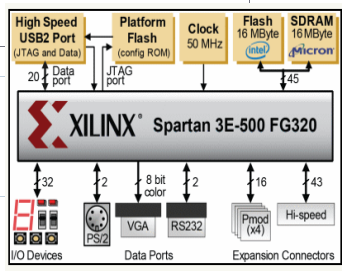


**Computer Engineering  
WS 2012**



**Digitale Systeme  
mit Rückführung**

Prof. Dr. B. Schwarz



**Digitale Systeme**



### 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**

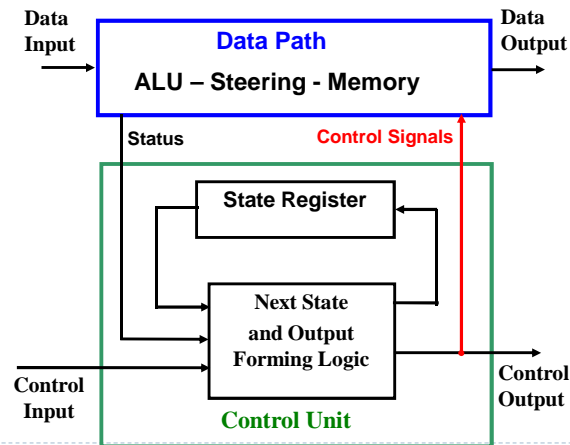
**Dynamische Systeme mit Rückführung**

**Multizyklusdatenpfad für ein Filter 1. Ordnung**

**Ergebnisübergabe an Folgezyklen**



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



3

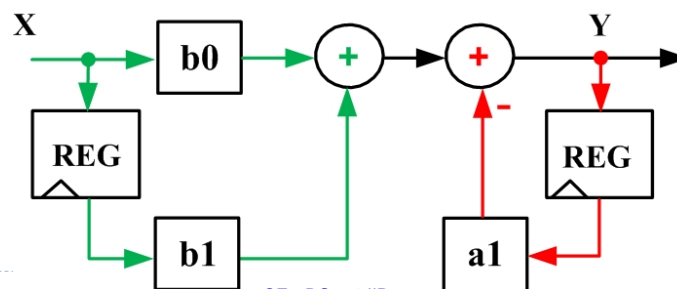
CE - DS 3.4 IIR



## Dynamische Systeme mit Rückführung

- Differentialgleichungen in der Audio-Signalverarbeitung  
Digitale Filter zur Veränderung des Signalspektrums
- Online-Simulation: Odometrie – Verfolgung von Fz-Bewegungen
- Differenzengleichungen für die SW- und VHDL-Modellierung

$$Y(n) = -a1 * Y(n-1) + b0 * X(n) + b1 * X(n-1)$$

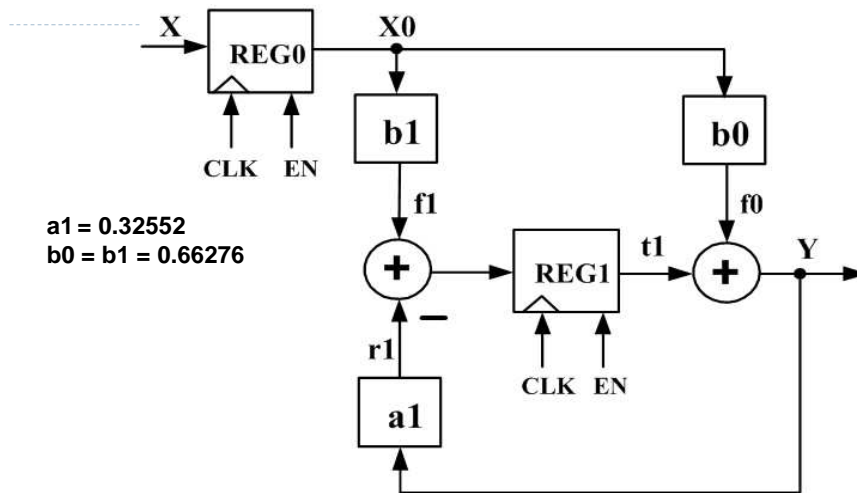


4

CE - DS 3.4 IIR



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



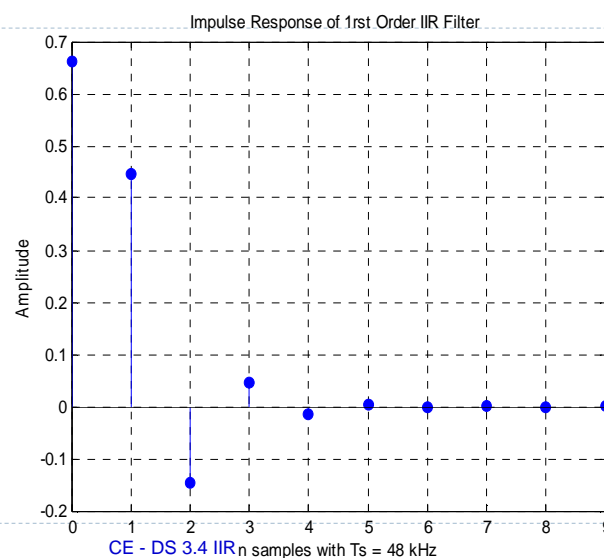
5

CE - DS 3.4 IIR



### Impulsantwort des IIR-Filters 1. Ordnung

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

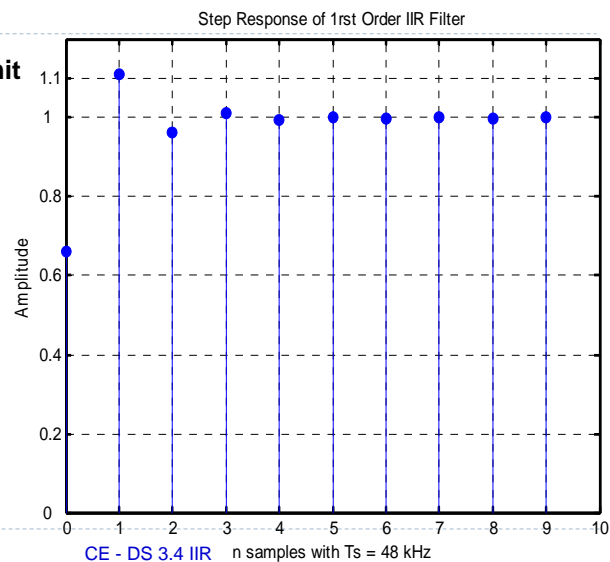


6

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

## Sprungantwort des IIR-Filters 1. Ordnung

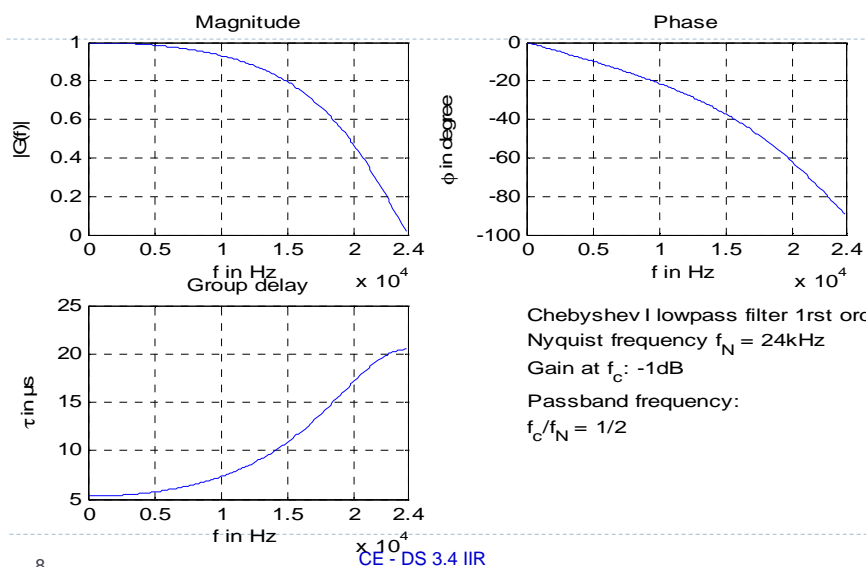
- Schritt-Anregung mit  
 $X(n \geq 0) = 1$



7

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

## Übertragungsverhalten des IIR-Filters 1. Ordnung



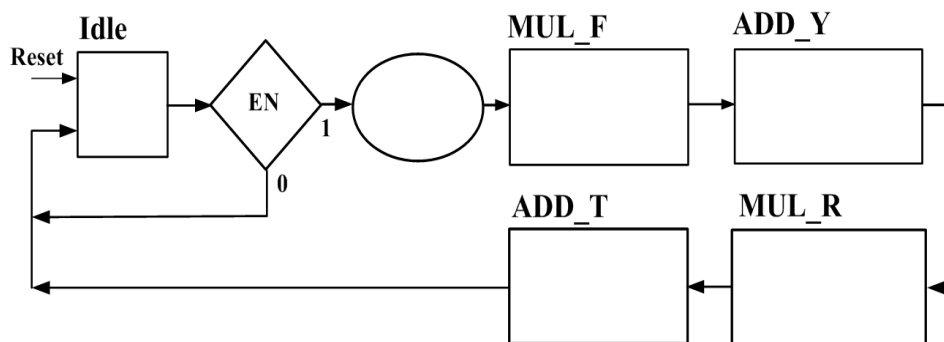
8



### 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:

$$Y(n) = t1 + b0 * X0(n); \quad t1+ = b1 * X0(n) - a1 * Y(n); \quad t1(n) = t1+(n-1)$$



9

CE - DS 3.4 IIR



### 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.

10

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  $a1$  u.  $b1 < 1$
- Register R1 u. R2 mit 10 Bit im Q9-Format für alle anderen  
     Summen und Produkte:    **sign. msb ... lsb**
- Produkt:     $f0=f1 = X0*b1 < 1$  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 :     $r1=a1*Y < 1$  wird für R2 auf Q9 reduziert
- Summen:     $Y=f0+t1 > 1$     **sign, g. msb ... lsb** in R0  
                $t1=f0-r1 < 1$     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 DR0: 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
          DR0 := ADD;
        else
          DR0 := X(X'left) & X; -- sign extension: X0 s,Q9
        end if;
        REG0 <= DR0 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;
end
15
CE - DS 3.4 IIR

```



### 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
        DR2 := ADD(WIDTH-1 downto 0); -- result Mux1
        -- 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
    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;

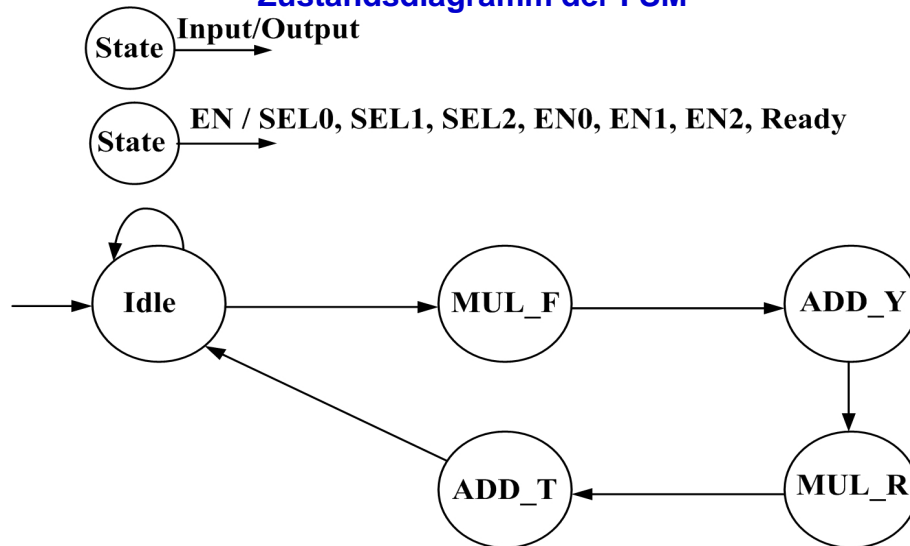
ADDER: ADD <= (REG1(REG1'left) & REG1) + (REG2(REG2'left) & REG2) after DELAY;
Y <= REG0;
16
CE - DS 3.4 IIR

```





### Zustandsdiagramm der FSM



17

CE - DS 3.4 IIR



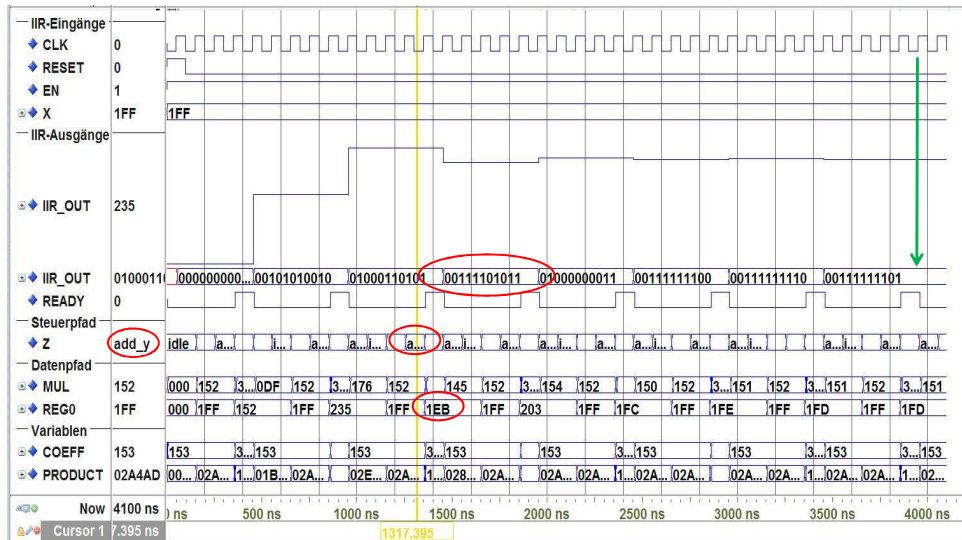
### Modell des IIR-Prozessorelementes (5)

```

--- Control path: FSM ---
Z_REG: process (CLK)

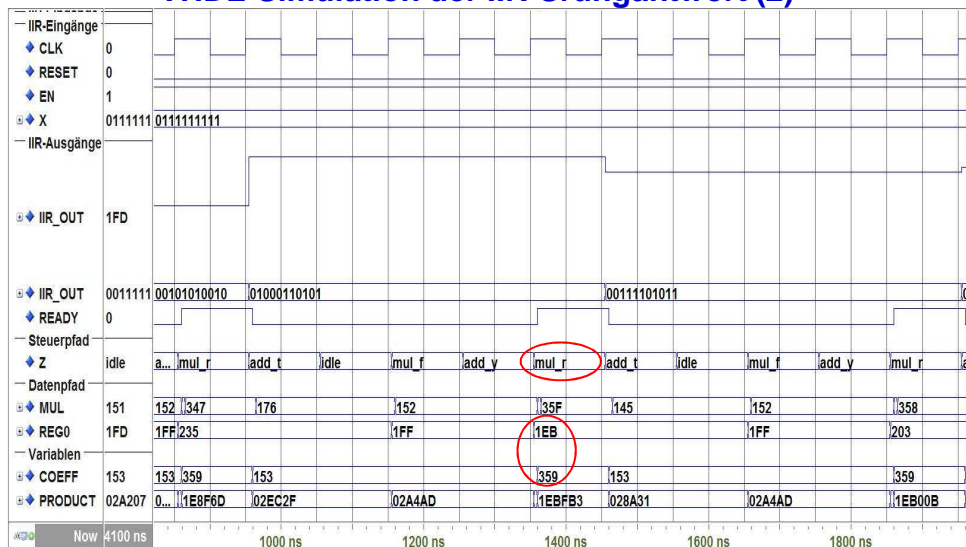
  A_UE_SN: process (Z, EN)
  begin
    -- default assignments with aggregate and type qualifier for string ""
    (EN0, EN1, EN2, SEL0, SEL1, SEL2, READY) <= std_logic_vector("0000000") after DELAY;
    Z_PLUS <= IDLE after DELAY;
    case Z is
      when IDLE =>
        if (EN = '1') then
          Z_PLUS <= MUL_F after DELAY;
          EN0 <= '1' after DELAY; -- X in reg0 with sel0=0
        end if;
      when MUL_F =>
        EN1 <= '1' after DELAY; -- 1st coeff with sel2=0 f0=b1*x0; reg1
        Z_PLUS <= ADD_Y after DELAY;
      when ADD_Y =>
        EN0 <= '1' after DELAY; -- y=f0+t1 in reg0
        SEL0 <= '1' after DELAY;
        Z_PLUS <= MUL_R after DELAY;
      when MUL_R =>
        EN2 <= '1' after DELAY; -- r1=a1*y in reg2 mit sel1=0
        SEL2 <= '1' after DELAY; -- 2nd coeff selected
        Z_PLUS <= ADD_T after DELAY;
        READY <= '1' after DELAY; -- Enable für Weiterverarbeitung
      when ADD_T =>
        EN2 <= '1' after DELAY; -- t1=f1+r1 in reg2
        SEL1 <= '1' after DELAY; -- result mux
      when others => null;
    end case;
  end process A_UE_SN;
end MULTI_CYCLE;
  
```

## VHDL-Simulation der IIR-Sprungantwort



19

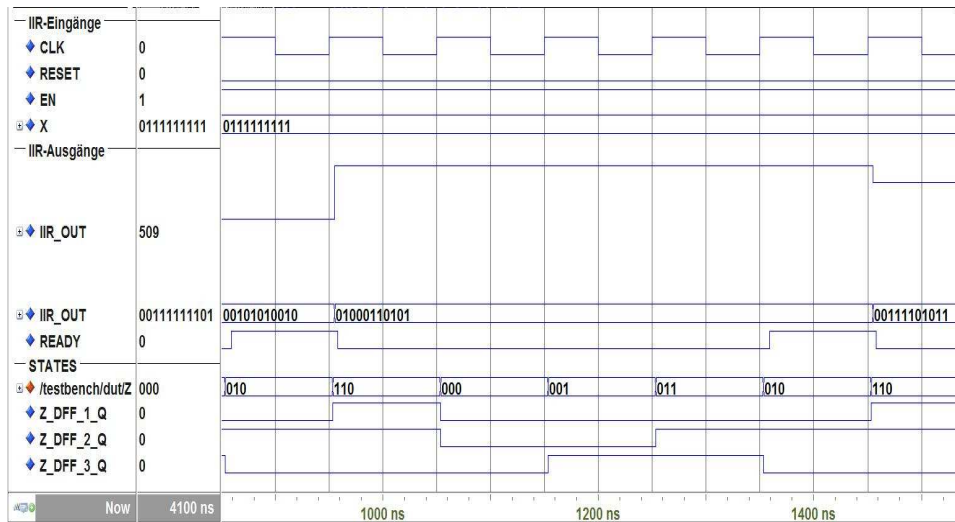
## VHDL-Simulation der IIR-Sprungantwort (2)



20



## Timing-Simulation der IIR-Sprungantwort

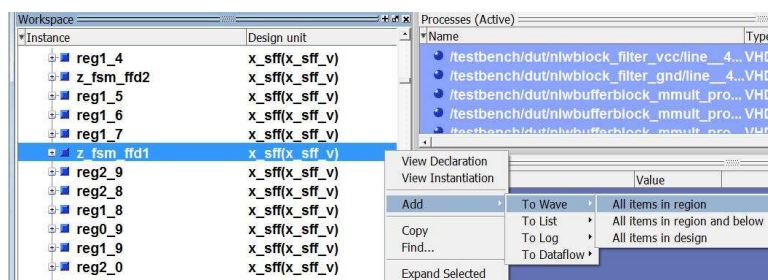


21

CE - DS 3.4 IIR



## Timing-Simulation: Selektion interner Register



add wave -noupdate -divider "STATES"

add wave -height 30 -label Z\_DFF\_1\_Q sim:/testbench/dut/z\_fsm\_ffd1/o

add wave -height 30 -label Z\_DFF\_2\_Q sim:/testbench/dut/z\_fsm\_ffd2/o

add wave -height 30 -label Z\_DFF\_3\_Q sim:/testbench/dut/z\_fsm\_ffd3/o

add wave -height 30 -label Z\_DFF\_1\_D sim:/testbench/dut/z\_fsm\_ffd1/i

add wave -height 30 -label Z\_DFF\_2\_D sim:/testbench/dut/z\_fsm\_ffd2/i

add wave -height 30 -label Z\_DFF\_3\_D sim:/testbench/dut/z\_fsm\_ffd3/i

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:

$$u = dy/dx = y' \Rightarrow$$

$$u' = du/dx = -3xu - 3y$$

- Übergang auf Differenzen mit

$$\Delta u = u(n) - u(n-1)$$

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

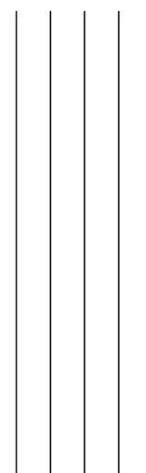
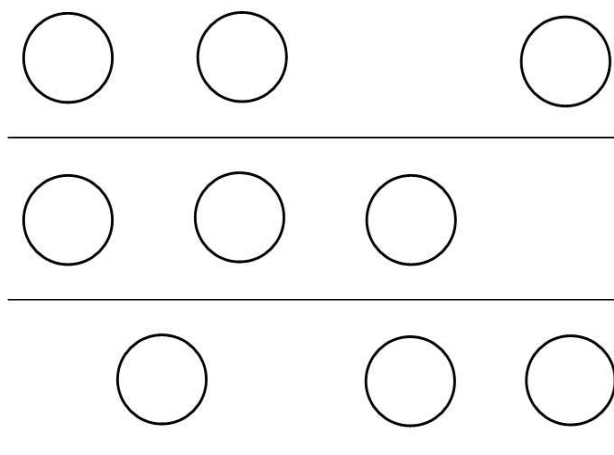
$$u(n) = u(n-1) - x(n-1)u(n-1) \cdot 3\Delta x - y(n-1) \cdot 3\Delta x$$

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

23

CE - DS 3.4 IIR

## Sequenzgraph



24

CE - DS 3.4 IIR



## Phasendiagramm zum Pipelinesystem mit Rückkopplung

|       |                         |                         |                               |                  |                  |                               |
|-------|-------------------------|-------------------------|-------------------------------|------------------|------------------|-------------------------------|
| St. 3 | $f3(0, 0, 0) = 0, 0, 0$ | $f3(0, 0, 0) = 0, 0, 0$ | $f3(u0, y0, x0) = u1, y1, x1$ |                  |                  | $f3(u1, y1, x1) = u2, y2, x2$ |
| St. 2 | $f2(0, 0, 0)$           | $f2(u0, y0, x0)$        | $f2(0, 0, dx)$                |                  | $f2(u1, y1, x1)$ |                               |
| St. 1 | $f1(u0, y0, x0)$        | $f1(0, 0, dx)$          | $f1(0, 0, dx)$                | $f1(u1, y1, x1)$ |                  |                               |

- Nach  $n=3$  Takten liegt ein erstes Ergebnis  $(u1, y1, x1)$  ausgehend von den Anfangswerten  $(u0, y0, x0)$  aus der 3. Stufe vor (Latenz  $n = 3$ ).
- Mit dem  $(n+1)$ -Takt ist der zweite Zyklus gestartet.
- Die zwischenzeitlichen (Takte  $n-1, n-2, n+2, \dots$ ) Ergebnisse basieren auf Initialisierungen der Stufen 1. u. 2. und liefern keine verwertbaren Beiträge zum Anfangswertproblem mit  $(u0, y0, x0)$ .
- Der Durchsatz ist somit  $fclk/n$ , sodass kein Vorteil durch das Pipelining erreicht wird.
- Ein äquivalenter Multizyklus-DP mit dem Durchsatz  $fclk/n$  kann also mit geringerem Aufwand an HW-Ressourcen realisiert werden.

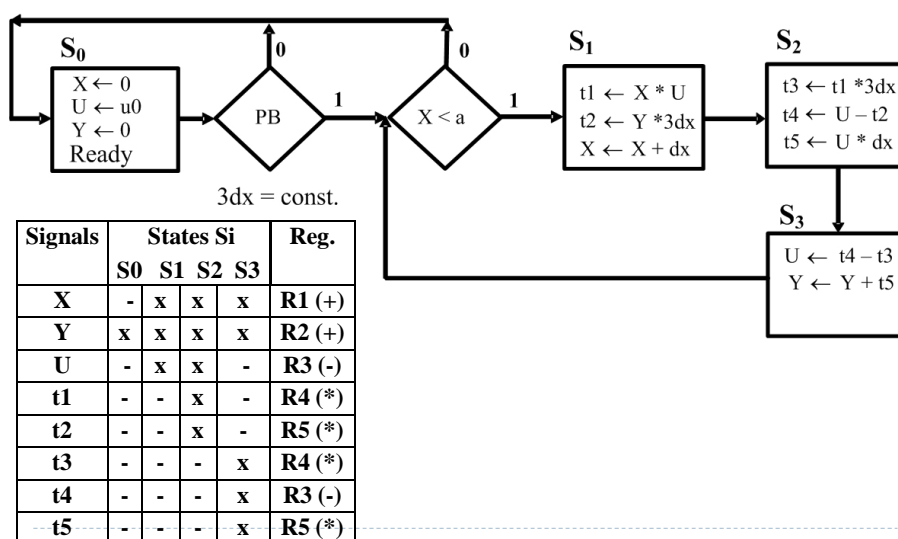
25

CE - DS 3.4 IIR



Register aus gleichem Operator gefüllt: kein MUX am Registeringang

## Realisierung als Multizyklus-Datenpfad



26

CE - DS 3.4 IIR

## Timing

