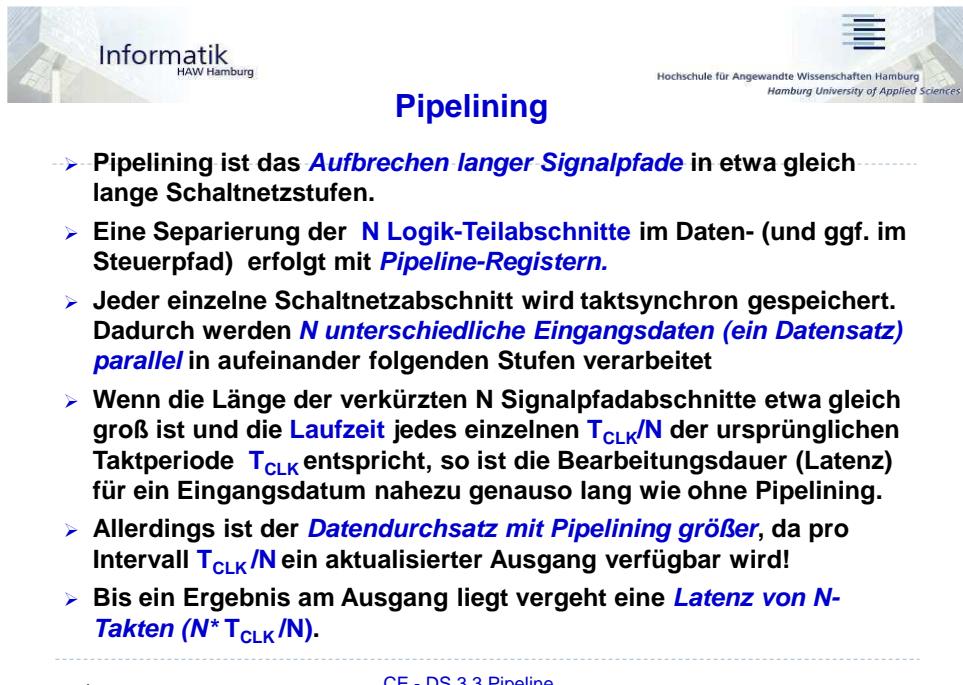
Prozessorelement für eine S-Kurvenapproximation  
Gemeinsame Nutzung von Hardware Funktionseinheiten  
(Ressource Sharing)  
Gemeinsame Register- / Speicher-Nutzung  
(Register Sharing)  
VHDL-Modell des PE mit Fixed-Point Arithmetik im Q-Format

### Datenpfad in Pipelinestruktur

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



3



4



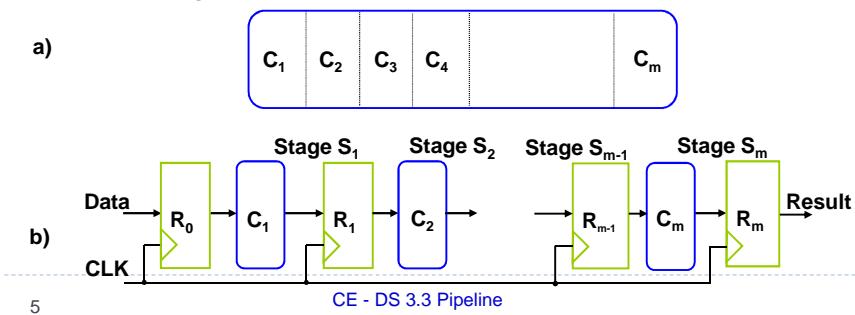
**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Pipelining of Data Paths

- 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.



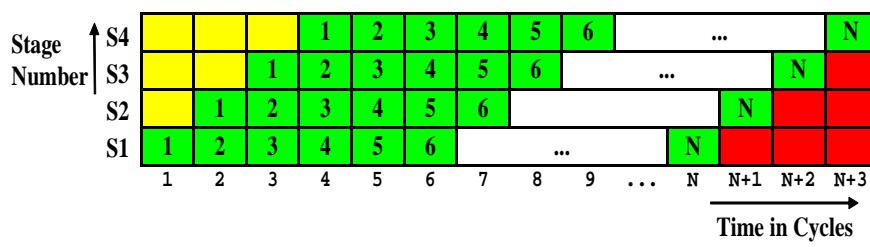

**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## 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)

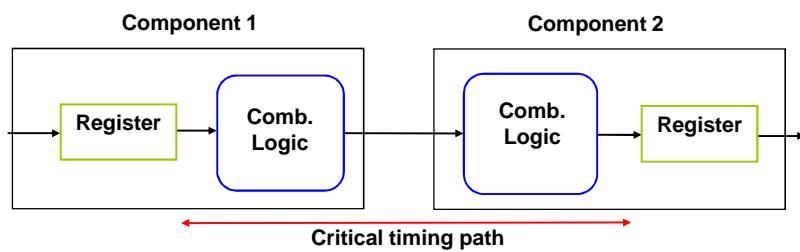
- Looking at one time instant, i.e. time unit 4, we can see that all stages of the pipeline are busy working on different data items. A data item is completely processed when it leaves the last stage. The first data item leaves the pipeline when all stages are filled. After that has accomplished **each 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.

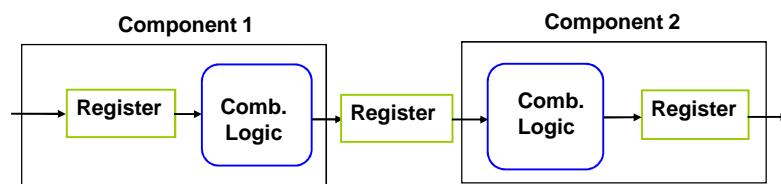


8

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

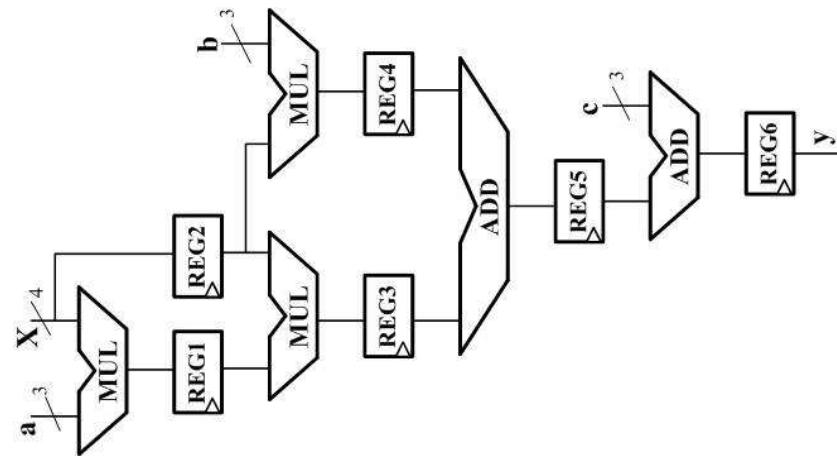
CE - DS 3.3 Pipeline

## Pipelining-Struktur für die S-Kurvenapproximation

10

CE - DS 3.3 Pipeline

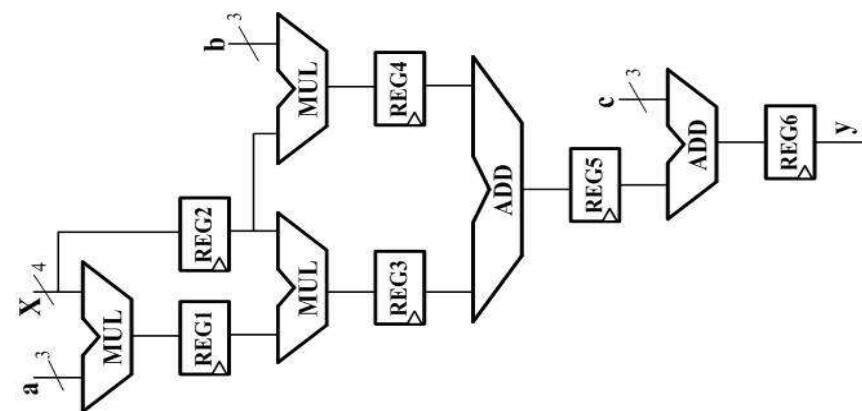
## Polynomauswertung



11

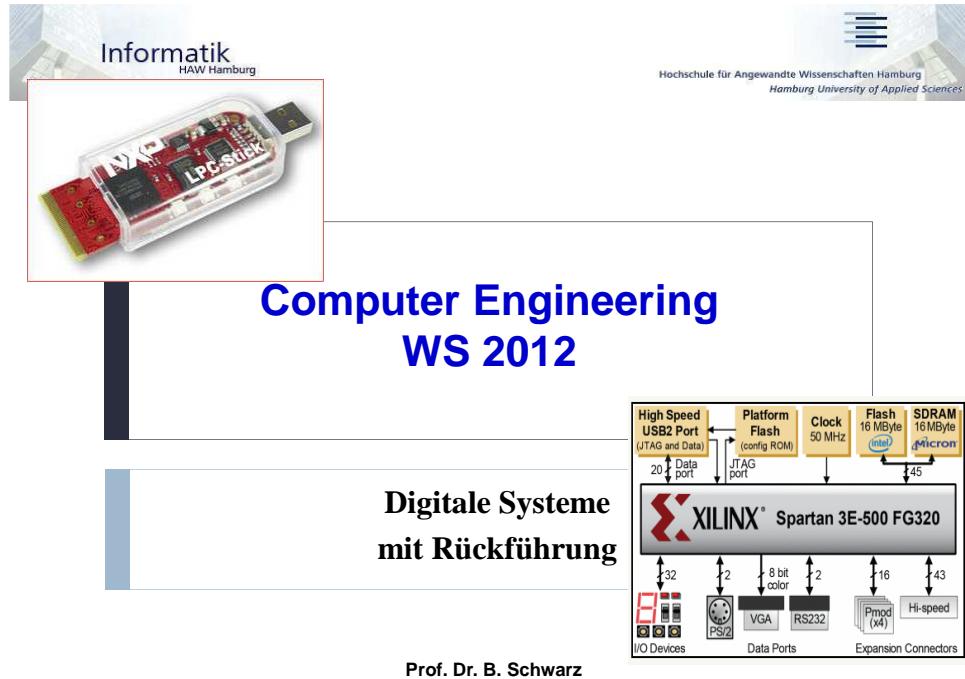
CE - DS 3.3 Pipeline

## Polynomauswertung mit zeitveränderlichen Parametern



12

CE - DS 3.3 Pipeline

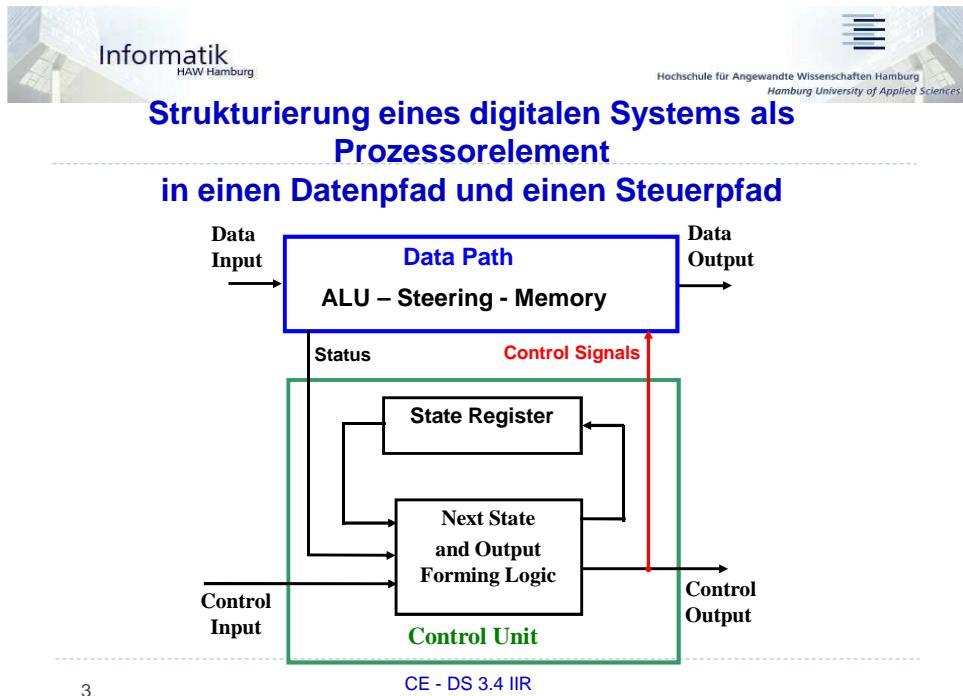


### 3. Architektursynthese ASM-Diagramme

**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**

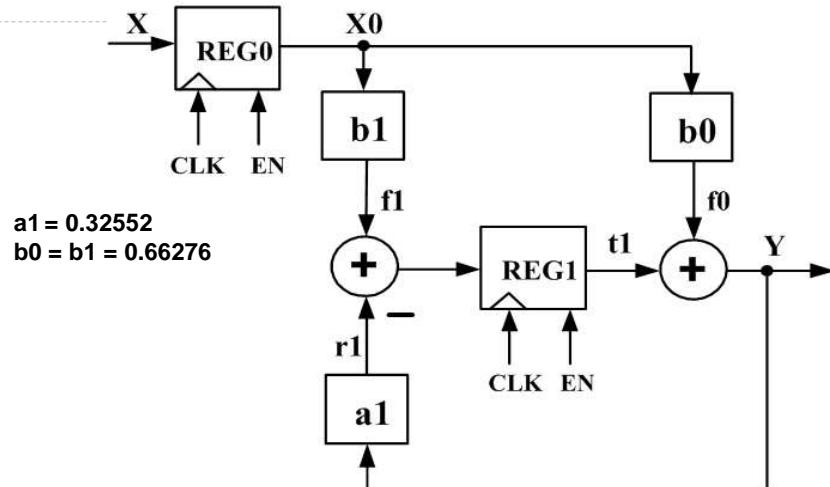


3



4

### Modifizierte Form des IIR-Filters 1. Ordnung

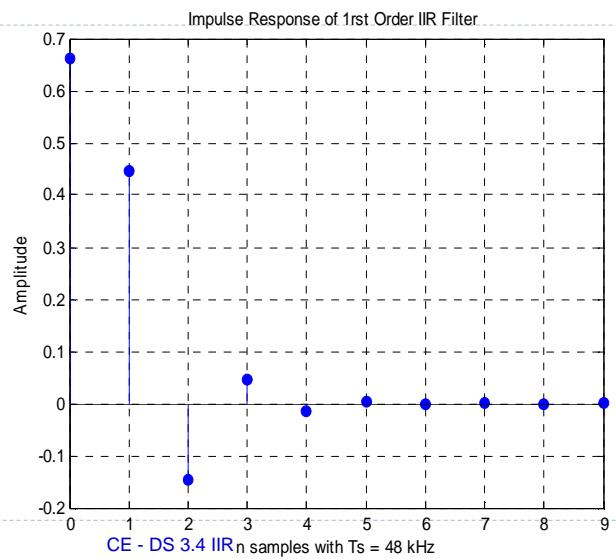


5

CE - DS 3.4 IIR

### Impulsantwort des IIR-Filters 1. Ordnung

- Einheitsimpuls-Anregung mit  $X(n=0) = 1$  und  $X(n>0) = 0$ .

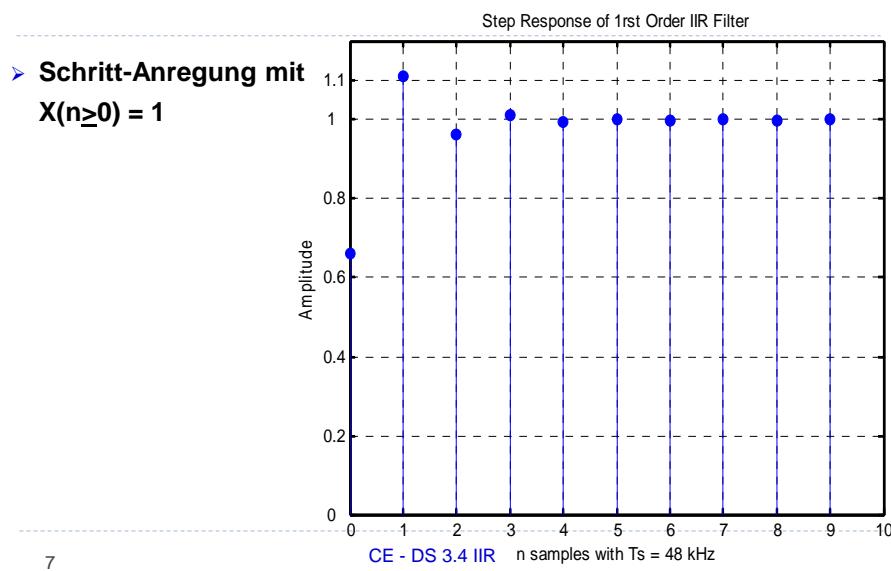


6

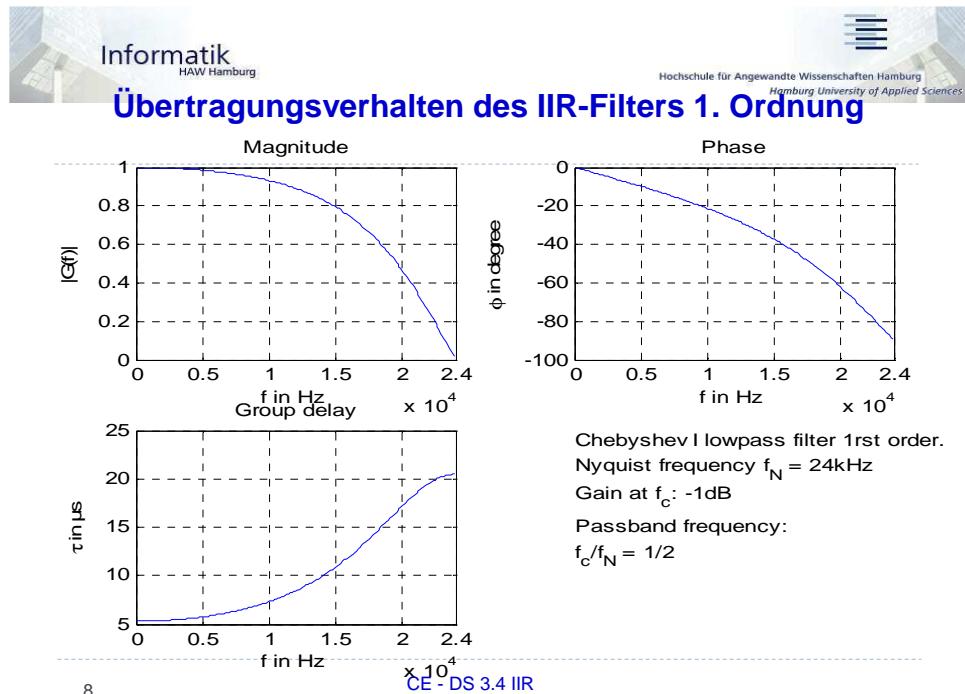
**Informatik**  
HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Sprungantwort des IIR-Filters 1. Ordnung



7



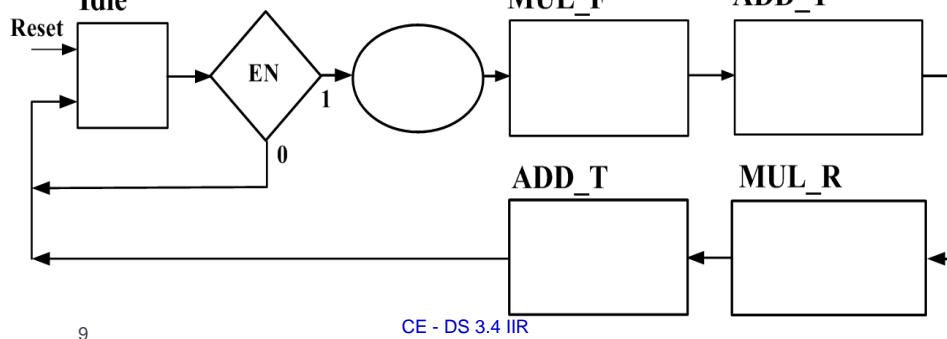
8



ASM zum IIR-Filter 1. Ordnung

- Datenpfad mit je einem Addierer und einem Multiplizierer.
  - Eingangssignal X soll zu Beginn des Zyklus gespeichert werden.
  - Systemrückkopplung realisiert durch Ergebnisübernahme im Folgezyklus:  
$$Y(n) = t1 + b0 * X0(n); \quad t1^+ = b1 * X0(n) - a1 * Y(n); \quad t1(n) = t1^+(n-1)$$

LII MUL E ADD V



9

CE - DS 3.4 IIF



## Register-Sharing

Signal	Idle	Mul_f	Add_y	Mul_r	Add_t	Reg(1)	Reg(2)
X0							
f0							
Y							
f1							
r1							
t1							

- Registerzuordnung der Operanden einer Operation so, dass Operanden-Multiplexer gespart werden.
  - Registerzuordnung der Operationsergebnisse so, dass Ergebnis-Multiplexer gespart werden.

CE - DS 3.4 IIR



## Datenpfad des IIR-Filters 1. Ordnung

11

CE - DS 3.4 IIR



## Q-Format Vektordimensionierung

- Voller Signalhub für  $Y > 1$  gespeichert im Register R0: 11 Bit  
sign, g. msb ... lsb Q9 ergänzt mit Guardbit
- Vorzeichenerweiterung für X in R0 (X0)
- Koeffizienten im Q9-Format, da  $a_1 \text{ u. } b_1 < 1$
- Register R1 u. R2 mit 10 Bit im Q9-Format für alle anderen Summen und Produkte: sign, msb ... lsb
- Produkt:  $f_0 = f_1 = X_0 * b_1 < 1$  wird für R1 auf Q9 reduziert:  
Produkt: sign,g,g. msb ... lsb mit Bereich [20 : 0]; 21 Bit  
Mul: Produkt[18:9] 10 Bit
- Produkt :  $r_1 = a_1 * Y < 1$  wird für R2 auf Q9 reduziert
- Summen:  $Y = f_0 + t_1 > 1$  sign, g. msb ... lsb in R0  
 $t_1 = f_0 - r_1 < 1$  reduziert auf Q9 in R2

12

CE - DS 3.4 IIR



## Modell des IIR-Prozessorelementes (1)

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
-- Difference equation 1st order
entity FILTER is
    generic(DELAY:    time := 5 ns;
            WIDTH: natural := 10);
    port(
        CLK, RESET, EN: in std_logic;
        X: in std_logic_vector(WIDTH-1 downto 0);    -- Q9      s.msb ... lsb
        Y: out std_logic_vector(WIDTH downto 0);     -- s,Q9   s,g.msb ... lsb
        READY: out std_logic
    );
end FILTER;
architecture MULTI_CYCLE of FILTER is
constant A1: std_logic_vector(WIDTH-1 downto 0) := "1101011001";
-- constant A1: std_logic_vector(WIDTH-1 downto 0) := "0010100111";
constant B1: std_logic_vector(WIDTH-1 downto 0) := "0101010011";-- = B0

```

13

CE - DS 3.4 IIR



## Modell des IIR-Prozessorelementes (2)

```

-- Registers source of operands, target of results
signal REG0      :std_logic_vector(WIDTH downto 0)  := (others => '0'); -- power up
signal REG1, REG2:std_logic_vector(WIDTH-1 downto 0) := (others => '0'); -- power up

signal MUL  : std_logic_vector(WIDTH-1 downto 0); -- s.msb ... lsb
signal ADD  : std_logic_vector(WIDTH downto 0);   -- s,g.msb ... lsb
-- control signals --
signal EN0, EN1, EN2: std_logic;
signal SEL0, SEL1, SEL2: std_logic;
--- FSM ---
type STATES is (IDLE, MUL_F, ADD_Y, MUL_R, ADD_T);
attribute ENUM_ENCODING: string;           -- minimum bit change
attribute ENUM_ENCODING of STATES: type is "000 001 011 010 110";
signal Z, Z_PLUS: STATES := IDLE;          -- Power up state

```

14

CE - DS 3.4 IIR



### Modell des IIR-Prozessorelementes (3)

```

begin
  ----- Data Path -----
  -----REGISTER-----
REGISTER0:process(CLK)
variable DRO: std_logic_vector(WIDTH downto 0);-- REG0-Plus s,Q9
begin
  if (CLK = '1' and CLK'event) then
    if (RESET = '1') then
      REG0 <= (others => '0') after DELAY;
    elsif (EN0 = '1') then
      if SEL0 = '1' then          -- input Mux0
        DRO := ADD;             -- Y s,Q9
      else
        DRO := X(X'left) & X;   -- sign extension: X0 s,Q9
      end if;
      REG0 <= DRO after DELAY;
    end if;
  end if;
end process REGISTER0;

REGISTER1:process(CLK)
begin
  if (CLK = '1' and CLK'event) then
    if (RESET = '1') then
      REG1 <= (others => '0') after DELAY;
    elsif (EN1 = '1') then
      REG1 <= MUL after DELAY;           -- f0=f1 Q9
    end if;
  end if;
end process REGISTER1;           CE - DS 3.4 IIR
15

```

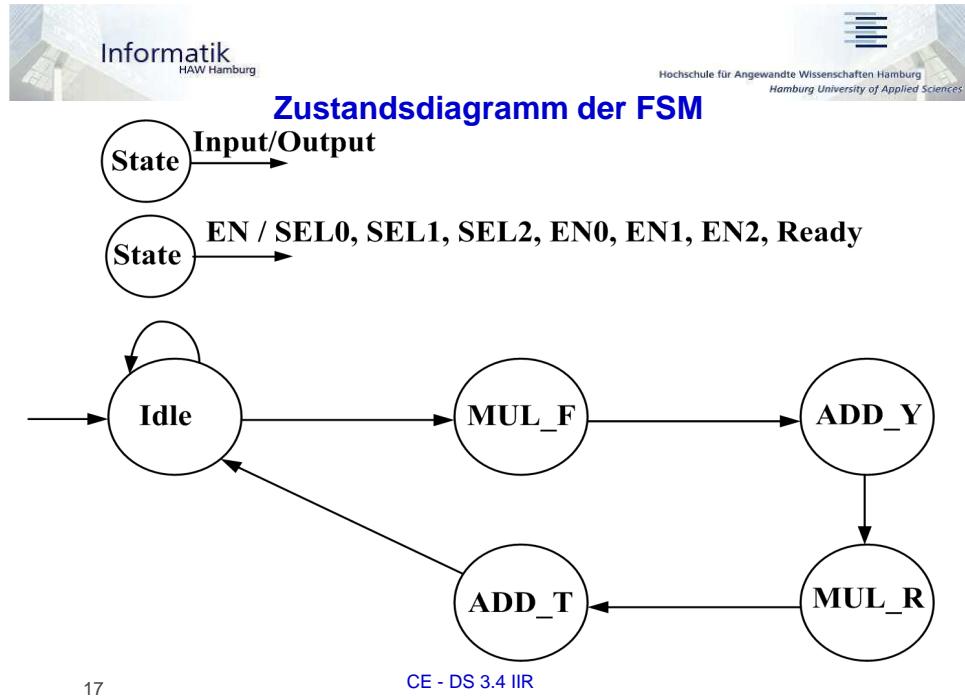


### Modell des IIR-Prozessorelementes (4)

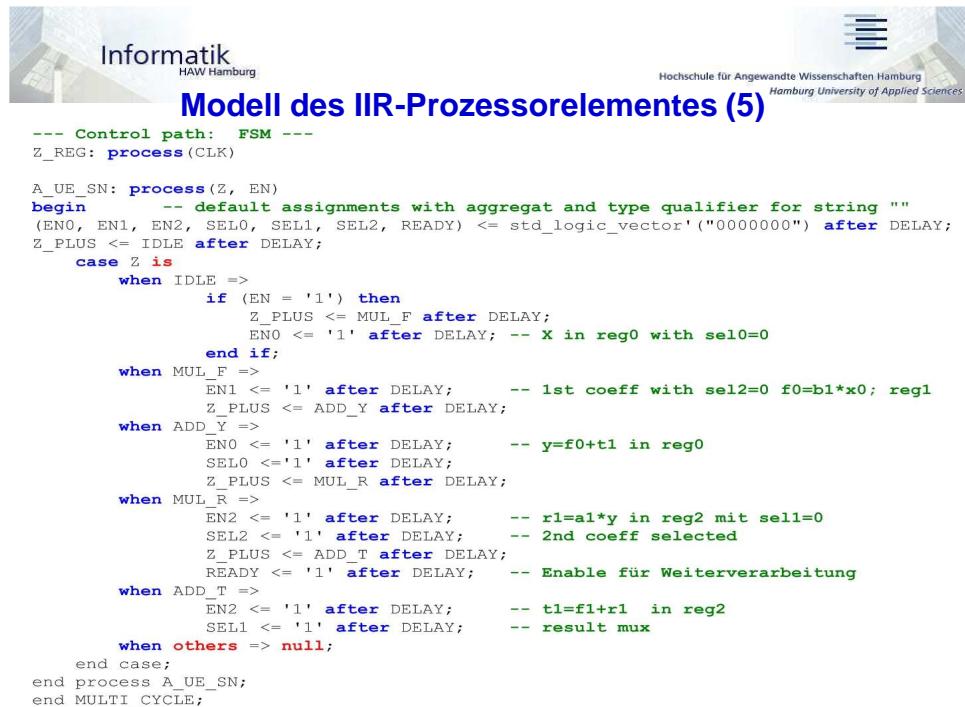
```

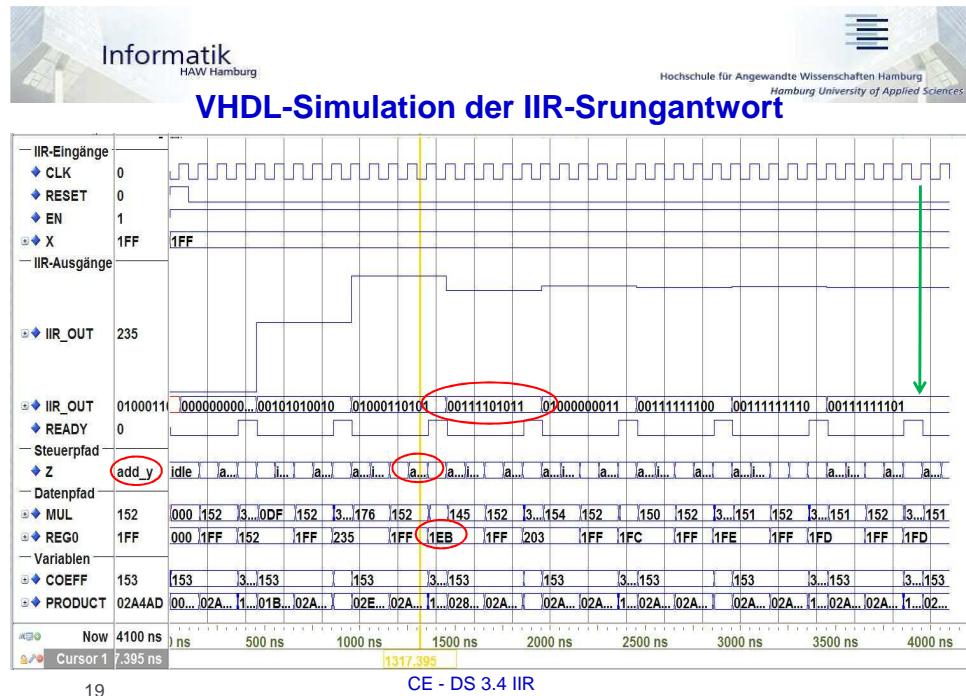
REGISTER2:process(CLK)
variable DR2: std_logic_vector(WIDTH-1 downto 0); -- REG2-Plus Q9
begin
  if (CLK = '1' and CLK'event) then
    if (RESET = '1') then
      REG2 <= (others => '0') after DELAY;
    elsif (EN2 = '1') then
      if SEL1 = '1' then          -- result Mux1
        DR2 := ADD(WIDTH-1 downto 0); -- t1 s,Q9 -> Q9
      else
        DR2 := MUL;              -- r1 Q9
      end if;
      REG2 <= DR2 after DELAY;
    end if;
  end if;
end process REGISTER2;
MULTIPLIER: process(SEL2, REG0)
variable COEFFICIENT: std_logic_vector(WIDTH-1 downto 0); -- Q9
variable PRODUCT: std_logic_vector(2*WIDTH downto 0);       -- s,g,Q18 [20:0]
begin
  if (SEL2 = '0') then    -- Operand Mux2
    COEFFICIENT := B1;   -- f0 = b1*x0
  else
    COEFFICIENT := A1;   -- r1 = a1*y
  end if;
  PRODUCT := COEFFICIENT * REG0; -- Q9 * Q10 = s,g,Q18 [20:0]
  MUL <= PRODUCT(2*WIDTH-2 downto WIDTH-1) after DELAY; -- [18:9]
end process MULTIPLIER;
ADDER: ADD <= (REG1(REG1'left) & REG1) + (REG2(REG2'left) & REG2) after DELAY; -----
Y <= REG0;           CE - DS 3.4 IIR
16

```



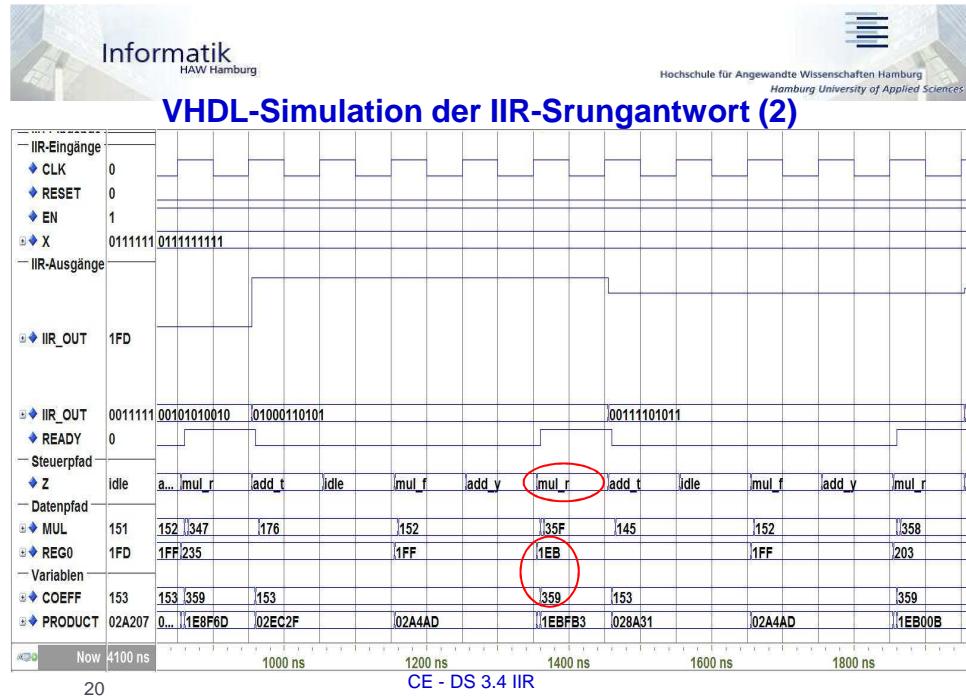
17



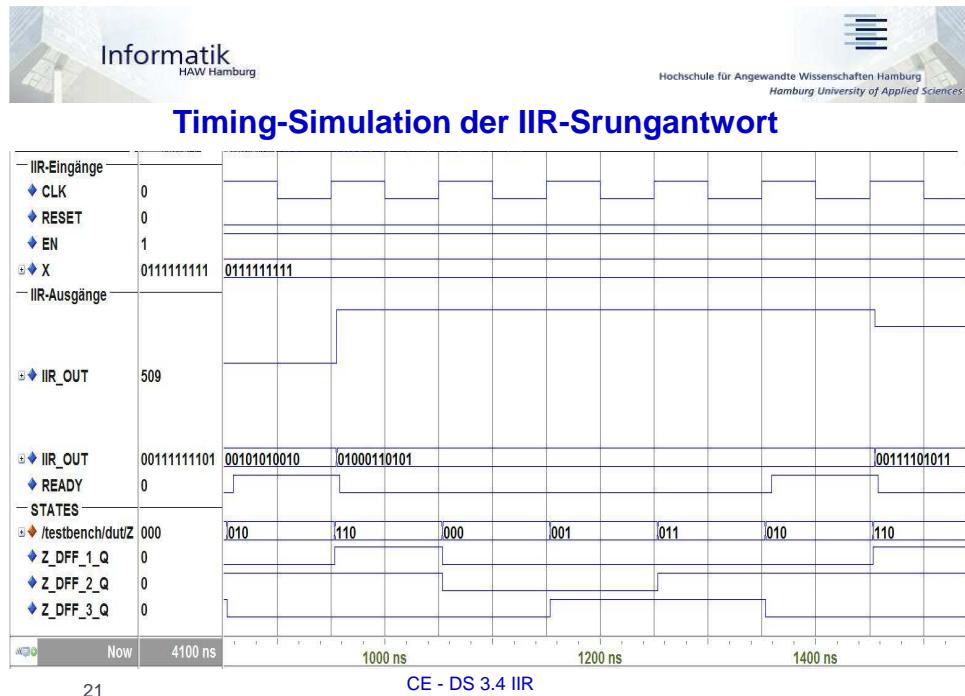


19

CE - DS 3.4 IIR

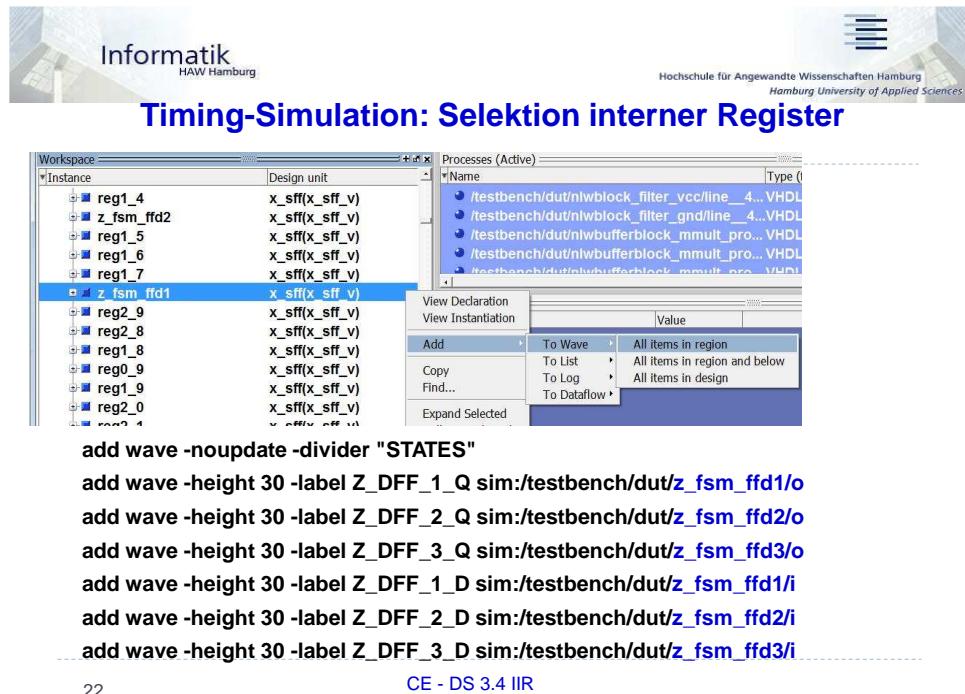


20



21

CE - DS 3.4 IIR



22

CE - DS 3.4 IIR



## Numerische Lösung einer Dgl. 2. Ordnung

- Viel zitiertes Beispiel für eine Pipelining-Anwendung (DeMicheli; Teich) :

$$y'' + 3xy' + 3y = 0$$

- Lösung mit der Rechteckintegration:

$$y' = dy/dx = (y(n) - y(n-1))/\Delta x; \quad \Delta x = x(n) - x(n-1)$$

- Ordnungsreduktion durch Substitution:

$$u = dy/dx = y' \Rightarrow \quad u' = du/dx = -3xu - 3y$$

- Übergang auf Differenzen mit  $\Delta u = u(n) - u(n-1)$

- Differenzengleichungen für  $u'$  und  $y'$ :

$$u(n) = u(n-1) - x(n-1)u(n-1)*3\Delta x - y(n-1)*3\Delta x$$

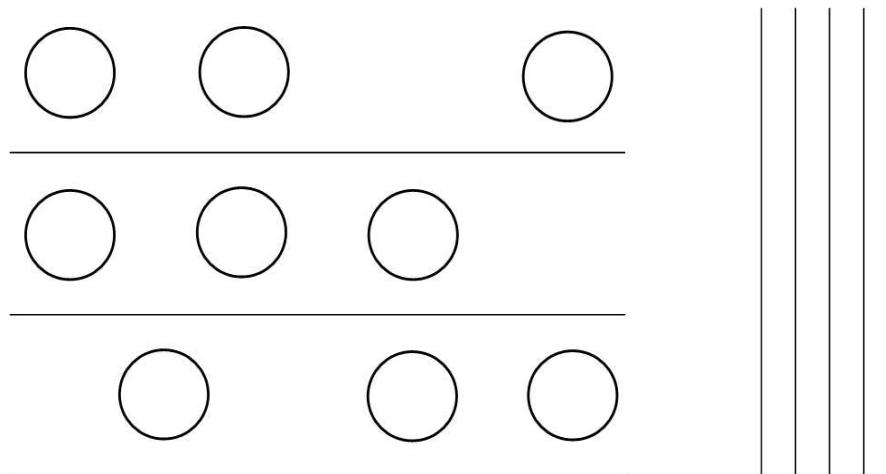
$$y(n) = y(n-1) + u(n-1) \Delta x$$

23

CE - DS 3.4 IIR



## Sequenzgraph



24

CE - DS 3.4 IIR

## Phasendiagramm zum Pipelinesystem mit Rückkopplung

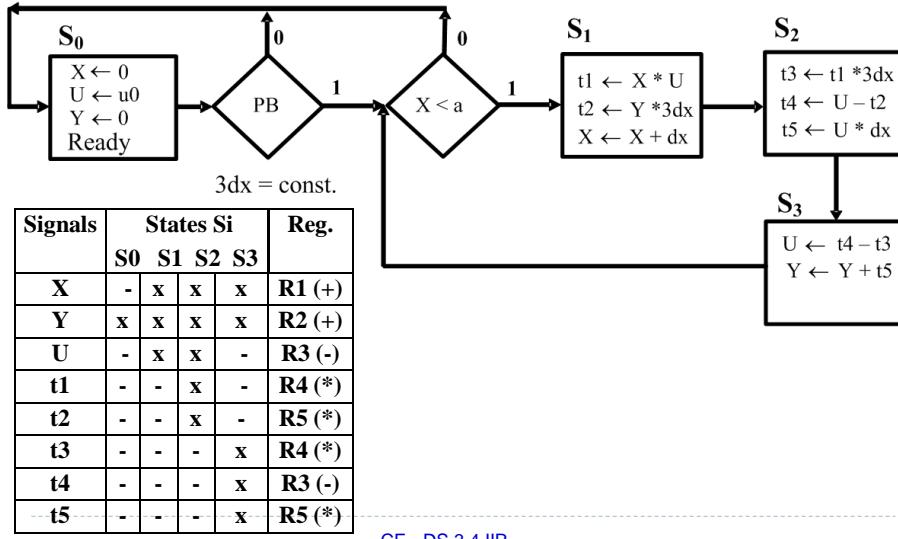
St. 3	$f3(0, 0, 0) = 0, 0, 0$	$f3(0, 0, 0) = 0, 0, 0$	$f3(u_0, y_0, x_0) = u_1, y_1, x_1$			$f3(u_1, y_1, x_1) = u_2, y_2, x_2$
St. 2	$f2(0, 0, 0)$	$f2(u_0, y_0, x_0)$	$f2(0, 0, dx)$		$f2(u_1, y_1, x_1)$	
St. 1	$f1(u_0, y_0, x_0)$	$f1(0, 0, dx)$	$f1(0, 0, dx)$	$f1(u_1, y_1, x_1)$		

- Nach n=3 Takte liegt ein erstes Ergebnis ( $u_1, y_1, x_1$ ) ausgehend von den Anfangswerten ( $u_0, y_0, x_0$ ) aus der 3. Stufe vor (Latenz n = 3).
- Mit dem (n+1)-Takt ist der zweite Zyklus gestartet.
- Die zwischenzeitlichen (Takte n-1, n-2, n+2, ...) Ergebnisse basieren auf Initialisierungen der Stufen 1. u. 2. und liefern keine verwertbaren Beiträge zum Anfangswertproblem mit ( $u_0, y_0, x_0$ ).
- Der Durchsatz ist somit  $f_{clk}/n$ , sodass kein Vorteil durch das Pipelining erreicht wird.
- Ein äquivalenter Multizyklus-DP mit dem Durchsatz  $f_{clk}/n$  kann also mit geringerem Aufwand an HW-Ressourcen realisiert werden.

25

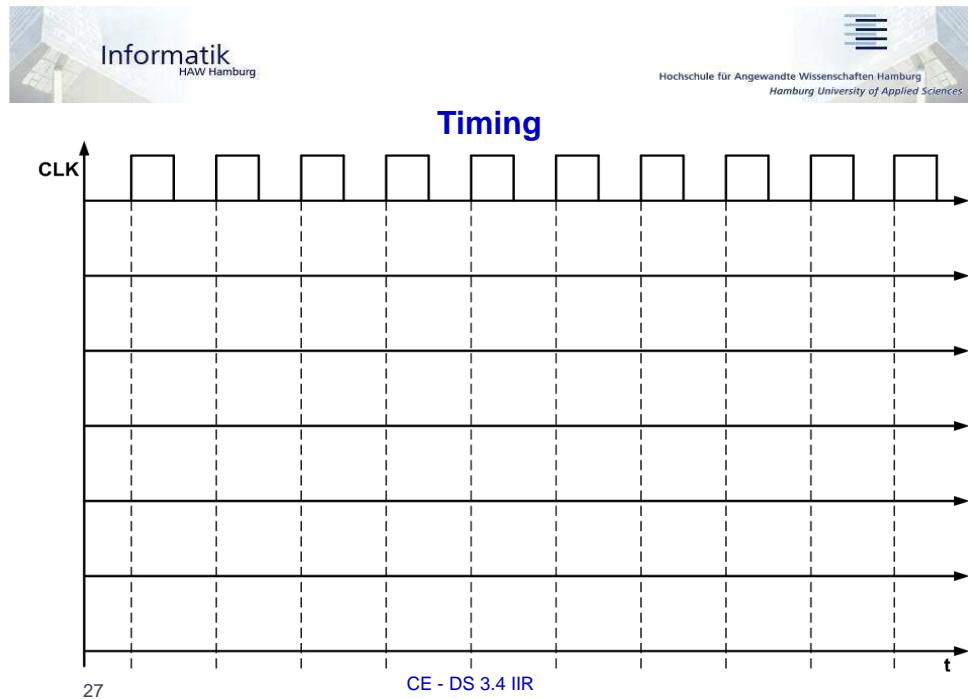
CE - DS 3.4 IIR

## Realisierung als Multizyklus-Datenpfad



26

CE - DS 3.4 IIR



27

Informatik  
HAW Hamburg

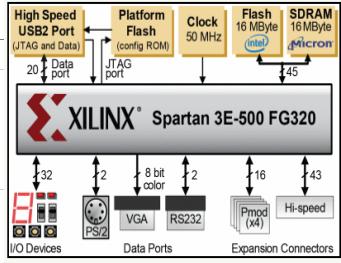


Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

# Computer Engineering WS 2012

## Digitale Systeme $\mu$ C - FPGA

Prof. Dr. B. Schwarz



```

graph TD
    HSP[High Speed USB2 Port] <--> JTAG[JTAG port]
    JTAG --> ROM[Platform Flash config ROM]
    ROM --> C[Clock 50 MHz]
    C --> F[Flash 16 MByte Intel]
    C --> S[SDRAM 16 MByte Micron]
    F <--> S
    IIO[I/O Devices] --- 32bit[32]
    IIO --- 2bit[2]
    IIO --- VGA[VGA]
    IIO --- RS232[RS232]
    32bit --- 8bit[8 bit color]
    8bit --- 2bit
    2bit --- PS2[PS2]
    16bit[16] --- Pmod[Pmod x4]
    43bit[43] --- HiSpeed[Hi-speed]
    Pmod --- HiSpeed
    
```

Informatik  
HAW Hamburg

Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## Synchronisation

### Einleitung: $\mu$ C – FPGA Kommunikation

Asynchrone Eingangssignale

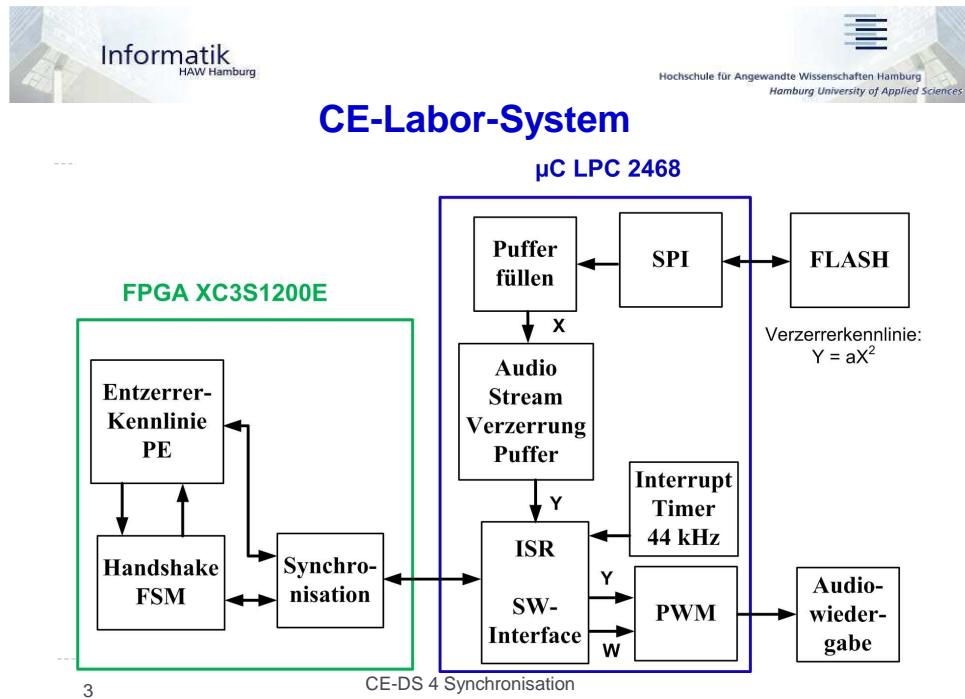
MTBF bei Flipflops mit metastabilen Zuständen

Synchronisationsschaltung für lange Impulse

Synchronisationsschaltung für kurze Impulse

### Kommunikation zwischen asynchronen Clock-Bereichen

Vier-Phasen Handshake



## Asynchrone Eingänge

### FPGA-intern:

- Synchrone Systeme mit einer Clock als Referenzsignal.
- Taktflankenereignis bestimmt die Datenaufnahme und die Ausgangsaktualisierung.
- Bedingung:  $T_{CLK} > T_{PD} + T_{LOGIK} + T_{SU}$

### Externe Eingänge:

- Pegeländerungen treten unabhängig von der FPGA-Clock auf.
- μC-Clock ist kein ganzes Vielfaches der FPGA-Clock.
- Phasenlage der Clocks ist unbestimmt.
- Variable Taktanzahl pro C-Anweisung.




## Asynchrone Kommunikation

**Ereignigesteuerte 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.**

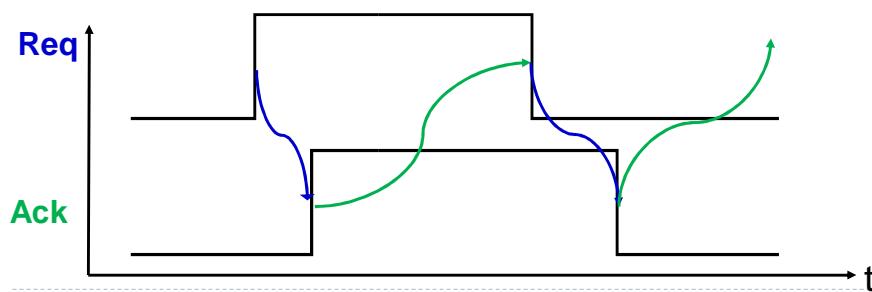
5

CE-DS 4 Synchronisation




## Handshake-Kommunikation

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

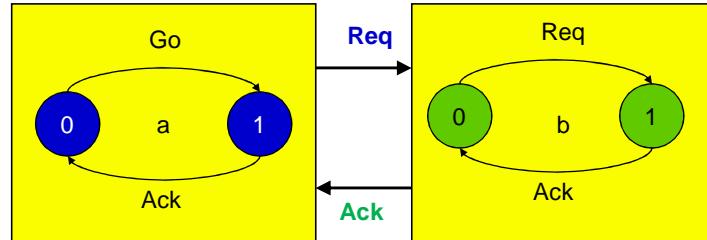


6

CE-DS 4 Synchronisation

## Wechselseitige Abstimmung

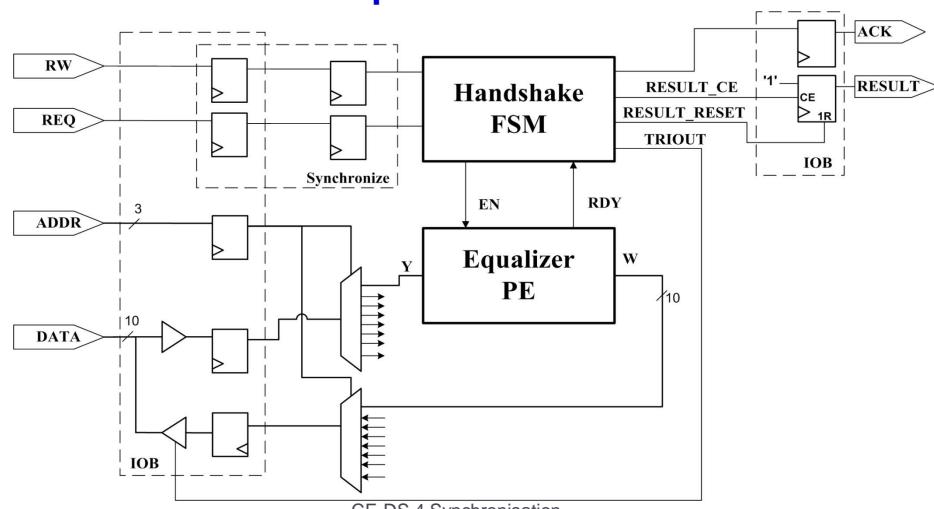
- Beide Seiten betreiben die Abstimmung mit einem Automaten.
- Zustandstransitionen zeigen den Fortschritt der abgestimmten Kommunikation an.



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.

9

CE-DS 4 Synchronisation



## Asynchrone Eingangssignale

- 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**  $t_E = t_s + t_h$  der abtastenden Clock.

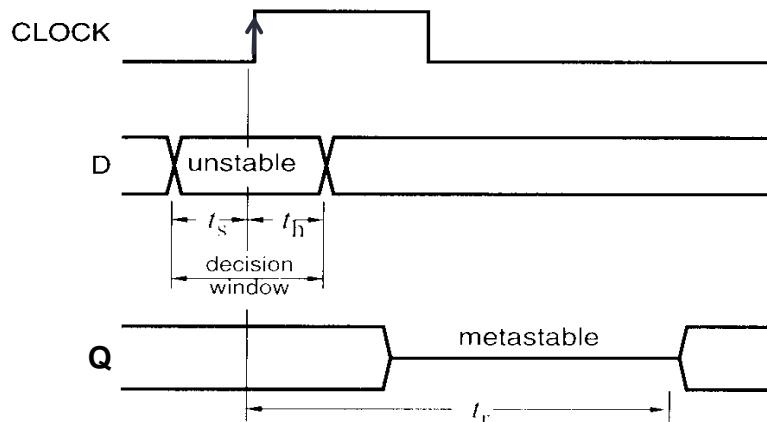
Dazu gehören z.B.:

- 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

## 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

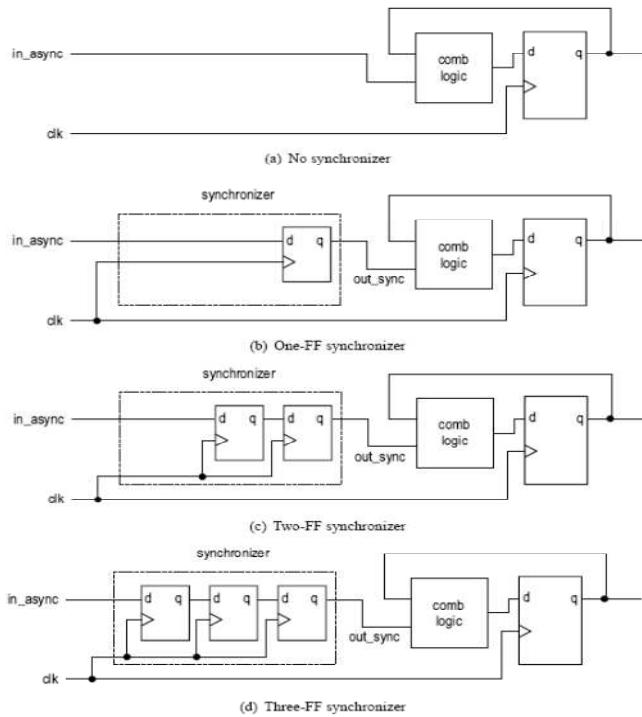


## Synchronisierer

**Unterschiedliche lange Zeitfenster für die Auflösungszeit  $t_r$  verfügbar.**

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

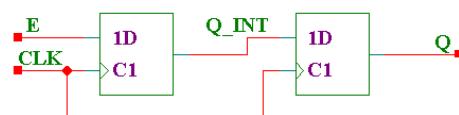
13



## 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

CE-DS 4 Synchronisation



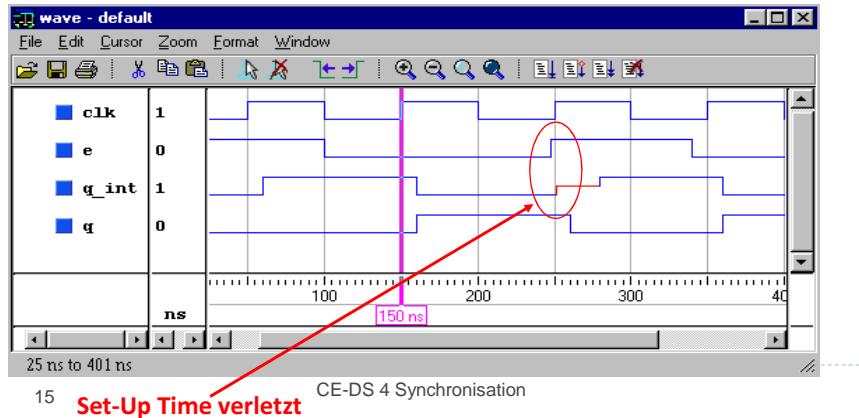
**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## 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.




**Informatik**  
HAW Hamburg



Hochschule für Angewandte Wissenschaften Hamburg  
Hamburg University of Applied Sciences

## 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.
  - **τ**: D-FF Konstante des Übergangsverhaltens von metastabilen Zuständen



## MTBF

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

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

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

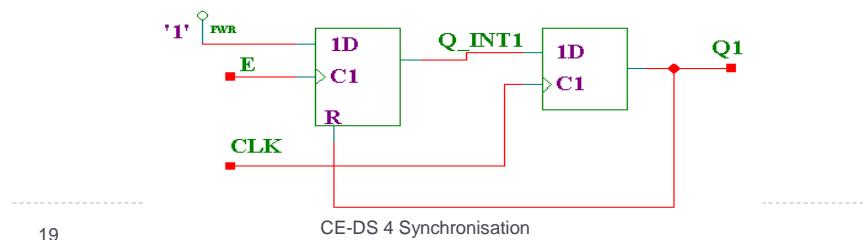


## Beispiel: MTBF

- 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  $10^5 \text{ 1/s}$  auf.
- Die verfügbare Zeit zum Abklingen des metastabilen Zustands ist  $tr = 1/f - t_{\text{SU}} = 80\text{ns}$ .
- 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  $16\text{MHz}$  getaktet werden soll, ergibt sich:  
 $tr = 42.5 \text{ ns} \rightarrow MTBF(42.5 \text{ ns}) = 3.1 \text{ s. Nich takzeptabel!}$
- Berechnen Sie die MTBF für den Baustein XC95108-20 jeweils bei  $f = 16\text{MHz}$  und  $f = 50\text{MHz}$ !

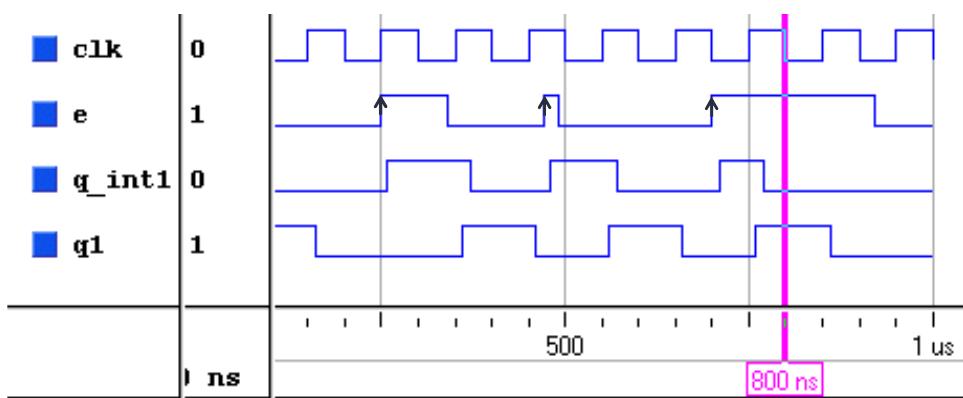
## Synchronisationsschaltung für kurze Impulse

- 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.



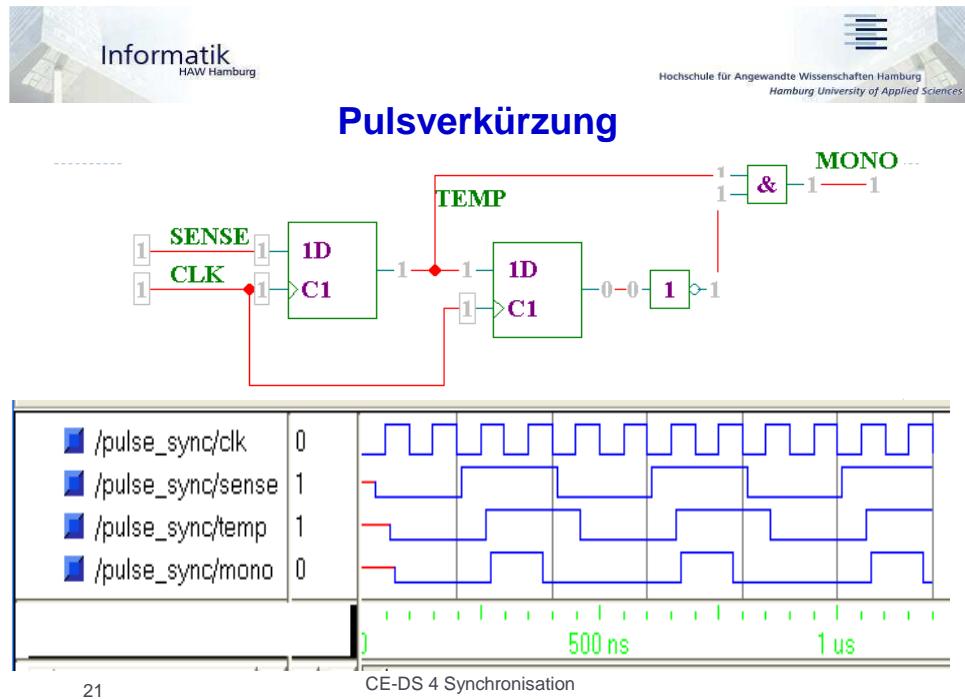
19

## Pulsverlängerung

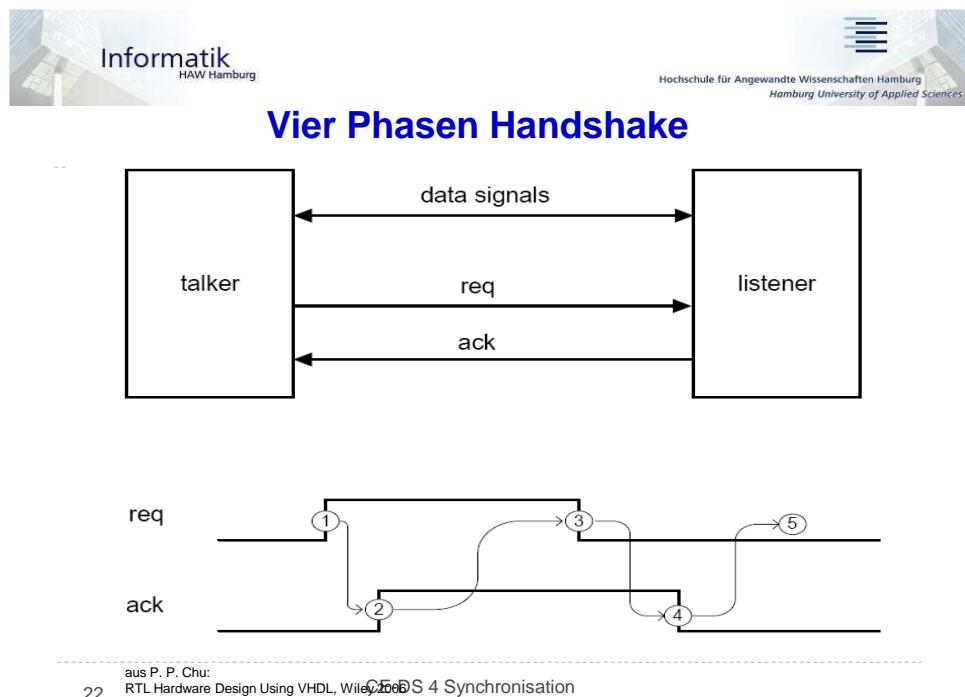


20

CE-DS 4 Synchronisation

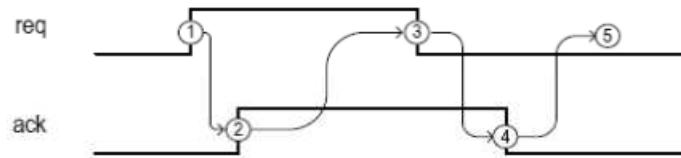


21



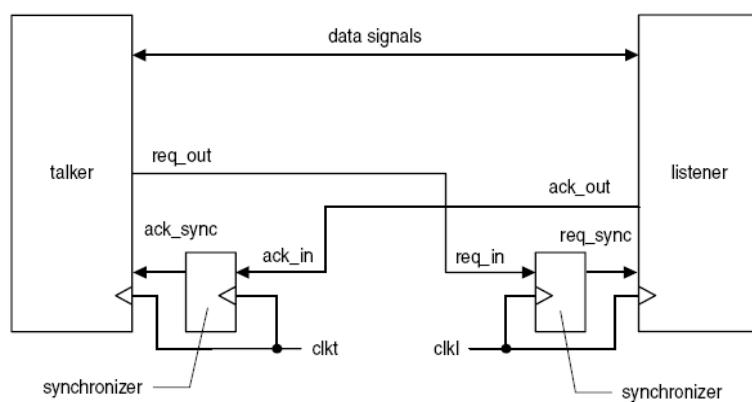
22

## Aufforderungs- und Bestätigungssequenz

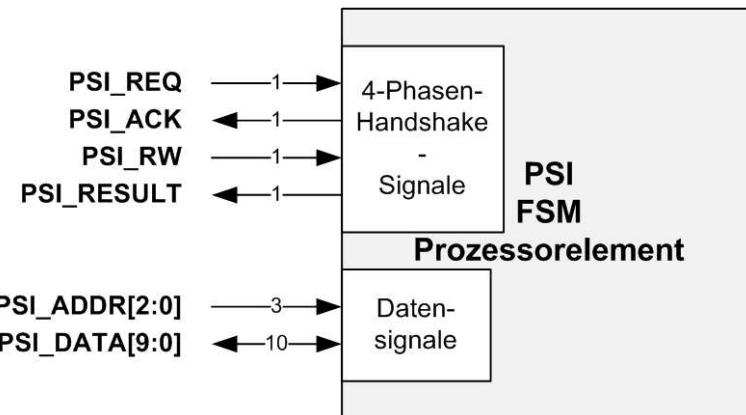


- Phase 1: Talker aktiviert      req = 1      (A)
- Phase 2: Listener aktiviert      ack = 1      (B, A)
- Phase 3: Talker deaktiviert      req = 0      (B, A)
- Phase 4: Listener deaktiviert ack = 0      (B)
- Talker kann neuen Request starten

## Synchronisation erforderlich für Talker und Listener in unterschiedlichen Clock Domains



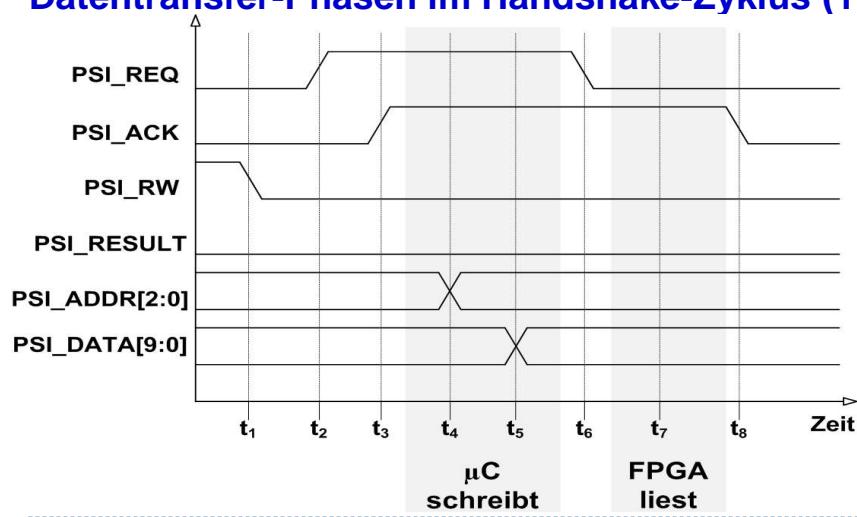
## $\mu$ C – FPGA Interface



25

CE-DS 4 Synchronisation

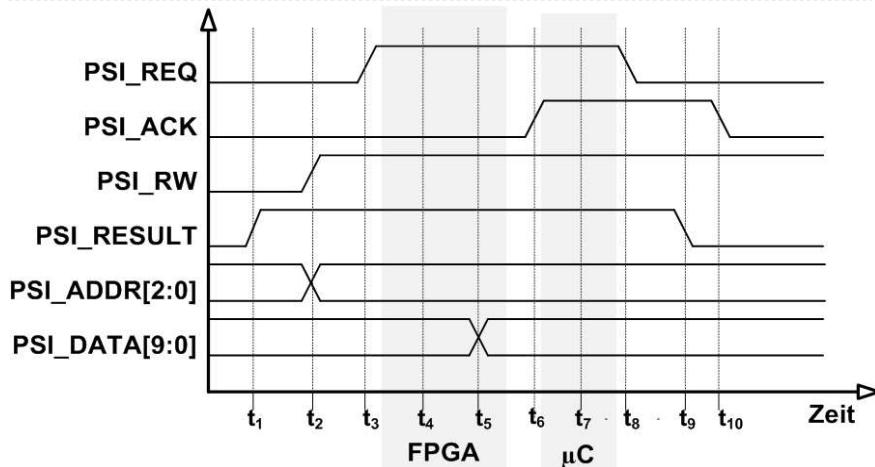
## Datentransfer-Phasen im Handshake-Zyklus (1)



26

CE-DS 4 Synchronisation

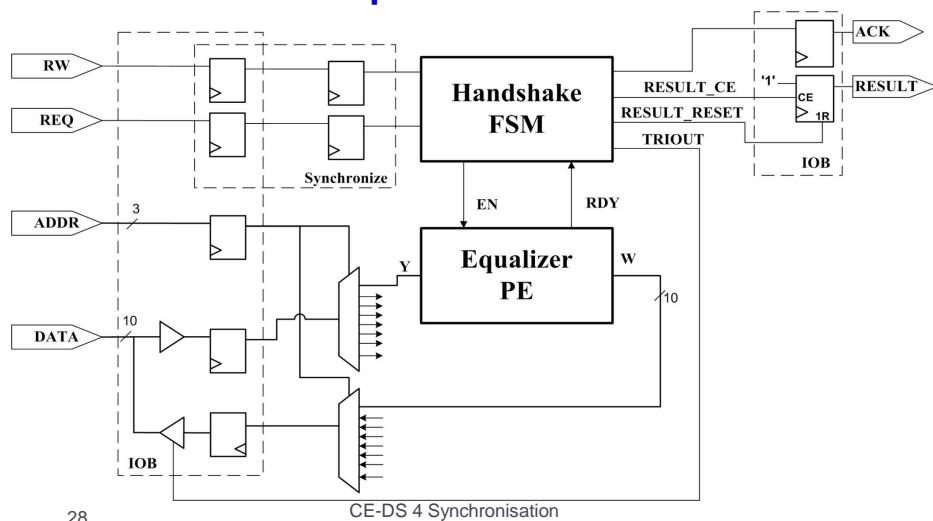
## Datentransfer-Phasen im Handshake-Zyklus (2)



27

CE-DS 4 Synchronisation

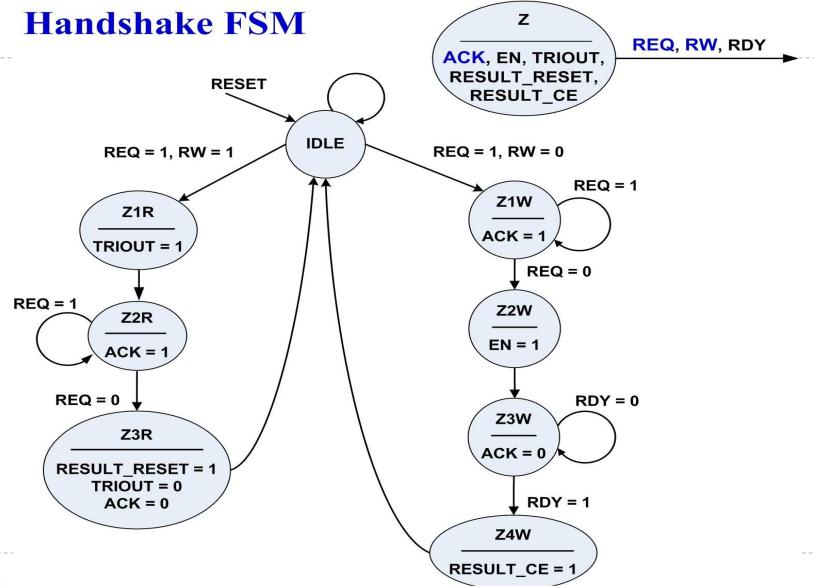
## Asynchrone Kommunikation $\mu$ C - FPGA



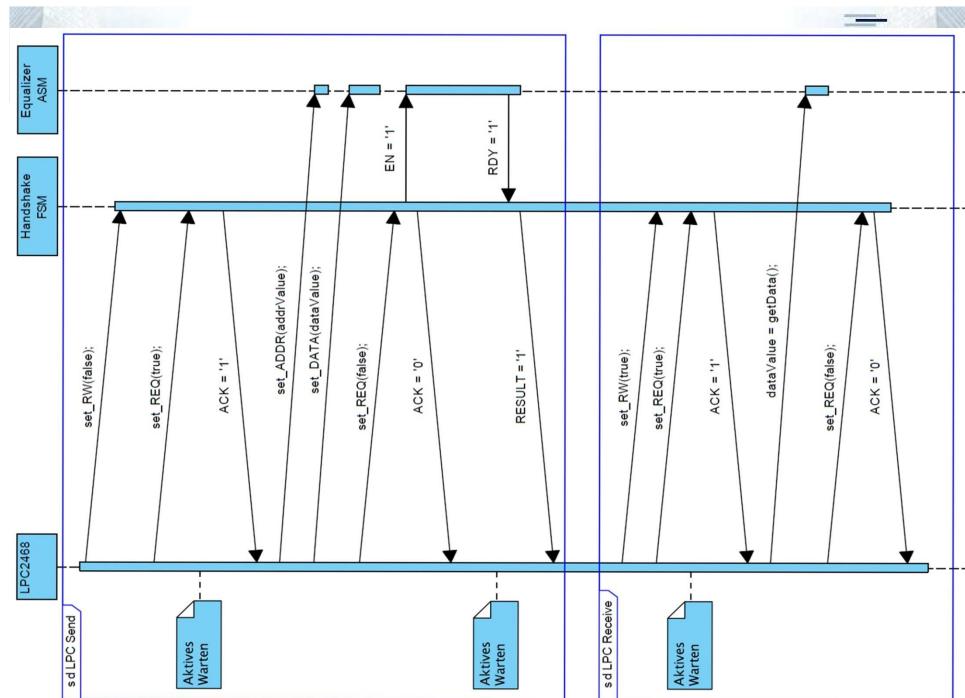
28

CE-DS 4 Synchronisation

## Handshake FSM

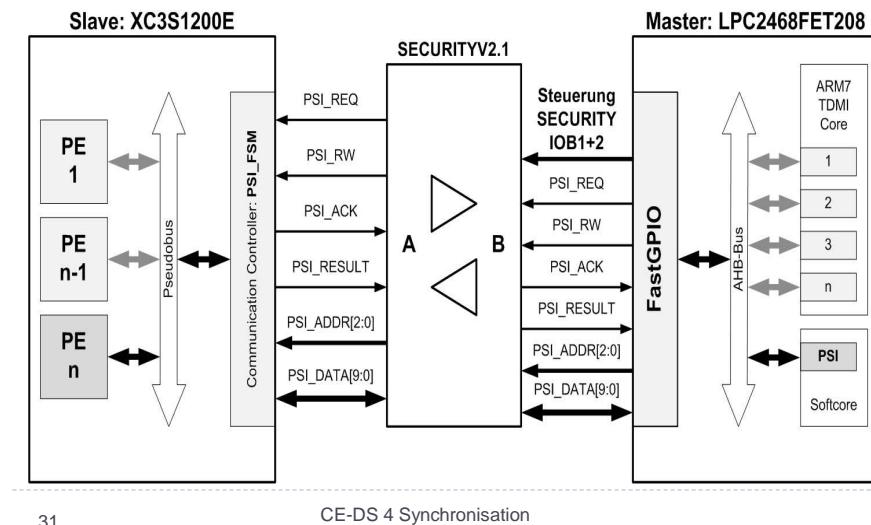


29





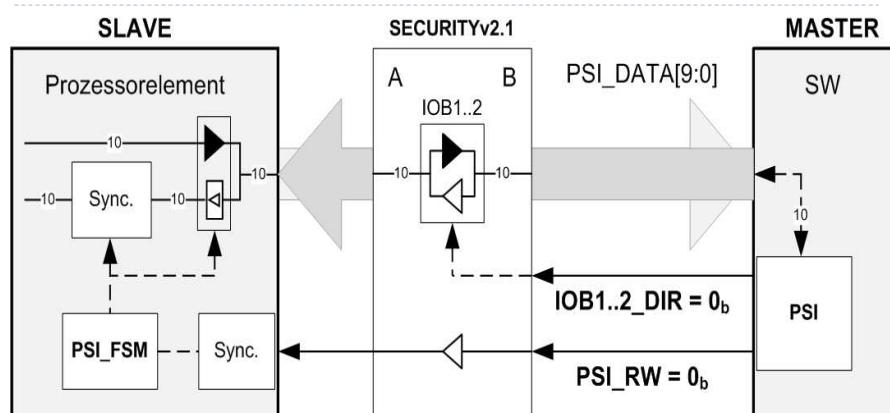
## Security Board Interface



31

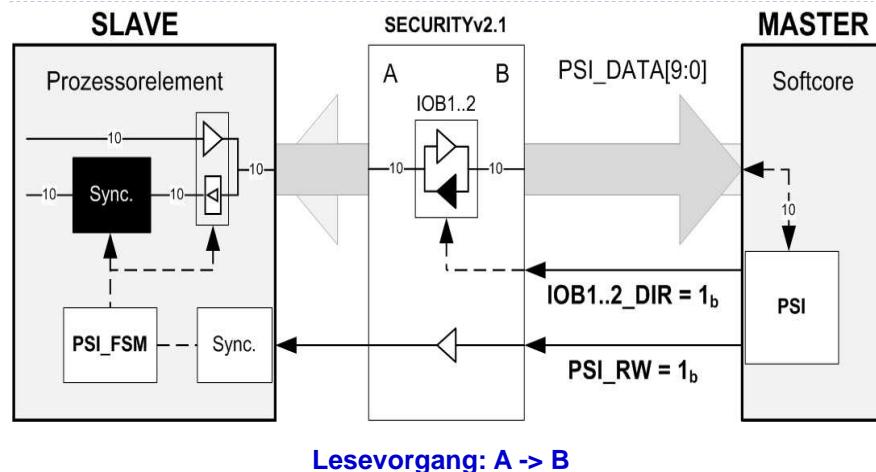


## Transfersteuerung μC - FPGA



32

## Transfersteuerung FPGA - µC



33

CE-DS 4 Synchronisation

## Write – Sequenz in ISR

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

34

CE-DS 4 Synchronisation



## Read – Sequenz in ISR

```
/** LPC-2468 Read from FPGA */
PSI_WAIT_RESULT_SET; /* 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

The logo features the text "Informatik HAW Hamburg" and "Hochschule für Angewandte Wissenschaften Hamburg" with "Hamburg University of Applied Sciences" underneath.

Name	Typ	Daten-/Steuersignale Anmerkung	Transfer	TI-LPC uC Port[Pin]	Development Boards			NEXYS2		
					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	JA1-31
PSI_ADDR[1]				P1[14]			34	IOB4<1>	C11	JA1-32
PSI_ADDR[2]				P1[13]			29	IOB4<2>	E11	JA1-33
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]	IOB2	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			
IOB2_DIR	S	Transferrichtung IOB2- PSI_DATA[9,8]	unidirektional	P2[4]	IOB2	X3	33			



```
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;
```