Schieberegister

Das ist zuständig für das Speichern und Bewegen von Daten, deshalb wird es für den Entwurf von Rechnern genutzt.

Ein Schieberegister mit n Bits kann aus n Flip-Flops konstruiert werden, aber durch einen gemeinsamen Takt gesteuert werden.

Bei jedem Takt werden alle Bits des Register gleichzeitig aktualisiert

4 Arten

- Serial-in zu Parallel-out (SIPO): In diesem Modus werden die Eingabedaten seriell eingelesen und anschließend parallel ausgelesen.
- Serial-in to Serial-out (SISO): In diesem Modus werden die Eingabedaten seriell eingelesen und anschließend seriell ausgelesen.
- Parallel-in to Serial-out (PISO): In diesem Modus werden die Eingabedaten parallel eingelesen und anschließend seriell ausgelesen.
- Parallel-in to Parallel-out (PIPO): In diesem Modus werden die Eingabedaten parallel eingelesen und anschließend parallel ausgelesen.

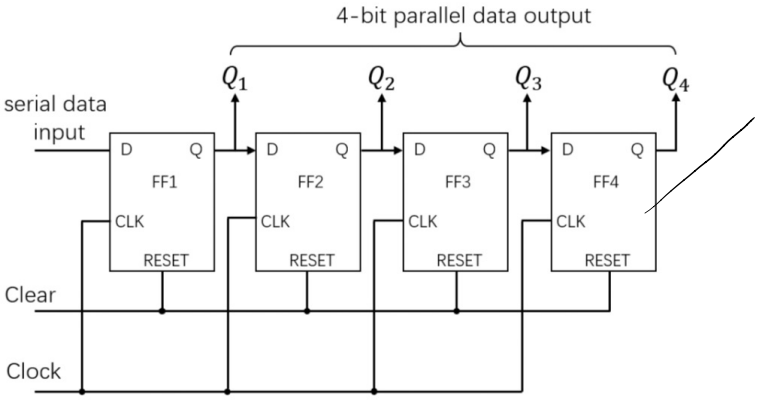


Abbildung 1: SIPO Schieberegister.

D-latch

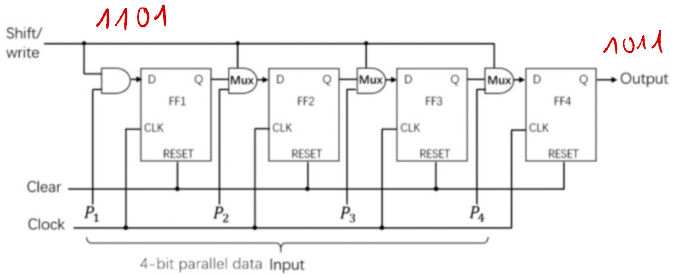
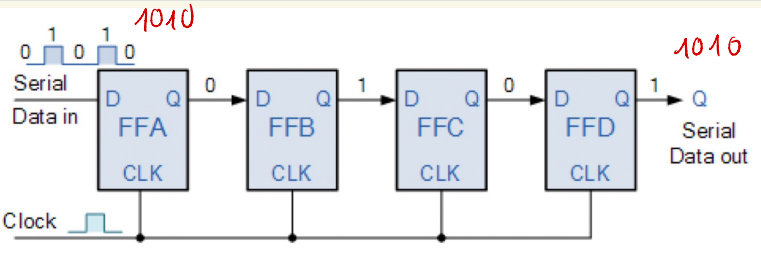
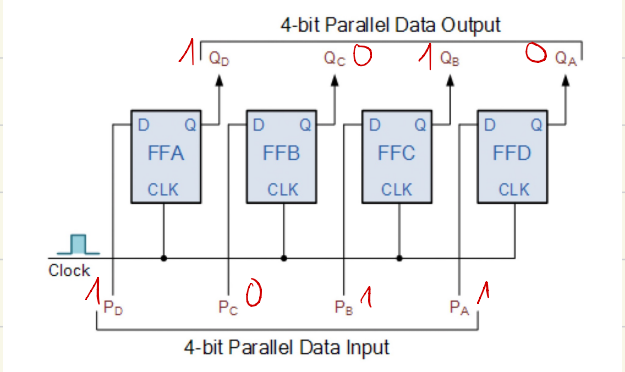


Abbildung 2: PISO Schieberegister.



SISO



PIPO

Synchroner Vorwärts - Rückwärts - Zähler

es wird aus 2 JK Flip-Flops aufgebaut. 4 Eingaben (clk, reset, C, D)
2 Ausgabe (Q_0, Q_1)

sync_counter.vhdl

```
library ieee;
use ieee.std_logic_1164.all;

entity sync_counter is
    port(
        reset, clk, C, D : in std_logic;
        q1, q0 : out std_logic
    );
end sync_counter;

architecture bh of sync_counter is
    component jk_flipflop is
        port(
            j,k, clk, reset : in std_logic;
            Q : out std_logic
        );
    end component;

        signal z0,z1,j0,j1,k0,k1 : std_logic;

    begin
                j0 <= C and ((not D) or z1);
                k0 <= C and (D or (not z1));
        j1 <= C and (not D) and z0;
        k1 <= C and D and z1 and (not z0);
        jk_flipflop_0 : jk_flipflop port map(j=>j0, k=>k0, clk=>clk, reset=>reset, q=>z0);
        jk_flipflop_1 : jk_flipflop port map(j=>j1, k=>k1, clk=>clk, reset=>reset, q=>z1);
        q0<=z0;
        q1<=z1;
end bh;
```

$$Q_0 = z_0$$
$$Q_1 = z_1$$

$$j_0 = C \wedge (\bar{D} \vee z_1)$$
$$k_0 = C \wedge (D \vee \bar{z_1})$$
$$j_1 = C \wedge \bar{D} \wedge z_0$$
$$k_1 = C \wedge D \wedge z_1 \wedge \bar{z_0}$$

Unterschied Moore Automat und Mealy Automat

• Moore Automat

– Vorteile: Einfacher umzusetzen da Ausgabe nur abhängig von Zustand; Ausgabe synchron zum Takt, daher robust gegen Glitches

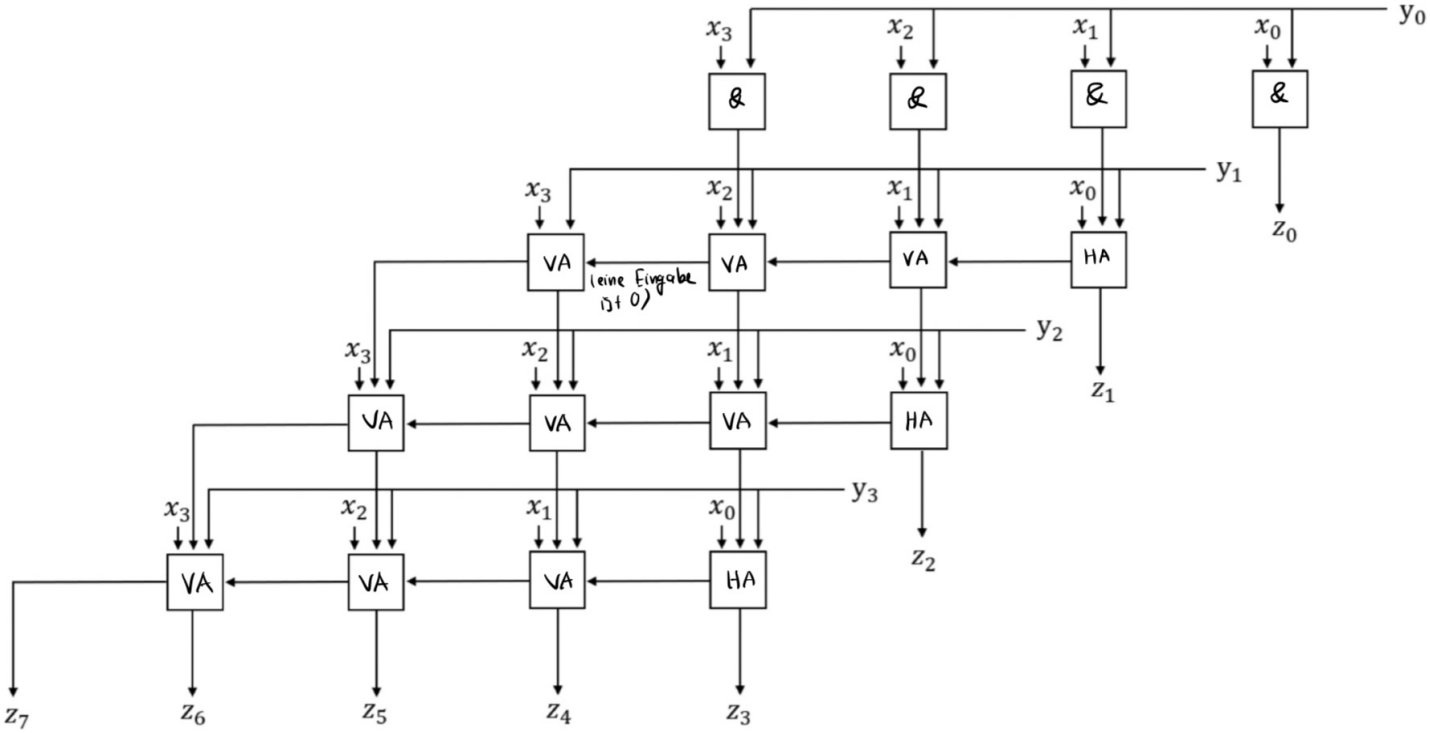
– Nachteile: Meist mehr Zustände als Mealy; Mindestens ein Takt erforderlich um auf Eingabe zu reagieren

• Mealy Automat

– Vorteile: Meist weniger Zustände, dadurch kleinere Schaltungen

– Nachteile: Ausgabe asynchron zum Takt, dadurch Glitches möglich

Blatt 7



Add-Shift Multiplizier

nutzt Schieberegister, Addierwerk und endliche Automaten

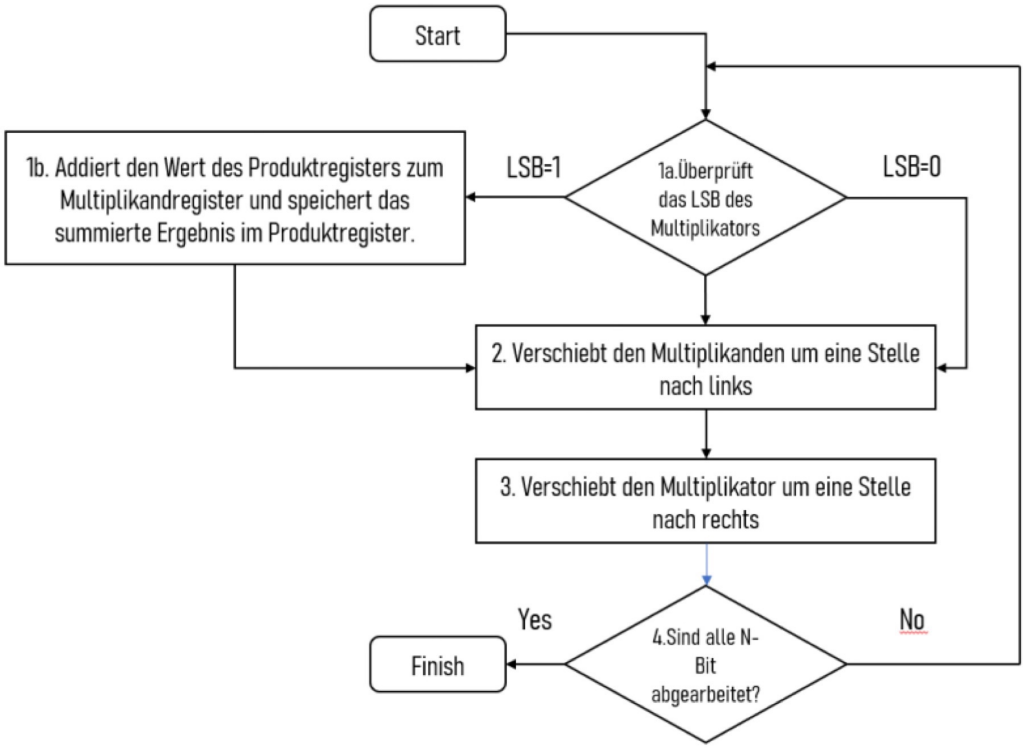


Abbildung 6: Flussdiagramm des Multiplizierwerk nach der *add shift* Methode.

c)

Add-shift-Multiplexer

- verwendet eine schrittweise Methode.
- multipliziert den Multiplikator mit jedem Bit des Multiplikanden und addiert die Teilergebnisse schrittweise, indem er die Bits des Multiplikators nach links schiebt.

Parallel-Multiplexer

- verwendet eine parallele Methode
- multipliziert den Multiplikator mit allen Bits des Multiplikanden gleichzeitig und addiert die Teilergebnisse in einem Schritt.

Add-shift-Multiplexer

Vorteil

- einfache Implementierung: nur Additionen und Shift-Operationen verwendet
- geringerer Hardwarebedarf: im Vergleich zum Parallel-Multiplexer

Nachteil

- langsam, große Berechnungszeit: da er schrittweise erfolgt, benötigt mehr Takte
- Höhere Latenz: das Ergebnis erst später verfügbar ist.

Parallel-Multiplexer

Vorteil

- schneller: da er parallel erfolgt
- niedrigere Latenz: das Ergebnis schneller verfügbar ist.

Nachteil

- Komplexere Implementierung: er erfordert eine komplexere Schaltung
- Höherer Hardwareaufwand.
- Höherer Stromverbrauch: Aufgrund der Verwendung einer größeren Anzahl von Schaltelementen