
Digitaltechnik

Kapitel 6, Schaltungsstrukturen

Prof. Dr.-Ing. M. Winzker

Nutzung nur für Studierende der Hochschule Bonn-Rhein-Sieg gestattet.
(Stand: 20.03.2019)

6.1 Grundstrukturen digitaler Schaltungen

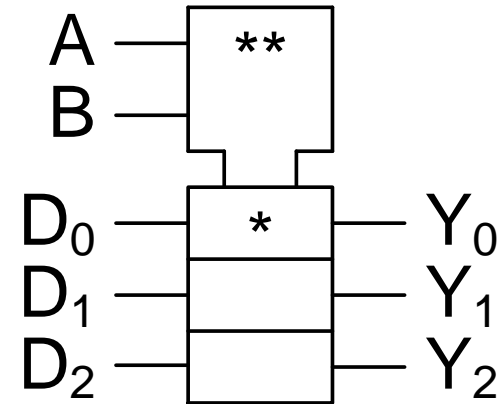
- Prinzipiell kann jede Schaltung mit Funktionstabelle bzw. Zustandstabelle erstellt werden
- Es gibt jedoch **Grundstrukturen**, die sich häufig in Schaltungen wiederfinden
- Diese Grundstrukturen sollten bekannt sein, damit man sie als Struktur verwenden kann
 - Diskrete Bauelemente
 - Schaltungsstruktur in VHDL

Siehe „Golden Rules of VHDL“:

- Have a general concept of what your hardware should look like

Darstellung von Schaltungsstrukturen

- Für die Darstellung von Digitalschaltungen gibt es eine standardisierte Darstellung
 - Eingänge links, Ausgänge rechts
 - Oberer Block mit Steuersignalen, welche die Datensignale beeinflussen
 - Unterer Block mit Datensignalen
 - Horizontaler Strich trennt voneinander unabhängige Datensignale
 - Abkürzungen und Symbole bei „*“ und „**“ geben die Funktion an
- Für viele der Schaltungen gibt es entsprechende diskrete Bauelemente aus der „4000er-Serie“
- Verwendung hauptsächlich in deutscher Literatur
- In der Praxis meist einfache Blockdiagramme mit Erläuterungen
- Bedeutung der Symbolik sollte verstanden werden



Top-Down Entwurf

- Der Entwurf von digitalen Systemen erfolgt üblicherweise nach dem **Top-Down Prinzip**
 - Ausgehend von der Spezifikation wird das Gesamtsystem in Teilschaltungen aufgeteilt
 - ➔ Bezeichnung auch: Untermodul
 - Die Untermodule werden wiederum in weitere Untermodule aufgeteilt
 - Als Teilschaltungen und Grundmodule werden oft bekannte Schaltungsstrukturen verwendet
- Beispiele für Teilschaltungen:
 - CPU, Filter, Speicher, ...
- Beispiele für Grundmodule:
 - Zähler, Mealy-/Moore-Automat, RAM, ROM, Addierer, Multiplexer, ...
- Die Untermodule werden dann einzeln entworfen und **Bottom-Up** bis zur Gesamtschaltung zusammengesetzt
 - Entwurf der Untermodule kann auf verschiedene Personen aufgeteilt werden
 - In großen Konzernen auch Aufteilung auf mehrere Standorte möglich

Top-Down Entwurf: Display-Controller für Beamer

- Display-Controller ist die Steuereinheit eines Daten- und Video-Projektors („Beamer“)

Funktionen

- Signalverarbeitung
 - Skalierung der Eingangsbilder auf Displaygröße
 - Deinterlacing für Video-Signale (Halbbilder auf Vollbilder)
 - Freeze: Einfrieren des Bildes
 - Keystone-Korrektur
 - Overlay des On-Screen-Display
- Steuerung des gesamten Geräts
 - Einschalten der Lampe, Lüftersteuerung, Betriebsstunden zählen, ...
 - Reaktion auf Tastendrucke, IR-Fernbedienung, ...
 - Auswahl der Eingangsquelle

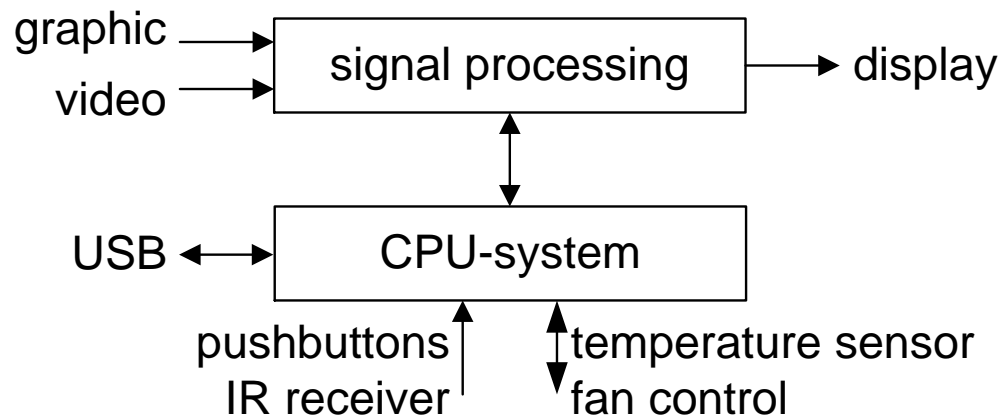
(image: <http://www.liesegang.de>)



Top-Down Entwurf: Display-Controller für Beamer (II)

Aufteilung in zwei Untermodule

- Signalverarbeitung
 - Vorgegebene Algorithmen, hohe Rechenleistung
- CPU-System für Steuerung
 - Flexibel programmierbar, geringe Rechenleistung

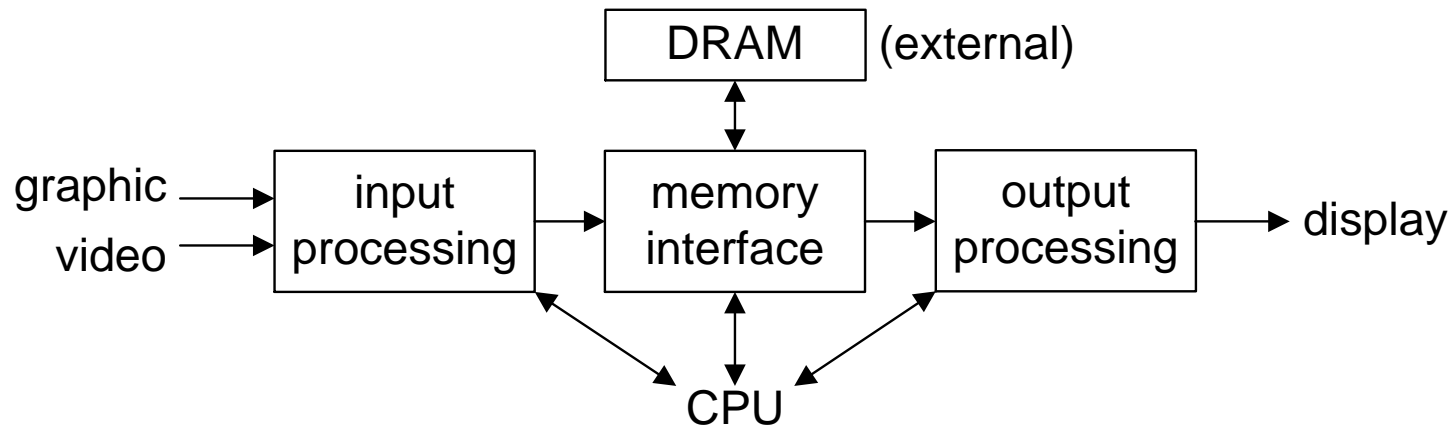


- Hintergrund dieser Aufteilung:
 - Bekannte Strukturen aus Lehrbüchern und Veröffentlichungen

Top-Down Entwurf: Signalverarbeitung

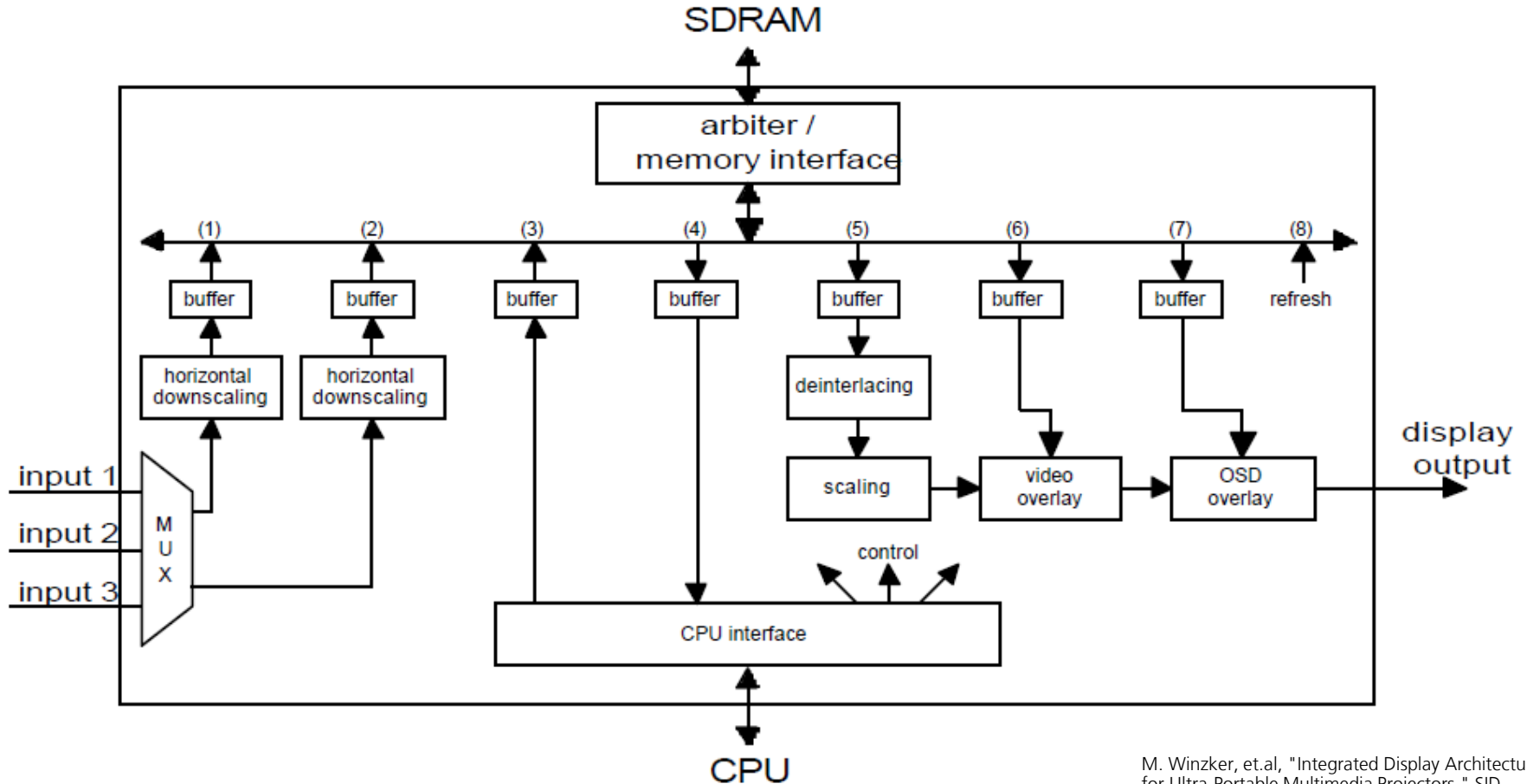
Aus Anforderungen ergibt sich weitere Struktur der Signalverarbeitung

- Skalierung, Keystone-Korrektur
 - ➔ Filter am Eingang und/oder Ausgang
- Einfrieren des Bildes
 - ➔ Bildspeicher
- Bildspeicher benötigt 2 Bilder (Wechsepuffer), 1280x1024 Pixel, 3 Farben, 8 Bit pro Farbe
 - ➔ Externer Bildspeicher: DRAM



Top-Down Entwurf: Signalverarbeitung (II)

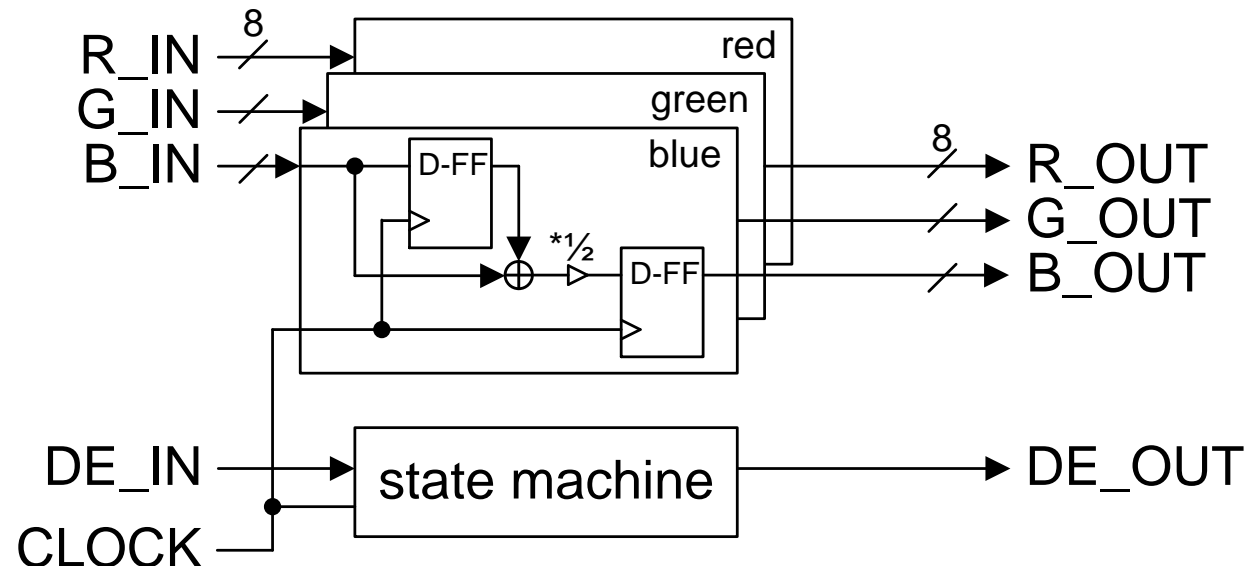
- Weitere Konkretisierung der Teilschaltungen
- Entwurf der Untermodule, z.B. Skalierung als Filter



M. Winzker, et.al, "Integrated Display Architecture for Ultra-Portable Multimedia Projectors," SID International Symposium, 2000.

Top-Down Entwurf: Horizontal Downscaling

- Extrem große Bilder sollen vor der Speicherung um dem Faktor zwei herunterskaliert werden
 - Besser das Bild wird mit schlechter Qualität angezeigt, als gar nicht
- Horizontal: Mittelwert aus zwei benachbarten Bildpunkten
- Vertikal: Jede zweite Zeile wird ausgelassen
 - ➔ Speicher für Bildzeilen wird gespart
- Automat („state machine“) lässt „data enable“ für jedes zweite Pixel und jede zweite Zeile aus



6.2 Kombinatorische Grundstrukturen

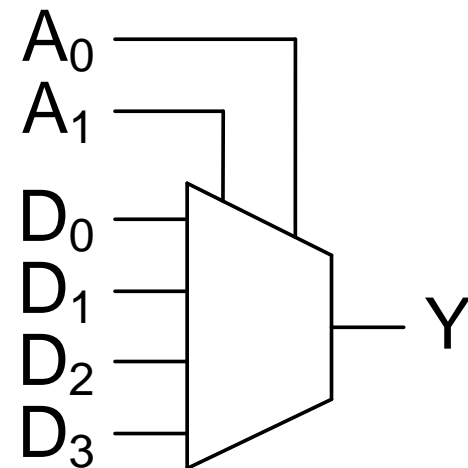
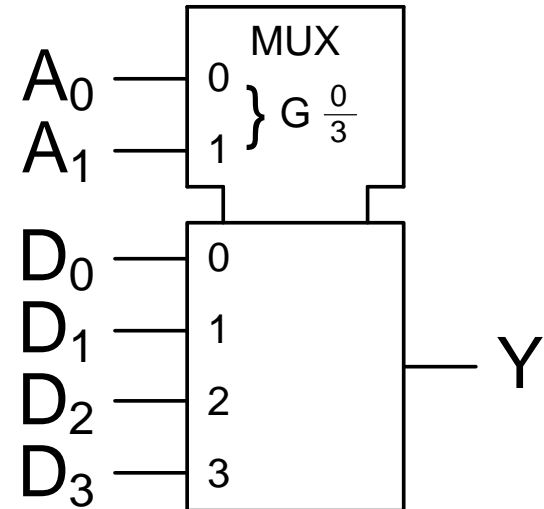
Multiplexer / Datenselektor

- Steuersignale A_0 , A_1 wählen einen der vier Eingänge für den Ausgang aus

- Funktionstabelle für 1-aus-4 Multiplexer

A_1	A_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

- In der Praxis wird oft eine andere (einfachere / anschaulichere) Darstellung verwendet
- IC 74LS151 enthält 1-aus-8 Multiplexer
- VHDL-Implementierung als Case-Anweisung

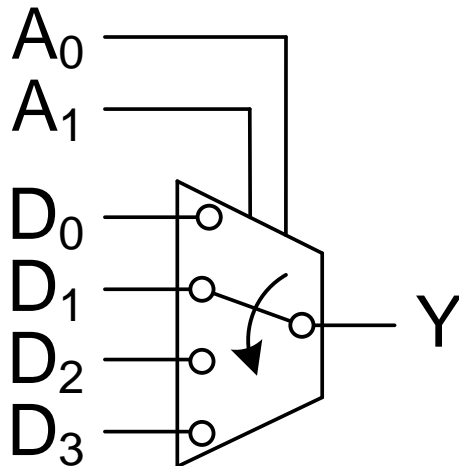


Demultiplexer

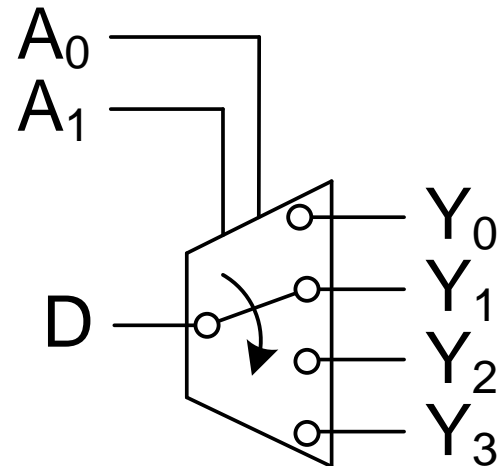
- Inverse Operation des Multiplexers
- 1-auf-4-Demultiplexer
 - Eingang D wird auf wählbaren Ausgang geschaltet
 - Die anderen Ausgänge sind ,0'

A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Multiplexer



Demultiplexer



Adressdecoder

- Bei einem Adressdecoder aktiviert eine Binärzahl mit n Stellen eine von 2^n Leitungen
- Eine Steuerleitung G aktiviert die Ausgänge

G	A ₂	A ₁	A ₀	Y ₇	Y ₆	Y ₅	Y ₄	Y ₃	Y ₂	Y ₁	Y ₀
0	-	-	-	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

- Die Schaltung entspricht exakt einem 1-auf-8 Demultiplexer des Datensignals G
 - Je nach Anwendungsgebiet ist die Bezeichnung als Adressdecoder verständlicher
- ➔ **Anwendung:** Adresse wählt einen von 8 Speicherbausteinen aus

Mehrstufige Logik

- Die gezeigten Schaltungen entsprechen meist einer **zweistufigen Logik**
- Auch Logikminimierung mit Karnaugh führt zu zweistufiger Logik
 - Inverter zur Erzeugung negierter Variablen zählen nicht als Logikstufe
- Für viele Anwendungen können kostengünstige Schaltungen entwickelt werden, wenn
 - mehr als zwei Logikstufen gewählt werden und
 - die Schaltung der Struktur der Aufgabe angepasst wird

Beispiel Addierer

Zwei Dualzahlen A und B, der Länge 8 bit, soll zu einer 9 bit Zahl S addiert werden

- Die Funktionstabelle hat 16 Eingänge, 9 Ausgänge und $2^{16}=65\,536$ Einträge
 - Schaltungserzeugung durch Karnaugh ist hierfür nicht praktikabel
- Eine sinnvolle Schaltung ergibt sich durch schrittweises Addieren der einzelnen Stellen
 - Diese Struktur entspricht dem Rechnen „von Hand“
- Die acht Stellen der Dualzahlen werden als A7, A6, ..., A0, bzw. B7, ..., B0 bezeichnet. A7 und B7 sind die höchstwertigsten Stellen

Algorithmus zur Addition von Dualzahlen

- Die beiden untersten Stellen A0 und B0 werden addiert; es ergibt sich ein **Ergebnis** S0 und ein **Übertrag** (engl. „**carry**“) in die nächste Stelle
- Für die Wertigkeit 1 werden die Stellen A1 und B1 mit dem Übertrag aus der vorherigen Stelle addiert und die Summe S1 und ein neuer Übertrag erzeugt
- Der Übertrag der letzten Wertigkeit 7 ergibt das höchstwertigste Summenbit S8

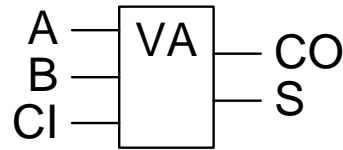
$$\begin{array}{rcccccccc} & A & & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ + & B & & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline & & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & \text{Übertrag} \\ S & & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

- Dieser Rechenalgorithmus kann in eine Addierer-Schaltung umgesetzt werden
- Es wird eine Teilschaltung, gebildet, die die Addition für eine Stelle vornimmt
- Diese Teilschaltung wird **Volladdierer (VA)** genannt, erhält **drei Eingangssignale**:
 - o Zwei Stellen der zu addierenden Zahlen: A, B
 - o Den Übertrag der vorherigen Stufe: CI (carry-in)und erzeugt **zwei Ausgangssignale**:
 - o Die Summe dieser Stelle: S
 - o Den Übertrag zur nächsten Stufe: CO (carry-out)

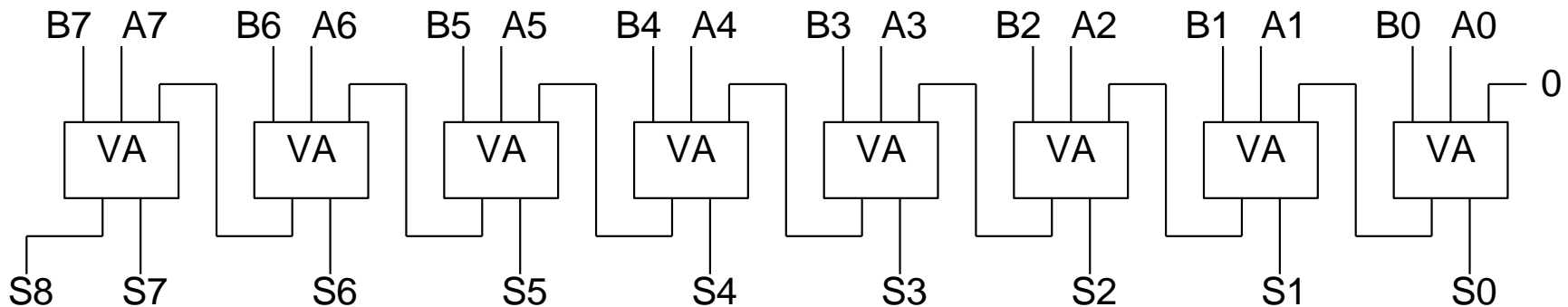


Ripple-Carry-Addierer

- Damit ergibt sich das Symbol und die Funktionstabelle eines Volladdierers
- Ein Volladdierer kann durch wenige Gatter implementiert werden
- Für die Addition zweier 8 bit Zahlen werden 8 Volladdierer benötigt
- Der Übertrag in die unterste Stelle ist Null



A	B	CI	CO	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



- Der Übertrag durchläuft alle Stellen. Darum bezeichnet man diesen Addierer als „**Ripple-Carry-Addierer**“ oder „**Ripple-Carry-Adder**“.

Ripple-Carry-Addierer (II)

Weitere Vereinfachung

- Die unterste Stufe hat immer den Eingang 0 und kann vereinfacht werden
- Ein Volladdierer ohne Carry-Eingang wird als **Halbaddierer** (HA) bezeichnet

Anwendung

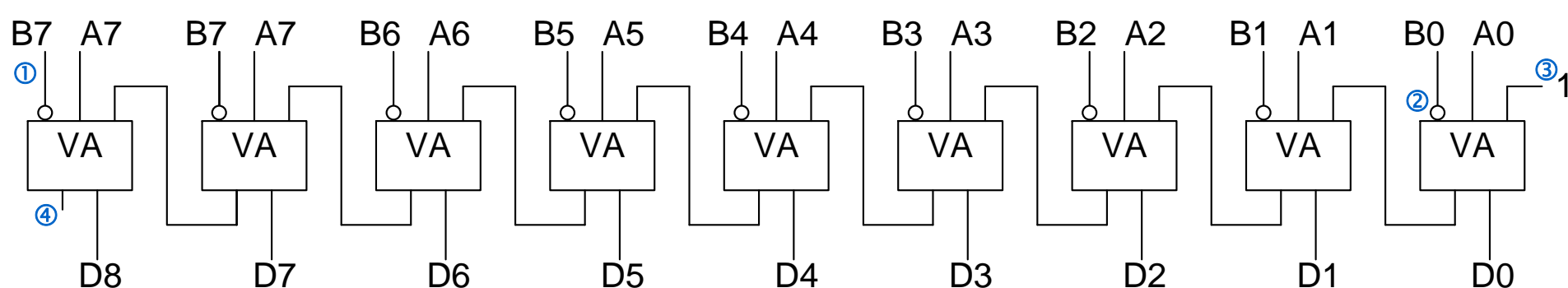
- Jede Logikstufe verursacht eine kurze Verzögerung des Signals
- Für die Addition kleiner Zahlen (z.B. 8 bit) ist der Ripple-Carry-Addierer meist schnell genug
- Für die Addition großer Zahlen (z.B. 32 bit) ist der Ripple-Carry-Addierer aber oft zu langsam
- Hierfür existieren andere Addierer-Strukturen, die mit etwas mehr Schaltungsaufwand deutlich schneller arbeiten

Weitere spezielle Schaltnetze:

- Für viele andere Aufgaben existieren spezielle Schaltnetze:
 - z.B.: Multiplikation, Division, Code-Umsetzung (z.B. Dualzahlen \leftrightarrow Gray-Code)

Subtraktion mit Ripple-Carry-Addierer

- Die Schaltungsstruktur des Ripple-Carry-Addierers kann auch zur Subtraktion $D = A - B$ verwendet werden
 - Annahme: Operanden 8bit 2er-Komplement, Ausgabe 9 bit 2er-Komplement
- Rechenweise:
 - Erweiterung der Operanden auf 9 Bit durch Kopie des MSB
 - Umwandlung des Operanden B in den negativen Wert durch Invertierung aller Bits und Addition von 1
 - Addition von 1 durch Carry-In am LSB
 - Übertrag aus vorderer Stelle entfällt
- Der Volladdierer mit invertiertem Eingang B kann ebenso wie „VA“ durch wenige Gatter implementiert werden

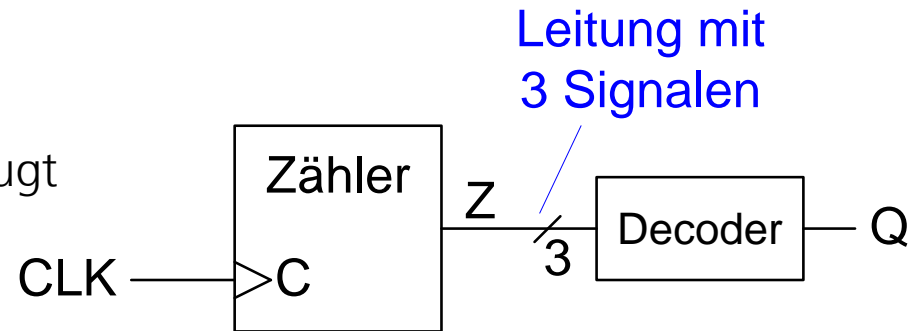


6.3 Spezielle sequentielle Schaltungen – Zähler

- **Zähler („Counter“)** sind ein wichtiges Grundelement für Schaltungen
- Manche Anwendungen benötigen direkt einen Zähler
- Andere Anwendungen können durch einen Zähler und eine einfache Ansteuer- und/oder Auswerteschaltung einfach implementiert werden

Beispiel:

- Jeden achten Takt soll ein Signal $Q=,1'$ und sonst $Q=,0'$ sein
- Q wird durch einen Automaten erzeugt
 - Zähler periodisch von 0 bis 7
 - Decoder gibt bei 7 eine ,1' aus



Die Grundfunktion des Zählers ist, für jeden Eingabeimpuls die Ausgabe auf einen nachfolgenden Wert zu setzen

- Der Eingabeimpuls ist üblicherweise der Takt
- Die Ausgabe ist üblicherweise eine Zahl

Steuersignale für Zähler

Die Grundfunktion kann durch verschiedenen Steuersignale erweitert werden

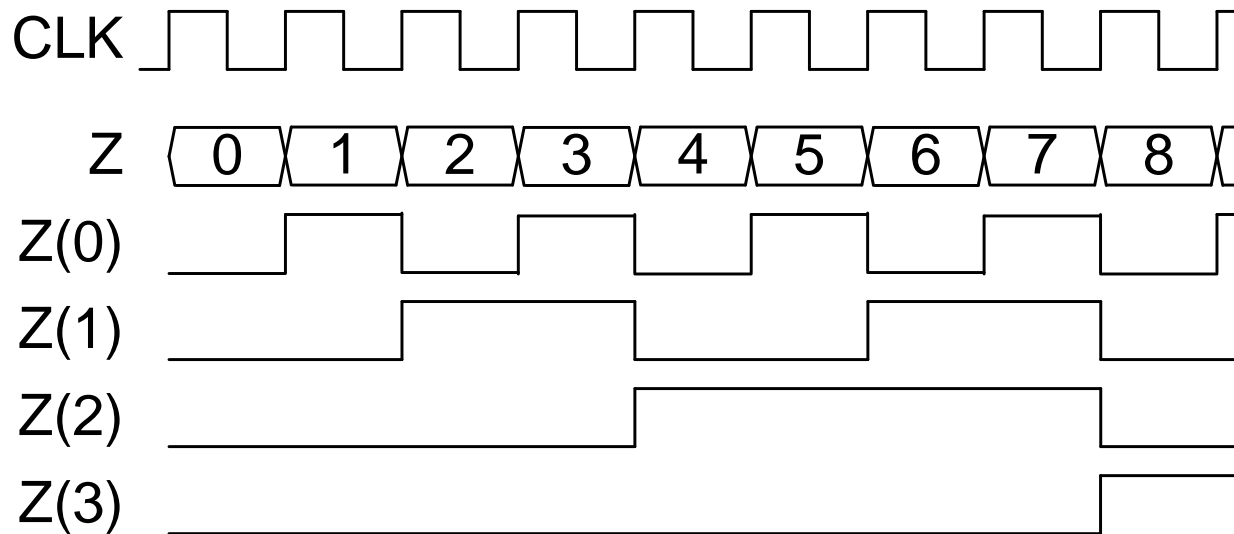
- **Enable:** Der Zähler geht nur zum nachfolgenden Wert, wenn der Steuereingang $\text{Enable} = 1$ ist
- **Clear:** Der Zähler springt wieder auf den Startzustand (auch als **Reset** bezeichnet)
- **Load:** Der Zähler springt auf einen vorgewählten Zustand oder lädt einen Zustand entsprechend einem weiteren Steuereingang
- **Up/Down:** Die Zählrichtung kann gewählt werden
 - Dieses Steuersignal wird oft angegeben als: $\text{Up}/\overline{\text{Down}}$
 - o Bei 1 ist $\text{Up} = 1$ und der Zähler zählt aufwärts
 - o Bei 0 ist wegen der Invertierung $\text{Down} = 1$, der Zähler zählt abwärts

Zählweise und Ausgabe

- Die Ausgabe von Zählern erfolgt meist als Dualzahl
- Die Zählrichtung kann vorwärts, rückwärts oder steuerbar sein
 - Vorwärtszähler sind anschaulicher
 - Rückwärtszähler erlauben eine einfache Erkennung des Endzustandes Null
- Gezählt wird stets von oder bis **Null** (also nicht von oder bis Eins)
- Die meisten Zähler beginnen nach Erreichen des letzten Ausgabewertes automatisch wieder beim ersten Ausgabewert
 - Man bezeichnet dies als „**Modulo-m Zähler**“, wobei m die Anzahl der Zustände ist
 - Ein Modulo-m Aufwärtszähler hat die Ausgabewerte 0 bis m-1
 - Beispiel: Ein Modulo-5 Aufwärtszähler zählt 0, 1, 2, 3, 4, 0, 1, 2, 3, 4, 0, 1, ...
- Besonders einfach sind „**Modulo-2^N Zähler**“
 - Modulo-2^N Zähler durchlaufen alle N-stelligen Dualzahlen

Zähler als Frequenzteiler

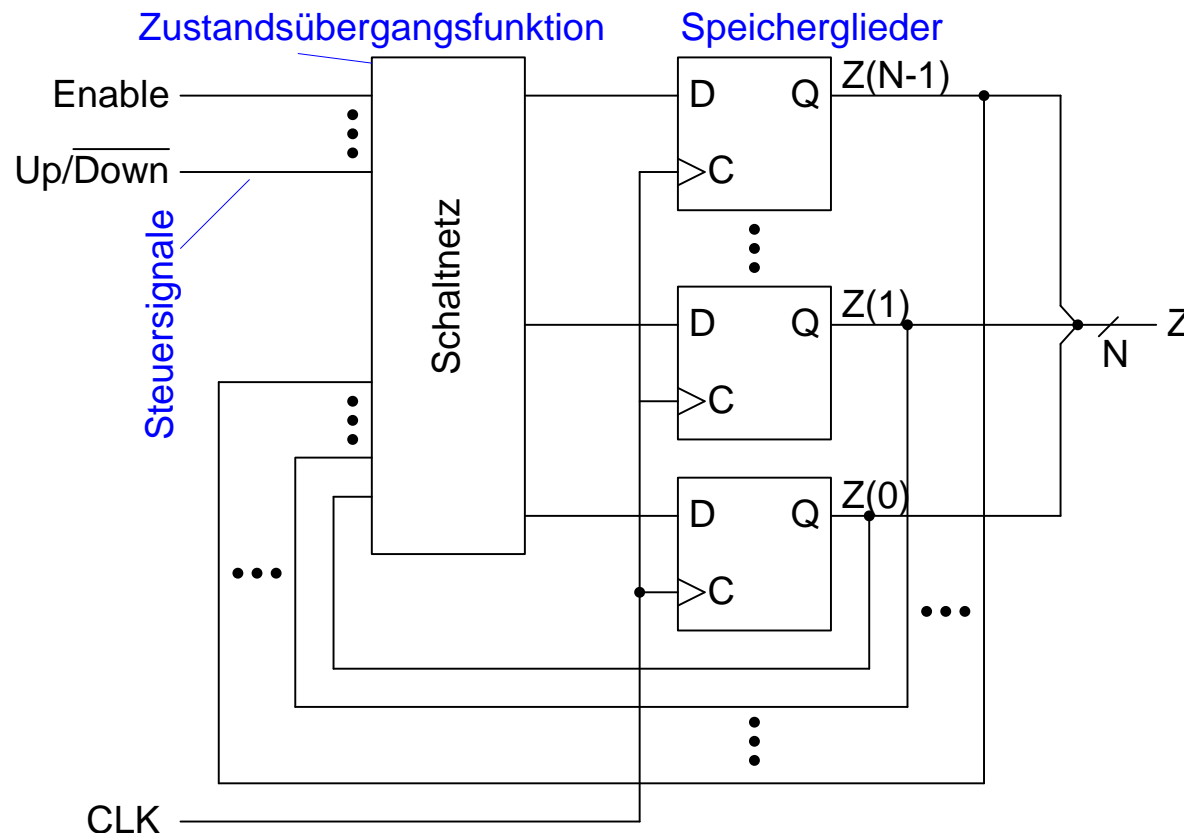
- Die einzelnen Stellen eines Modulo- 2^N Zählers stellen einen Frequenzteiler dar
 - Das unterste Bit (Bit 0) wechselt mit der halben Taktfrequenz des Eingangstaktes
 - Bit 1 wechselt einem Viertel der Eingangstaktfrequenz
 - Bit $N-1$ wechselt mit $CLK/2^N$, mit CLK = Eingangstaktfrequenz



- Aus einem vorhandenen Takt können mit einem Zähler einfach weitere Takte mit reduzierter Taktfrequenz erzeugt werden

Implementierung

- Ein Zähler wird als „normaler“ Automat implementiert
 - Speicherglieder zur Speicherung des aktuellen Zählerstandes
 - Schaltnetz zur Berechnung des nächsten Zählerstandes



Zähler mit Modulo ungleich 2^N

- Die Zählfunktion kann auch eine andere Periode als Modulo- 2^N haben
 - Beispiel: BCD-Zähler, als Dezimalzähler (BCD – Binary Coded Decimal)
- Im FPGA wird jede Funktion als LUT umgesetzt
 - Optimierung unnötig
 - ➔ Einfache Beschreibung als VHDL möglich

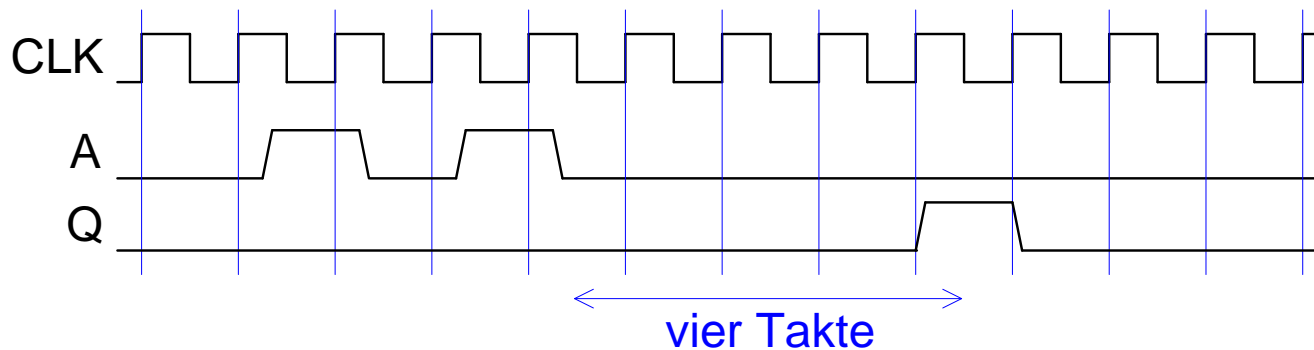
```
wait until rising_edge(clk);  
  
if (count >= "1001") then    -- value 9 or undefined  
    count <= "0000";  
else  
    count <= count + "0001";  
end if;
```

- Auch für ASIC-Implementierung ist eine einfache Beschreibung oft besser, als mit kryptischen Code einige Transistoren zu sparen

Automat mit Zählern

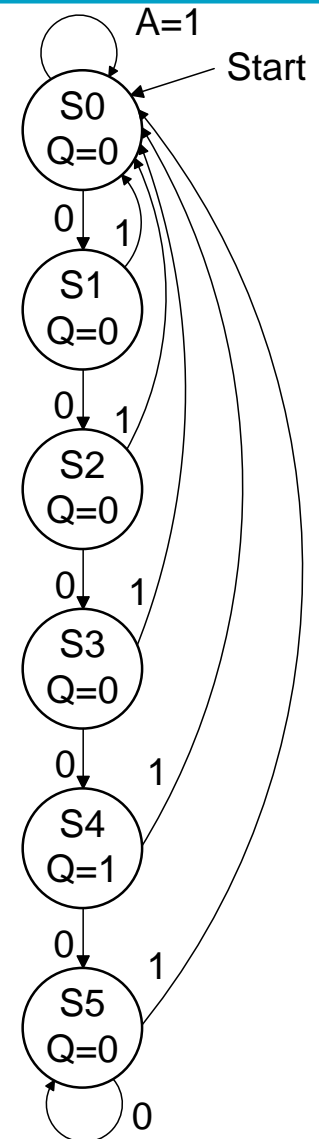
Beispiel:

Wenn ein Signal A von ,1' auf ,0' wechselt und vier Takte lang ,0' bleibt, soll für einen Takt der Ausgang Q auf ,1' gehen. Ansonsten ist Q gleich ,0'.



Das Zustandsfolgediagramm (Moore) hat sechs Zustände

- S0: Eingang ist ,1'
- S1: Eingang war 1-mal ,0'
- S2: Eingang war 2-mal ,0'
- S3: Eingang war 3-mal ,0'
- S4: Eingang war 4-mal ,0', Ausgabe von Q=,1'
- S5: Eingang war öfter als 4-mal ,0'

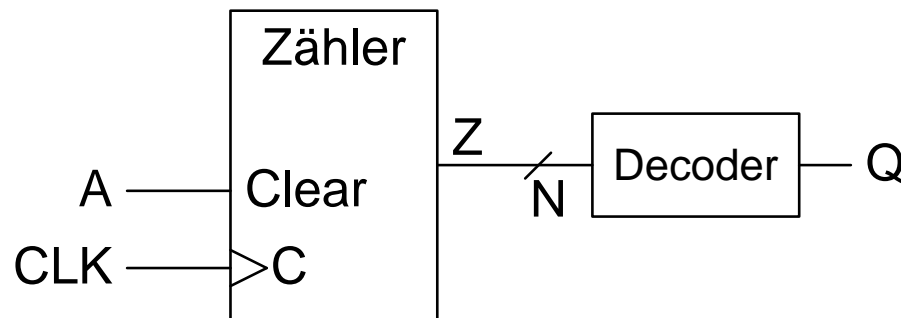


Automat mit Zählern (II)

- Bei einer Verzögerung von vier Takten kann ein Automat (relativ) einfach entwickelt werden
- Was ist, wenn die Verzögerung 40 oder 400 Takte betragen soll?

Lösung:

- Der Automat wird durch einen Zähler implementiert
- Der Eingang A setzt den Zähler auf den Startzustand
- Für $A=0'$ zählt der Zähler
- Wenn die gewünschte Verzögerung erreicht ist, setzt ein Decoder den Ausgang Q auf $1'$
- Der Zähler beginnt bei Erreichen des Endzustandes nicht neu



Automat mit Zählern (III)

```
architecture rtl of x_delay is
    signal count : integer range 0 to 63;
begin

process
begin
    wait until rising_edge(clk);

    if (a = '1') then                -- restart
        count <= 0;
    elsif (count = 41) then          -- limit
        count <= count;
    else                             -- increment
        count <= count + 1;
    end if;

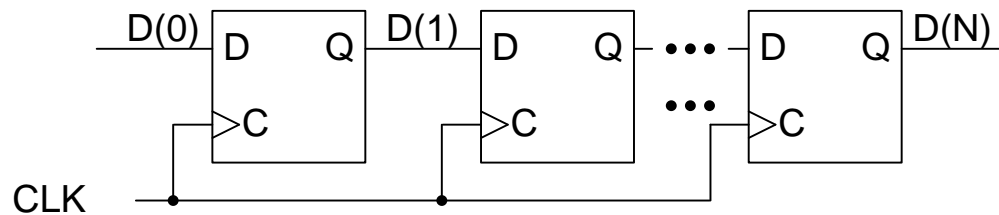
    q <= '0'; -- default
    if (count = 40) then
        q <= '1';
    end if;

end process;
end rtl;
```



Schieberegister

- Mehrere hintereinander geschaltete D-Flip-Flops werden als **Schieberegister** bezeichnet
- In einem Schieberegister werden die gespeicherten Werte mit jedem Takt einen Wert weiter geschoben



Anwendungen von Schieberegistern:

Verzögerung von Daten

- Wenn Daten vor oder innerhalb einer Verarbeitung um wenige Takte verzögert werden sollen, werden Schieberegister eingesetzt
- Bei größeren Verzögerungen (ab ca. 16 Takte) sind Speicher meist effizienter

Schieberegister (II)

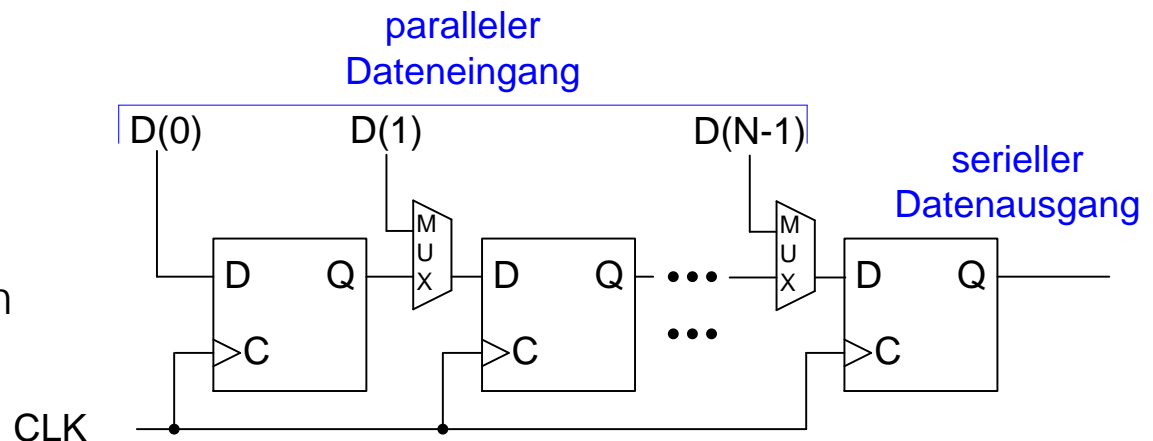
Weitere Anwendungen von Schieberegistern:

Seriell-Parallel-Wandlung

- Daten, die über eine serielle Leitung übertragen werden, können durch ein Schieberegister gespeichert werden
- Nachdem die komplette Information seriell im Schieberegister gespeichert ist, werden die Daten an den einzelnen Flip-Flops parallel gelesen

Parallel-Seriell-Wandlung

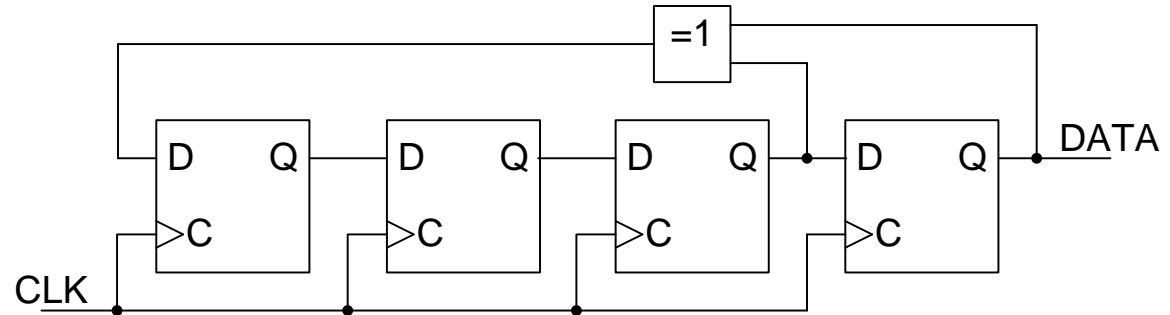
- Auch die umgekehrte Wandlung erfolgt über Schieberegister
- Daten werden parallel ins Schieberegister geschrieben und seriell gelesen
- Ein Steuersignal muss die Eingänge der Flip-Flops über einen Multiplexer umschalten



Rückgekoppeltes Schieberegister

- Bei einem **rückgekoppelten Schieberegister** werden einige Stellen XOR-verknüpft und wieder in das Schieberegister gegeben

- Englisch:
„**Linear Feedback Shift Register**“
oder „**LFSR**“



- Bei geeigneter Wahl der Rückkopplung werden bei einem N-bit-Schieberegister **$2^N - 1$ verschiedene Zustände** durchlaufen
 - Alle 2^N möglichen Kombinationen treten auf, ausgenommen alle Stellen auf ,0‘
 - Der Startwert darf nicht alle Stellen auf ,0‘ setzen
 - Die Abgriffe der Rückkopplung sind für verschiedene Längen des Schieberegisters in Tabellen angegeben
- Eingesetzt werden LFSR z.B. für Pseudo-Zufallszahlen und als Zahlengeneratoren im GPS

6.4 Zeitverhalten

- Logikgatter benötigen eine gewisse (kurze) **Laufzeit** bis der Ausgang auf eine Änderung der Eingangsvariablen reagiert
- Die Laufzeit ist abhängig von der Technologie:
 - Als einzelnes Gatter in einem Gehäuse kann die Laufzeit über 10 ns betragen
 - Als Teil eines hochintegrierten ASICs sind Laufzeit unter 0,1 ns möglich

Frage:

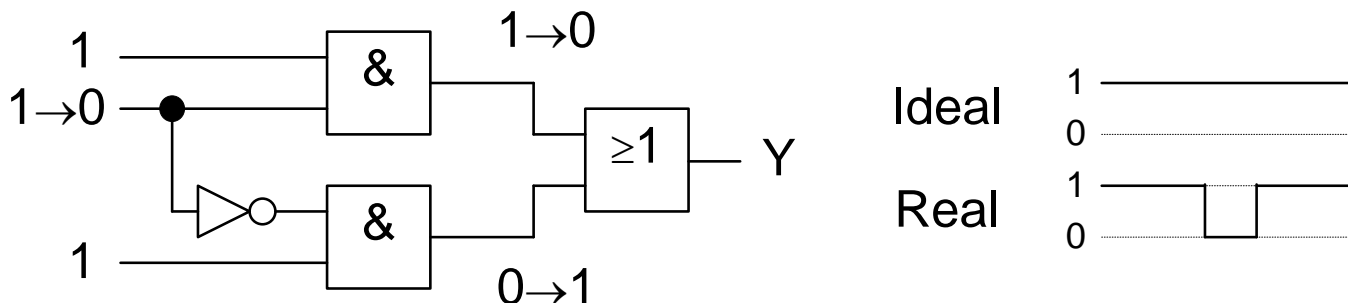
- Welche Taktfrequenzen sind mit einem 8-bit und 32-bit Ripple-Carry-Addierer möglich?
- Als Annahme soll ein Volladdierer eine zweistufige Logik enthalten und ein Logikgatter eine Verzögerung von 1 ns besitzen

Spikes

- Selbst gleichartige Gatter können eine unterschiedliche Laufzeit haben
- Beim Wechsel einer oder mehrerer Eingangsvariable treten in Schaltnetzen daher oft kurze Zwischenzustände auf
 - Diese werden als **Spike** (auch **Glitch** oder **Hazard**) bezeichnet

Beispiel:

- Bei der Schaltung rechts wechselt ein Eingang von 1 auf 0
- Die Ausgänge beider UND-Gatter wechseln dadurch ebenfalls ihren Wert
- Das untere UND-Gatter ist durch den vorgeschalteten Inverter etwas langsamer als das obere UND-Gatter
- Dadurch liegt an beiden Eingängen des ODER-Gatters kurzzeitig 0 an und der Ausgang ist ebenfalls kurzzeitig 0

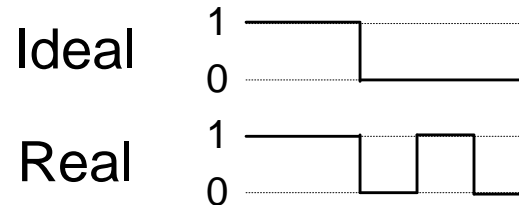
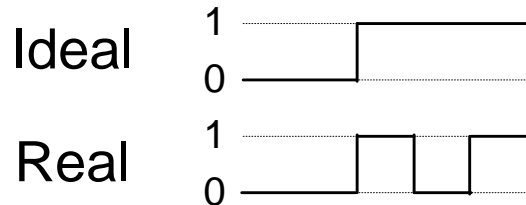


Spikes (II)

- Ein Spike kann ebenso auftreten, wenn der Ausgang der idealen Schaltung 0 bleiben würde



- Auch beim Wechsel des Ausgangswertes können durch unterschiedliche Laufzeiten Spikes auftreten



- Bei komplexen Schaltung, wie einem Ripple-Carry-Addierer können mehrere Übergänge auftreten, bis der endgültige Ausgangswert erreicht ist

Beispiel: Zeitverhalten eines Addierers

- Das Zeitverhalten eines Addierers kann mit einem Simulator dargestellt werden
 - Erläuterung später in Kapitel 7 und 8
- Addition wird durch „+“-Zeichen beschrieben
 - Erweiterung der Operanden auf 9 bit durch Concatenation-Operator „&“
 - „&“ ist kein logisches UND, sondern fügt Vektoren zusammen
 - Hier Konstante ‚0‘ (1 bit) und Eingang (8 bit)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity adder_8 is
    port ( a, b : in  unsigned(7 downto 0);
          s      : out unsigned (8 downto 0));
end adder_8;

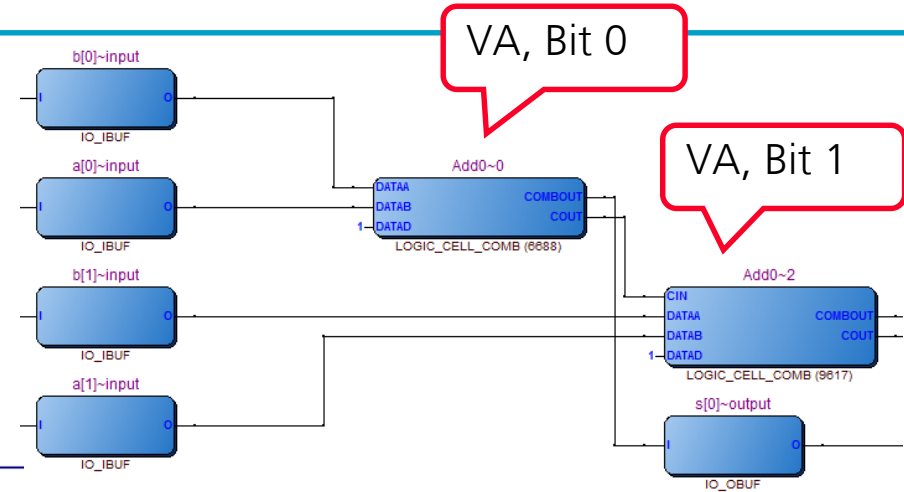
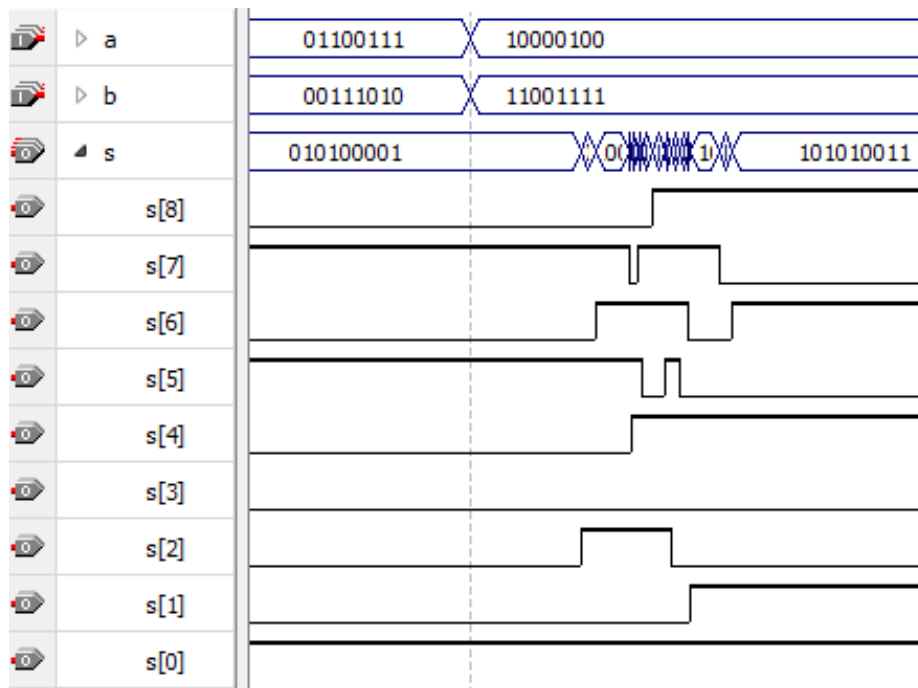
architecture behave of adder_8 is
begin
    s <= ('0' & a) + ('0' & b);
end behave;
```

Packages zur Definition der Datentypen und der arithmetischen Funktionen zur Addition

Erweiterung der Operanden auf 9 bit und Addition

Beispiel: Simulation des Addierers

- EDA-Tool setzt Beschreibung in Ripple-Carry-Struktur um (rechts)
- Simulation zeigt Zeitverhalten des Ausgangs (unten)
 - In Simulator „Timing“ wählen

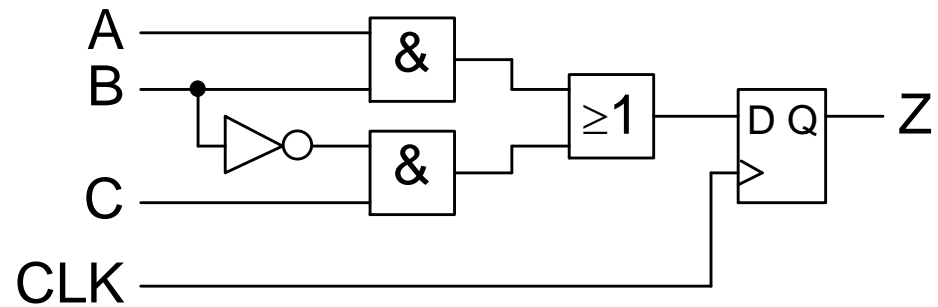


Screenshots: Altera-Tools
(zur Darstellung leicht nachbearbeitet)

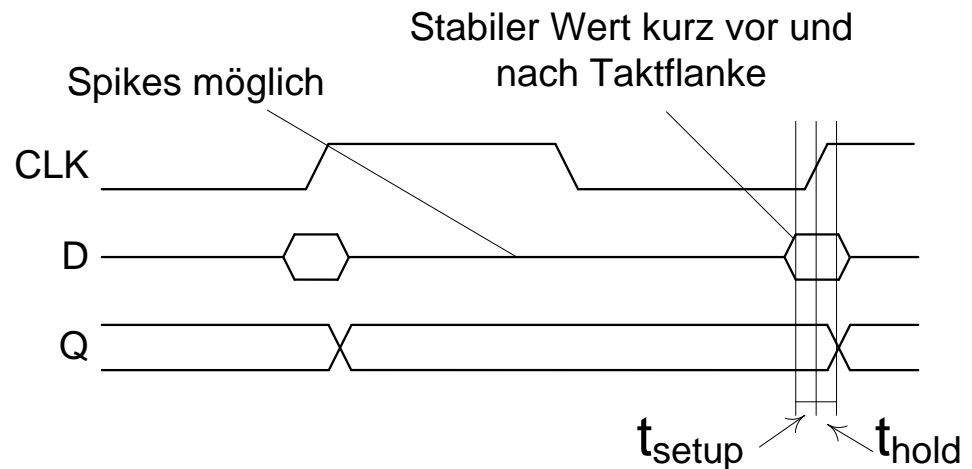
6.5 Taktkonzept und Taktübergänge

Speicherung durch Flip-Flops zur Vermeidung von Spikes

- Erst nachdem sich der stabile Zustand eingestellt hat, wird das Ergebnis von einem getakteten Flip-Flop gespeichert
→ Übliche Vorgehensweise

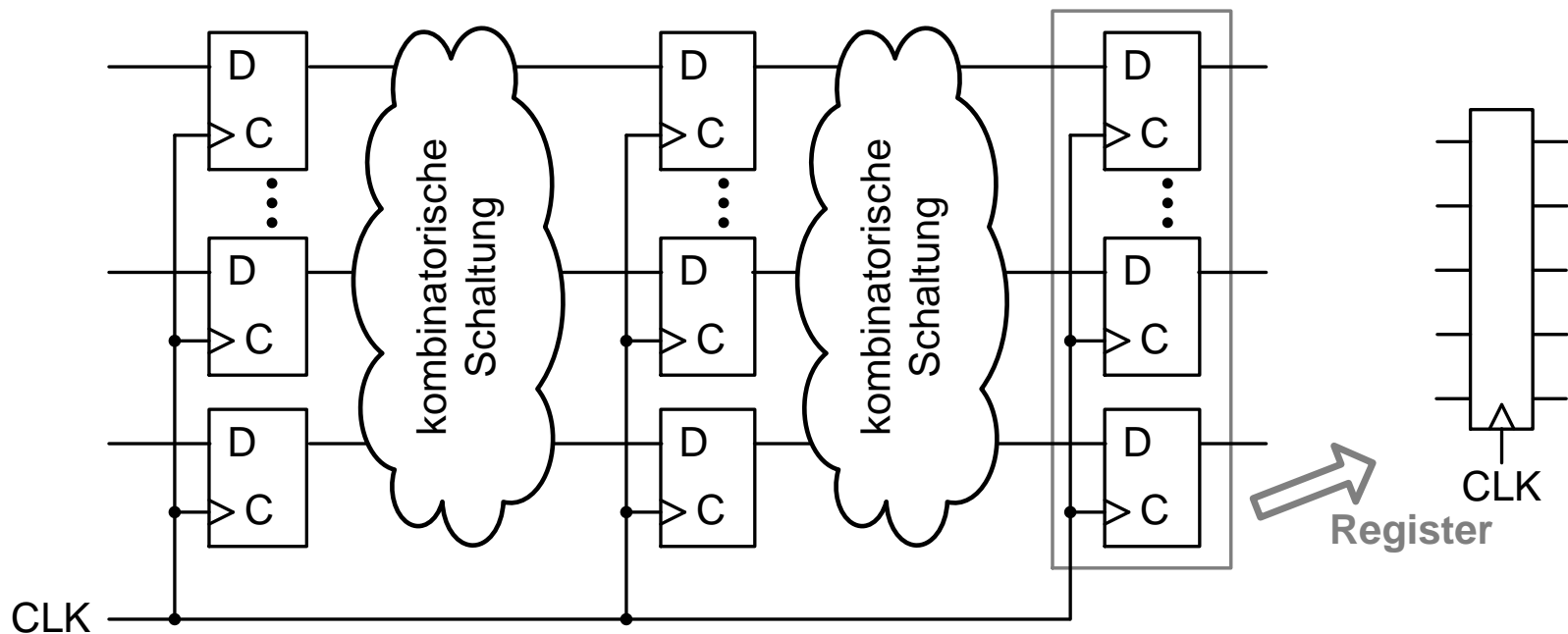


- Flip-Flops übernehmen die Eingangssignale bei der Taktflanke
 - Änderungen der Eingangssignale vor oder nach der Taktflanken werden nicht berücksichtigt
- Bei realen Flip-Flops dürfen sich die Steuersignale „kurz vor“ und „kurz nach“ der Taktflanke nicht ändern
- Diese Zeiten werden als **Setup-** und **Hold-Zeit** bezeichnet



Taktkonzept Register-Transfer-Level (RTL)

- Um ein „sauberes“ Arbeiten einer Schaltung sicherzustellen, werden alle Signale schrittweise in Flip-Flop-Stufen gespeichert
 - Diese Flip-Flop-Stufen werden als **Register** bezeichnet
- Gute Übersichtlichkeit bei diesem **Register-Transfer-Level** Schaltungskonzept
 - Klares Konzept, welche Informationen in einer Registerstufe vorhanden sind und was im nächsten Schritt mit diesen Informationen passieren soll.



Beispiel für RTL-Entwurf: Ampelsteuerung

- Aufgabe: Einfache Fußgängerampel mit fester Signalreihenfolge
- Eingangstakt 50 MHz

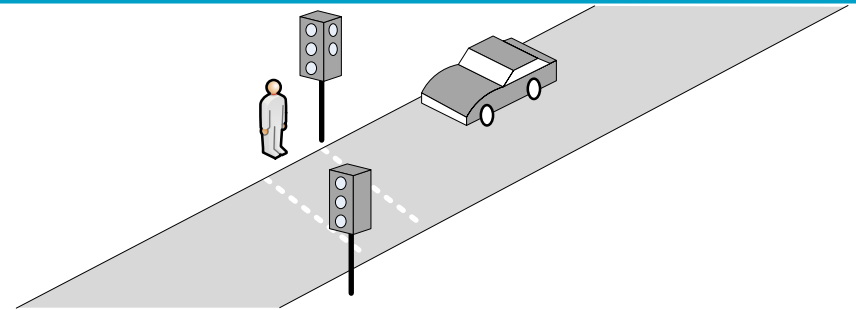
Aufteilung in drei Teilschritte:

- Aus dem Takt (50 MHz) wird ein Sekundensignal erzeugt
- Mit dem Sekundensignal werden 20 Schritte für den Ampelablauf gezählt
- Mit der Information, welcher Schritt des Ampelablaufs vorliegt, werden die Lichtsignale ausgegeben

Klarer Register-Inhalt der drei Teilschritte:

- Sekundensignal
 - Aktueller Schritt als Zahl von 0 bis 19
 - Lichtsignale der Ampel
-
- VHDL-Code im Lehrbuch und auf Webseite:

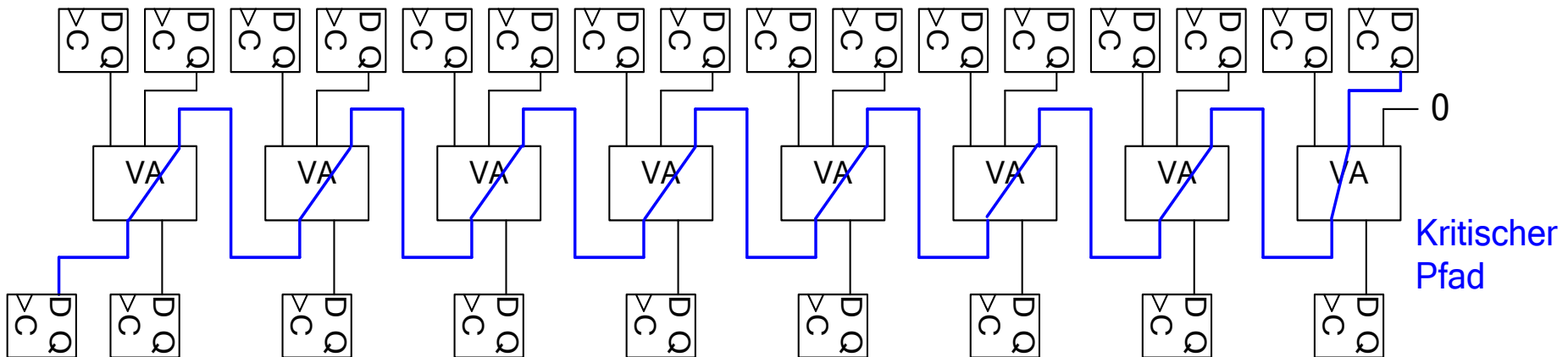
<http://www.hs-osnabrueck.de/buch-digitaltechnik>



Kritischer Pfad

- Das Schaltnetz zwischen den Flip-Flops muss schnell genug sein, in einem Taktzyklus wieder einen stabilen Zustand einzunehmen
 - Bei einer Taktflanke ändern sich die Eingangssignale des Schaltnetzes
 - Vor der nächsten Taktflanke müssen die korrekten Ausgangssignale vor den nächsten Flip-Flops anliegen
- Der langsamste Weg durch die Schaltfunktionen bestimmt die Geschwindigkeit der gesamten Schaltung
 - Er wird als **kritischer Pfad** bezeichnet

Beispiel: Ripple-Carry-Addierer (siehe auch oben)

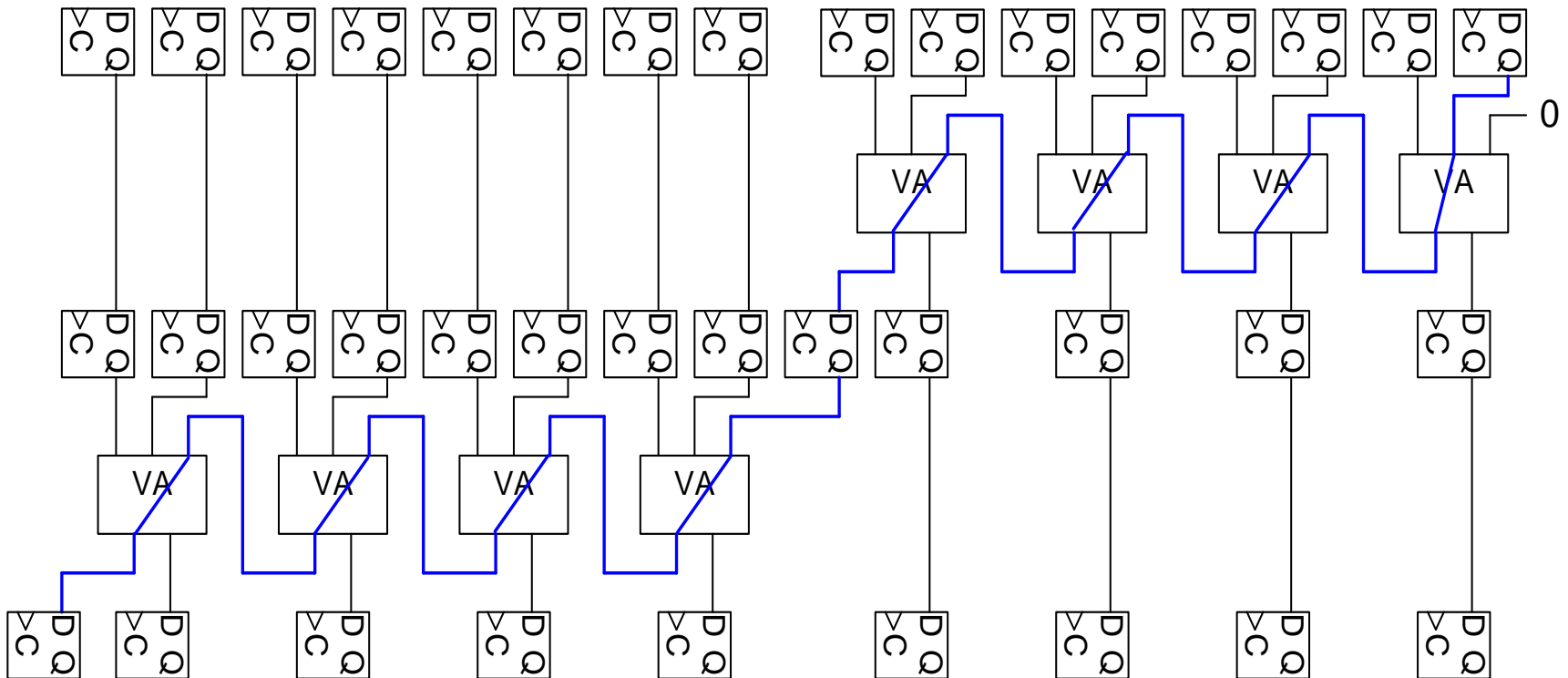


Pipelining

- Der kritische Pfad ist die Verzögerung von theoretisch
 - 8 Volladdierern
- Praktisch kommt hinzu
 - Setup-Time am Ende des Pfades
 - Clock-to-Output am Anfang des Pfades
 - Verdrahtung
- Das Einfügen von Flip-Flops beschleunigt die Verarbeitung in einer Schaltung
 - Der **kritische Pfad** wird verkürzt, somit kann die Schaltung mit höherer Taktfrequenz betrieben werden
 - Dies wird als „**Pipelining**“ bezeichnet
 - Sämtliche Eingangs- und Ausgangsvariablen müssen entsprechend verzögert werden

Pipelining eines Ripple-Carry-Addierers

- Pipeline-Stufe nach vier Volladdieren
- Verzögerungszeit des kritischen Pfades etwa halbiert
 - Durch Setup-Zeit der FFs etwas weniger als Halbierung der Verzögerungszeit



Latenzzeit

- Durch Pipelining ist ein Ergebnis erst nach mehreren Takten verfügbar
 - Diese Verzögerung wird als „**Latenzzeit**“ bezeichnet
 - Die Verarbeitungsleistung wird jedoch erhöht
 - Während einer Berechnung in der zweiten **Pipelinestufe**, kann der nächste Wert bereits in die erste Pipelinestufe gegeben werden
- ➔ Höhere Rechenleistung, „**Durchsatz**“, durch höheren Schaltungsaufwand
- Bereits beim Entwurf einer Schaltung können mehrere Flip-Flop-Stufen vorgesehen werden
- Dies wird z.B. in einer CPU benutzt
 - ARM7-CPU: 3 Stufen für „fetch“, „decode“, „execute“
 - Cortex-A8: 13 Pipeline-Stufen
 - Pentium 4: 31 Pipeline-Stufen
 - Intel Core i7: 14 Pipeline-Stufen
- Geschwindigkeitsverluste, falls nächste Verarbeitungsschritte noch nicht feststehen, z.B. bei einem Sprung („branch“)

6.6 Berechnung des kritischen Pfads

- Der kritische Pfad wird durch das EDA-Tool ermittelt
 - EDA – Electronic Design Automation
- Dabei wird der „worst case“ der Schaltungsparameter berücksichtigt
 - Abkürzung PVT
 - P – Prozess, also Fabrikationsprozess an der Grenze der Toleranzen
 - V – Voltage, also niedrigste spezifizierte Spannung
 - T – Temperature, also höchste spezifizierte Temperatur
- Start- und Endpunkt der berechneten Pfade sind üblicherweise Flip-Flops
 - An Ein- und Ausgängen von Schaltungen sollten Flip-Flops sein
 - Alle Flip-Flops sollten mit dem gleichen Takt arbeiten
- Die Zeitvorgabe ist ein „Constraint“, genau wie Position der FPGA-Pins
 - Wenn eine Zielvorgabe gesetzt ist, wird das Timing überprüft

Video zur Timing-Constraints mit Beispiel eines Timing-Fehlers:

<https://youtu.be/J7lu2456N6g>

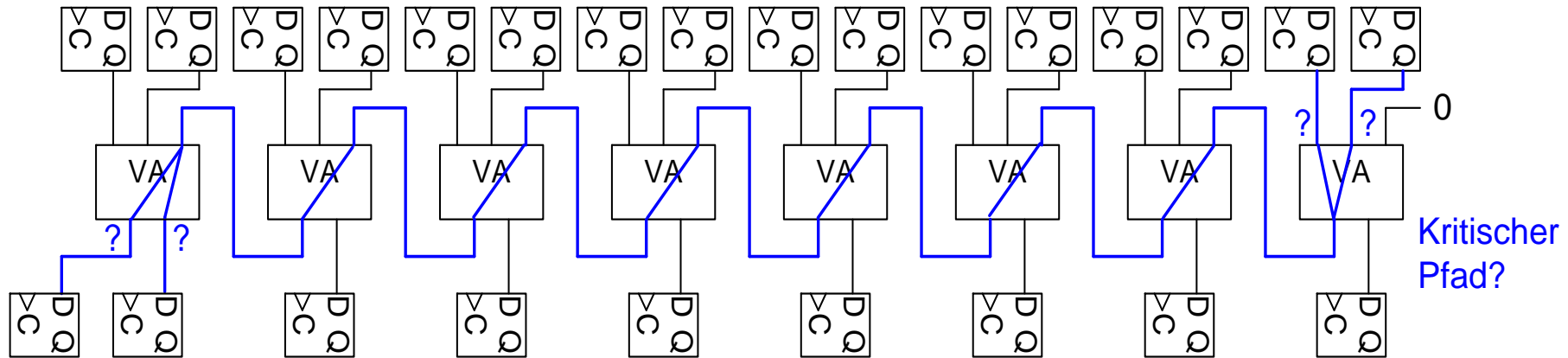
Berechnung des kritischen Pfads (II)

- Alle Wege von Quell-Flip-Flops zu Ziel-Flip-Flops werden analysiert
- Der langsamste Pfad ist der **kritische Pfad**
- Die Verzögerungszeit eines Pfades ergibt sich zu
 - Verzögerungszeit „Clock to Output“ des Quell-Flip-Flops
 - **„Interconnection Delay“** zum ersten Logikblock
 - **„Logic Delay“** des ersten Logikblocks
 - „Interconnection Delay“ zum zweiten Logikblock
 - „Logic Delay“ des zweiten Logikblocks
 - ...
 - „Logic Delay“ des letzten Logikblocks
 - „Interconnection Delay“ zum Ziel-Flip-Flop
 - **„Setup Time“** des Ziel-Flip-Flops

Welchen Einfluss hat die „Hold-Time“ der Flip-Flops?

Kritischer Pfad des Ripple-Carry-Addierers (8 Bit)

- Wie viele mögliche Timing-Pfade gibt es?
 - 16 Quell-FFs
 - 9 Ziel-FFs
 - Nicht alle Kombinationen sind verbunden

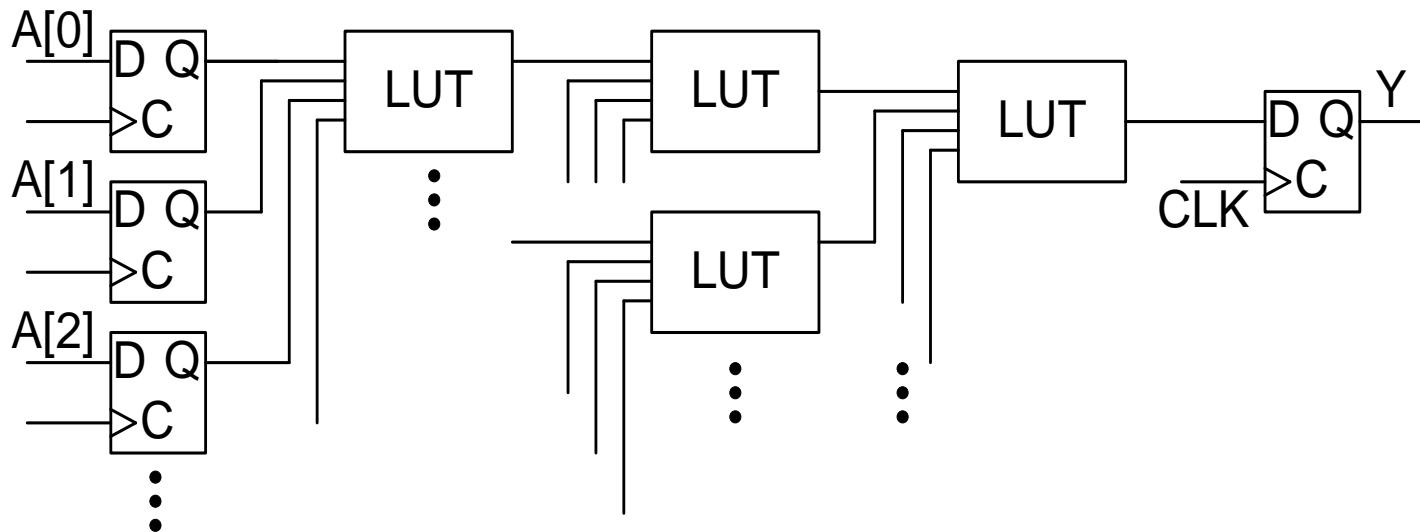


Aber:

- Der langsamste Pfad hängt auch von der Platzierung der Elemente ab
- Bei ungünstigem Layout kann auch ein anderer Pfad kritisch sein

Beispiel: 64-bit-ODER im FPGA

- Testschaltung mit drei Stufen von LUTs (Look-Up Table)
 - LUTs mit vier Eingängen
- Eingang A (64 bit) und Ausgang Y in FFs gespeichert



- Wie viele FFs und LUTs werden benötigt?

VHDL-Code für 64-bit-ODER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity or_64 is
    port ( clk : in  std_logic;
          a   : in  std_logic_vector(63 downto 0);
          y   : out std_logic);
end or_64;

architecture rtl of or_64 is
    signal a_q, zero : std_logic_vector(63 downto 0);
begin
    zero <= (others => '0');

    process
    begin
        wait until rising_edge(clk);

        a_q <= a;                -- input flip-flops

        if (a_q = zero) then -- or-function with FF
            y <= '0';
        else
            y <= '1';
        end if;
    end process;
end rtl;
```

Anmerkung:

Dieses Beispiel
verwendet die
ältere Software-Version
Altera Quartus II (9.1)



Timing Analyse des 64-bit-ODER

- Frequenz von 333 MHz als Anforderung gesetzt
- Einige Pfade verfehlen die Anforderung, andere erreichen sie
 - **Slack** ist der Abstand zwischen gefordertem und erzieltm Timing

Screenshot:
Altera Quartus II (9.1)

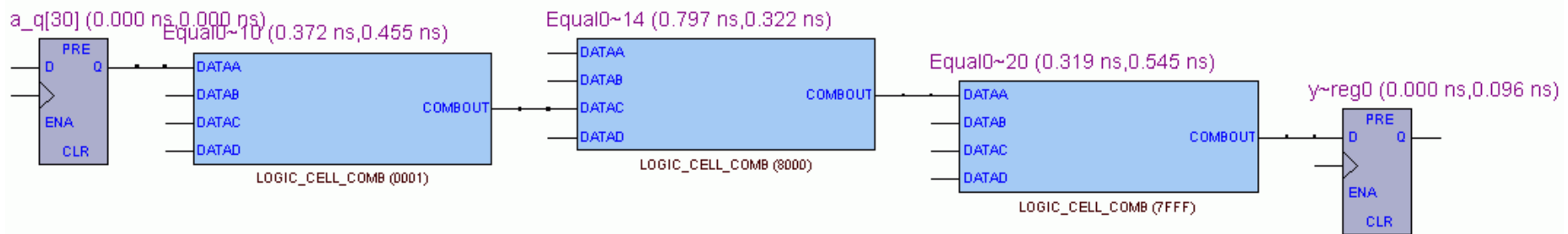
Device: Cyclone II
EP2C20F484C7

Compilation Report - Clock Setup: 'clk'							
Clock Setup: 'clk'							
	Slack	Actual fmax [period]	From	To	From Clock	To Clock	Required Setup Relationship
1	-0.143 ns	317.86 MHz (period = 3.146 ns)	a_q[30]	y~reg0	clk	clk	3.003 ns
2	-0.127 ns	319.49 MHz (period = 3.130 ns)	a_q[29]	y~reg0	clk	clk	3.003 ns
3	-0.052 ns	327.33 MHz (period = 3.055 ns)	a_q[50]	y~reg0	clk	clk	3.003 ns
4	-0.046 ns	327.98 MHz (period = 3.049 ns)	a_q[61]	y~reg0	clk	clk	3.003 ns
5	-0.031 ns	329.60 MHz (period = 3.034 ns)	a_q[18]	y~reg0	clk	clk	3.003 ns
6	-0.028 ns	329.92 MHz (period = 3.031 ns)	a_q[62]	y~reg0	clk	clk	3.003 ns
7	-0.025 ns	330.25 MHz (period = 3.028 ns)	a_q[49]	y~reg0	clk	clk	3.003 ns
8	-0.005 ns	332.45 MHz (period = 3.008 ns)	a_q[17]	y~reg0	clk	clk	3.003 ns
9	0.009 ns	334.00 MHz (period = 2.994 ns)	a_q[26]	y~reg0	clk	clk	3.003 ns
10	0.030 ns	336.36 MHz (period = 2.973 ns)	a_q[5]	y~reg0	clk	clk	3.003 ns
11	0.030 ns	336.36 MHz (period = 2.973 ns)	a_q[25]	y~reg0	clk	clk	3.003 ns
12	0.052 ns	338.87 MHz (period = 2.951 ns)	a_q[6]	y~reg0	clk	clk	3.003 ns
13	0.154 ns	351.00 MHz (period = 2.849 ns)	a_q[1]	y~reg0	clk	clk	3.003 ns
14	0.155 ns	351.12 MHz (period = 2.848 ns)	a_q[27]	y~reg0	clk	clk	3.003 ns
15	0.166 ns	352.49 MHz (period = 2.837 ns)	a_q[54]	y~reg0	clk	clk	3.003 ns
16	0.174 ns	353.48 MHz (period = 2.829 ns)	a_q[22]	y~reg0	clk	clk	3.003 ns
17	0.179 ns	354.11 MHz (period = 2.824 ns)	a_q[33]	y~reg0	clk	clk	3.003 ns

Timing Analyse des 64-bit-ODER (II)

Screenshot:
Altera Quartus II (9.1)

- Der kritische Pfad ergibt sich bei
 - Start bei a_q[30]
 - Interconnection Delay 0.372 ns und Cell Delay 0.455 ns für erste LUT
 - Interconnection Delay 0.797 ns und Cell Delay 0.322 ns für zweite LUT
 - Interconnection Delay 0.319 ns und Cell Delay 0.545 ns für dritte LUT
 - Interconnection Delay 0.000 ns und Cell Delay 0.096 ns für Ziel-FF
 - Endpunkt Y
- Aus dem Timing Report:
 - Clock to Output Delay des Quell-FFs: 0.277 ns
 - Setup Time des Ziel-FF: -0.038 ns



Timing Analyse des 64-bit-ODER (III)

Angaben aus Timing Report

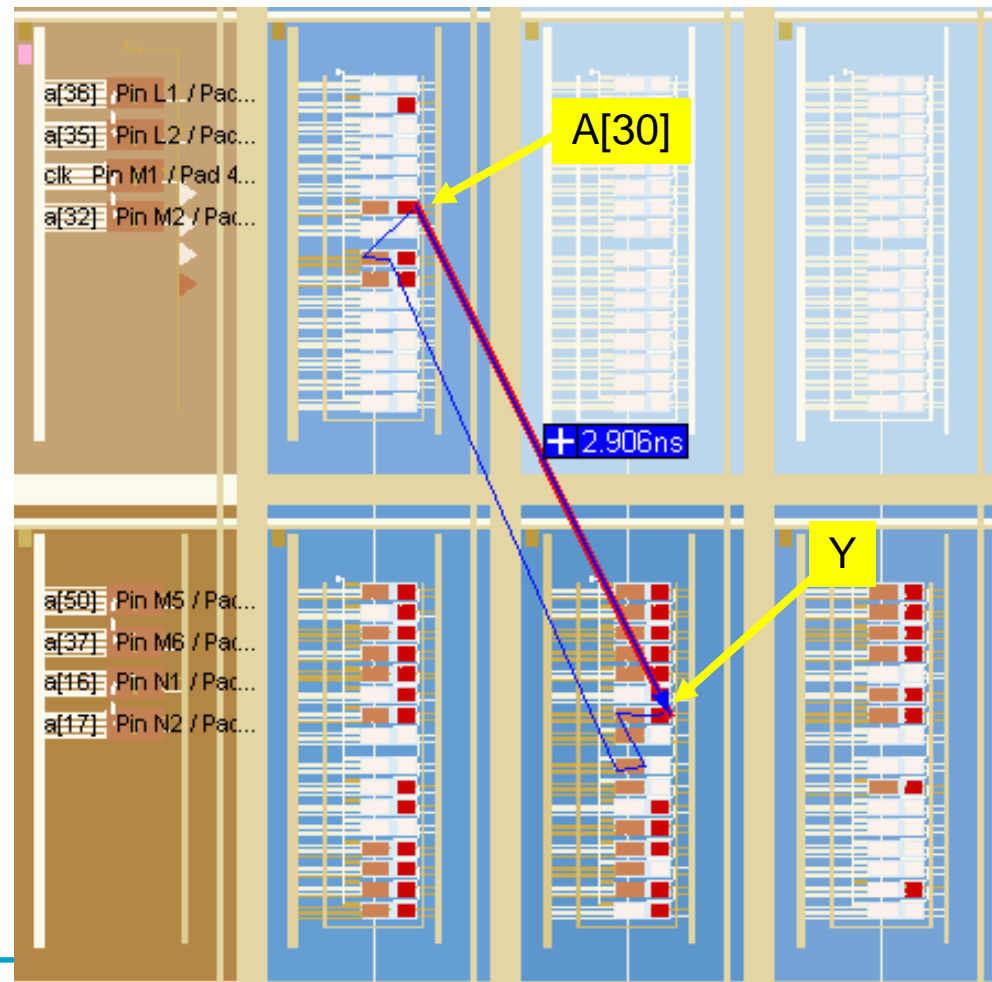
- Total interconnection delay: 1.488 ns
- Total cell delay: 1.418 ns
- **Sum: 2.906 ns**

Available time budget

- Clock period: 3.003 ns
- Clock to output of source FF: 0.277 ns
- Setup of destination FF: - 0.038 ns
- Clock jitter: 0.001 ns
- **Sum: 2.763 ns**
- **Slack: - 0.143 ns**

Grafische Darstellung des kritischen Pfads

Screenshot:
Altera Quartus II (9.1)



6.7 Übergang zwischen Takten

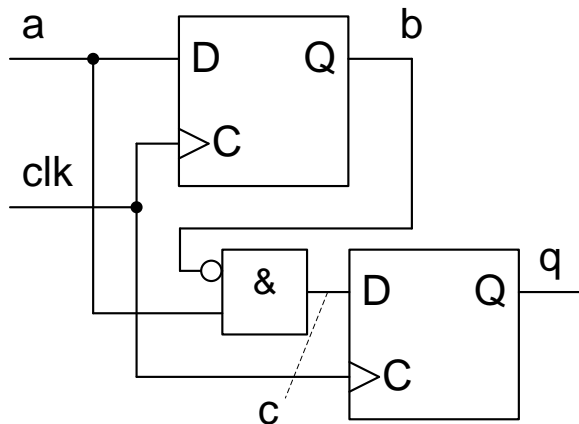
- Setup- und Hold-Zeiten lassen sich nur sicher berechnen, wenn Flip-Flops mit dem gleichen Takt betrieben werden
 - Der gleiche „Taktbereich“ wird als **Clock-Domain** bezeichnet
- Wenn irgendwie möglich sollte stets der gleiche Takt, also eine Clock-Domain, benutzt werden
 - Dies ist jedoch nicht immer möglich, z.B. in einem PC
 - CPU-Takt
 - Takt für DRAM-Speicher
 - Takt der Grafikkarte
 - ...
- Beim Übergang zwischen Clock-Domains kann eine fehlerhafte Datenübernahme auftreten

Ein Datenbit darf beim Taktübergang nicht in mehreren Flip-Flops gespeichert werden.

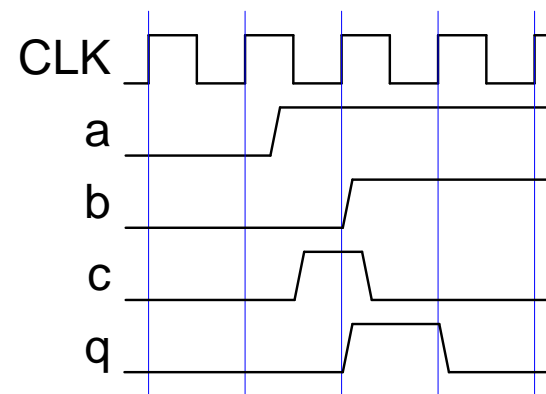
Fehlerhafte Datenübernahme bei Taktübergang

Beispiel: Flankenerkennung

- Das Signal ,a' kommt aus einer anderen Clock-Domain
- Mit dem Takt ,clk' soll ein Wechsel von ,0' nach ,1' erkannt werden
 - Bei einem Wechsel soll der Ausgang ,q' für einen Takt auf ,1' gehen
 - Dies wird als **Flankenerkennung** bezeichnet
- Funktionsweise der Schaltung:
 - Der vorherige Wert von ,a' wird als Wert ,b' gespeichert
 - Wenn ,a' (aktueller Wert) gleich ,1' und ,b' (vorheriger Wert) gleich ,0' ist, wird eine Flanke erkannt



Geplanter Ablauf



Fehlerhafte Datenübernahme bei Taktübergang (II)

Problem bei Datenübernahme

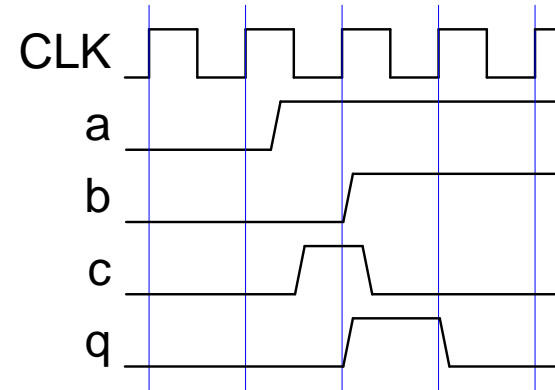
- Es ist nicht sichergestellt werden, dass bei einem Wechsel des Eingangs alle Flip-Flops die Information zum gleichen Zeitpunkt übernehmen

Fehlerablauf

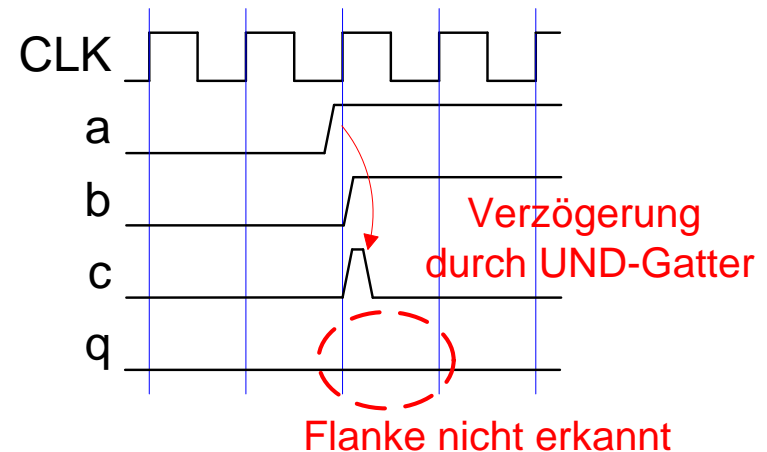
- ,a' wechselt kurz vor dem Takt von ,0' auf ,1'
- ,c' erkennt mit kurzer Verzögerung die steigende Flanke (da a=,1' und b=,0')
- FF ,b' übernimmt den neuen Wert
- FF ,q' übernimmt den neuen Wert **nicht**, wegen Verzögerung des Signals ,c'
- Im nächsten Takt ist ,b' schon ,1'
 - ➔ **Fehler:** Die Flanke wird nicht erkannt

Achtung: Der Fehler tritt nur sporadisch auf, nämlich wenn ,a' sich kurz vorm Takt ändert

Geplanter Ablauf



Ablauf im Fehlerfall

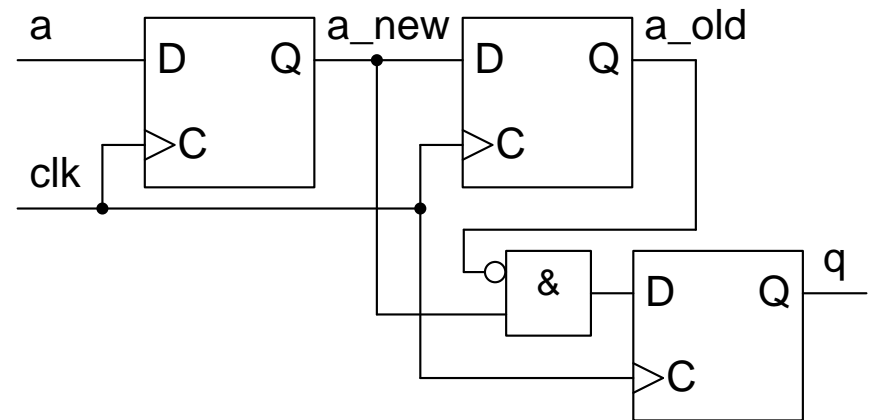


Sichere Datenübernahme bei Taktübergang

- Zur Fehlervermeidung darf das Eingangssignal beim Taktübergang nur in einem Flip-Flop gespeichert werden
- Umsetzung:
 - Der Eingang A wird zunächst mit dem Takt übernommen
 - Der getaktete Eingangswert wird mit dem vorherigen Wert verglichen

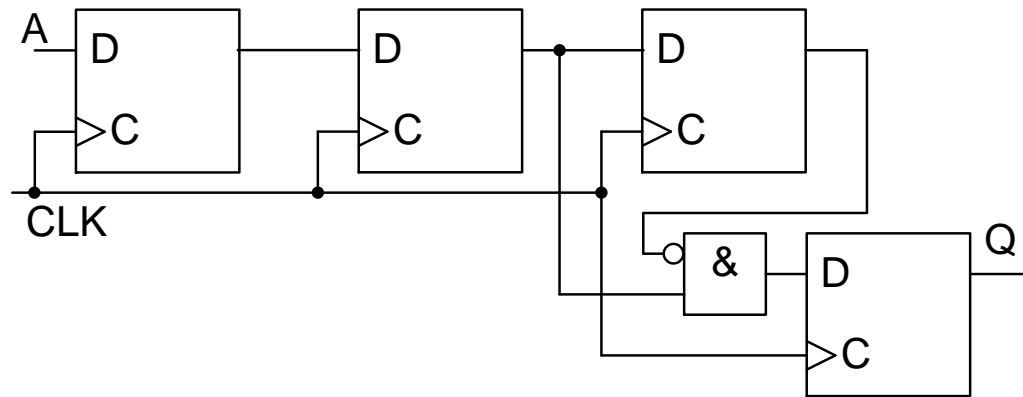
```
process
begin
    wait until rising_edge(clk);

    a_old <= a_new;
    a_new <= a;
    if ( (a_old='0') and
        (a_new='1') ) then
        q <= '1';
    else
        q <= '0';
    end if;
end process;
```



Metastabilität

- Weiteres Problem bei Taktübernahme ist die Einhaltung der Setup- und Hold-Zeiten
- Es kann der Fall eintreten, dass ein Flip-Flop in der Mitte zwischen 0 und 1 „hängt“
- Dieser Zwischenzustand wird als **Metastabilität** bezeichnet
 - Tritt selten auf, kann aber einen Fehler verursachen
- Schutz gegen Metastabilität durch zwei hintereinandergeschaltete Flip-Flops
 - Erst nach dem zweiten Flip-Flop wird das Signal im Taktbereich verwendet

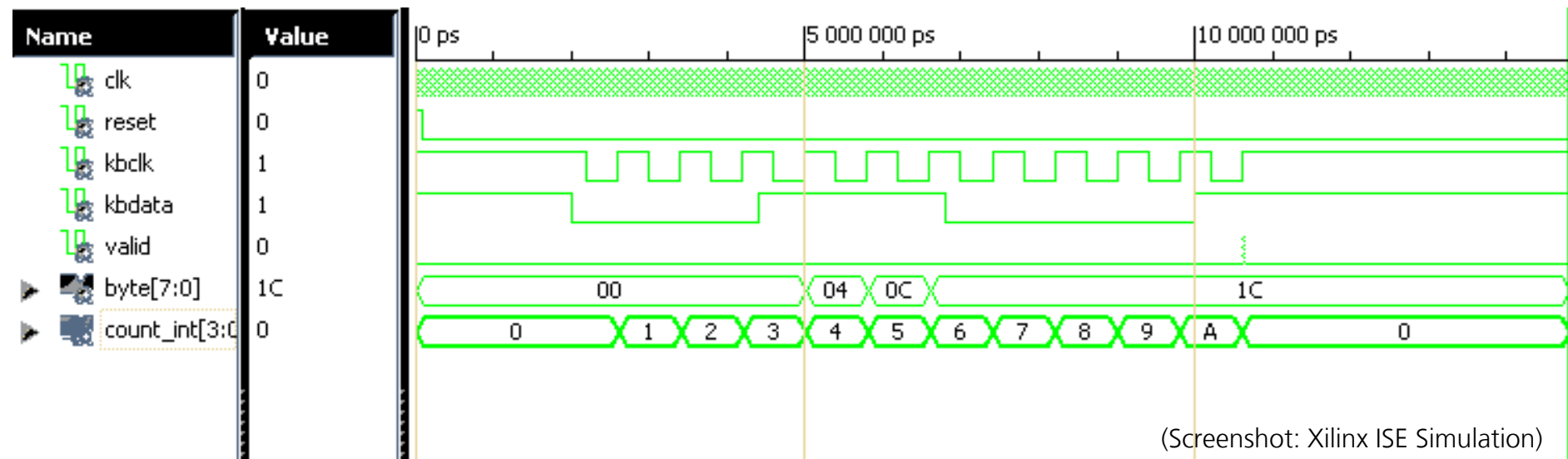


Nachteil:

- Das zweite Flip-Flop erhöht die Latenzzeit
- Synchronisation gegen Metastabilität wird darum nicht immer eingesetzt

Reales Beispiel für Fehler bei Taktübergang

- Decodierung des PS2-Datenstroms einer PC-Tastatur
- Funktionsweise (siehe Simulation)
 - Tastatur liefert Takt und Daten
 - 11 Takte für ein Byte (8 Bit Information plus Startbit, Stopbit, Parity)
 - Schaltung zählt Taktflanken mit COUNT_INT
 - Nach 11 Taktflanken (Zähler auf 0xA) liegt Codewort 0x1C vor und wird mit VALID angezeigt



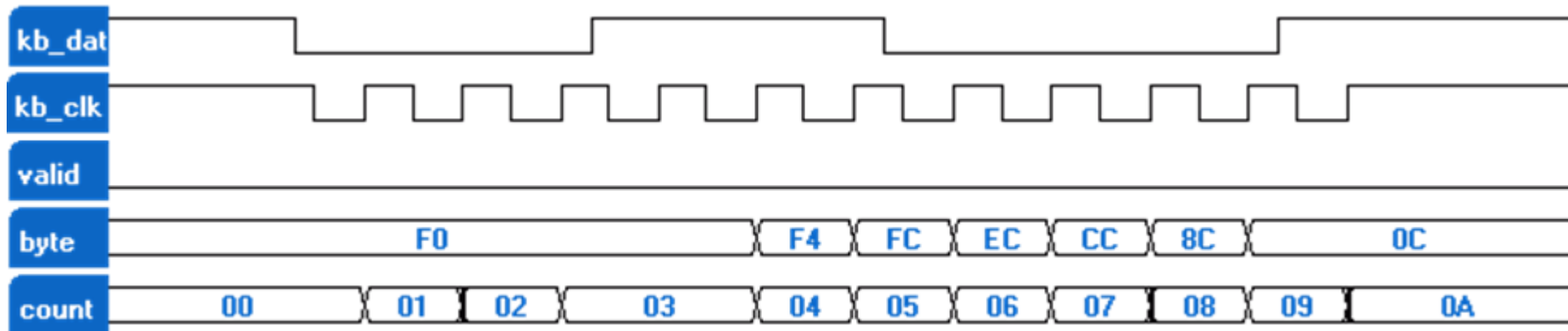
Reales Beispiel für Fehler bei Taktübergang (II)

Umsetzung auf FPGA

- Schaltung reagiert, aber nicht korrekt und mit stets anderen Ergebnissen

Debugging durch Analyse mit Logicanalyzer

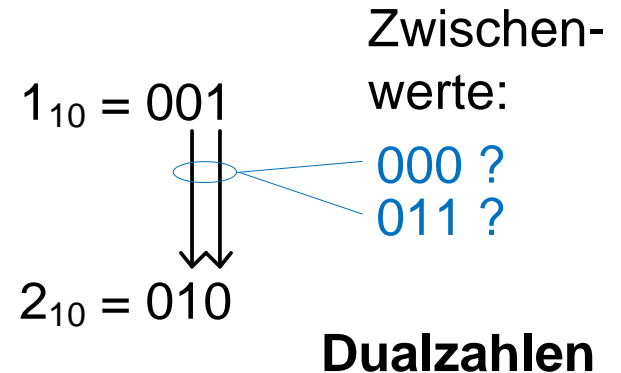
- Kopie von KB_CLK, KB_DATA und des internen Zählers COUNT_INT
→ Ergebnis: Zähler überspringt manchmal Flanke (hier bei **0x03**)
- Ursache: Fehlerhafte Flankenerkennung von KB_CLK



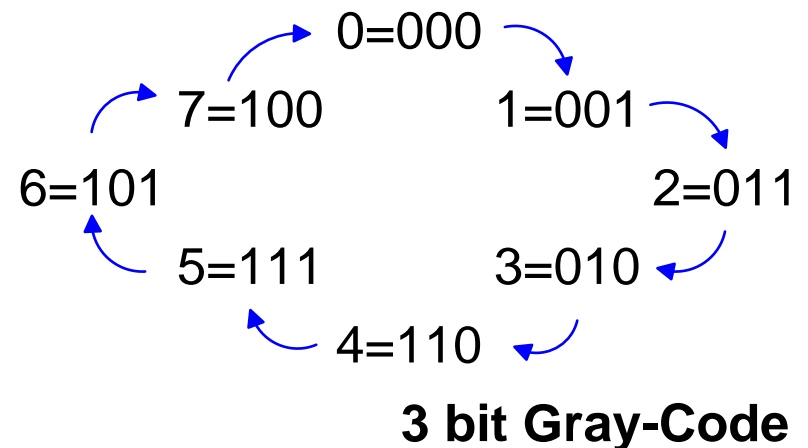
(Screenshot: Digiview)

Taktübernahme mehrerer Signalwerte

- Die gleichzeitige Taktübernahme mehrerer Signalwerte ist komplex
 - Datenbits eines Wortes müssen im Zusammenhang bleiben
- Beispiel: Dualzahl mit 3 bit
 - Wechsel von 1_{10} auf 2_{10}
 - Die einzelnen Bits können nicht **exakt gleichzeitig** übernommen werden
 - Beim Übergang kann fälschlich „000“ oder „011“ gelesen werden

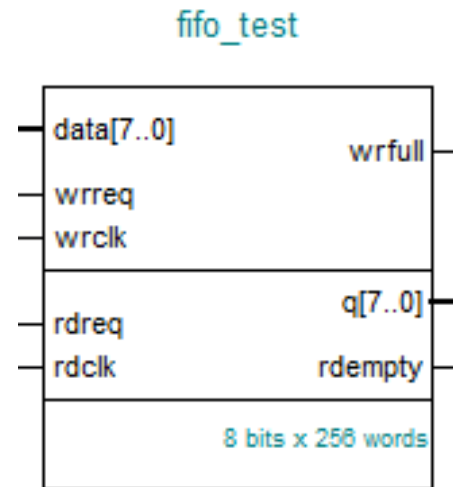


- Mögliche Lösung: **Gray-Code**
- Der **Gray-Code** ist ein **einschrittiger Code**
 - Aufeinander folgende Codewörter unterscheiden sich stets an nur einer Stelle



FIFO-Speicher

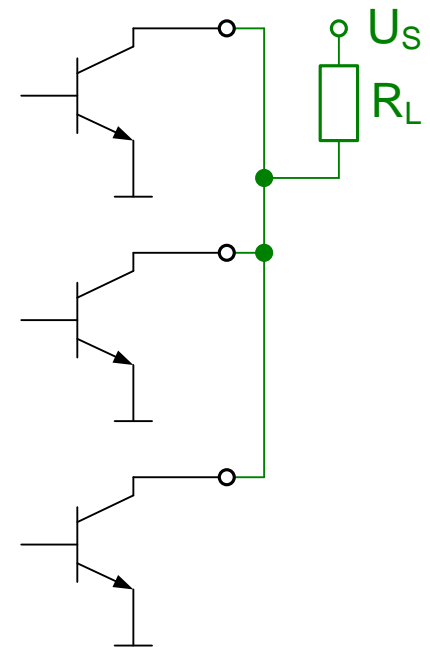
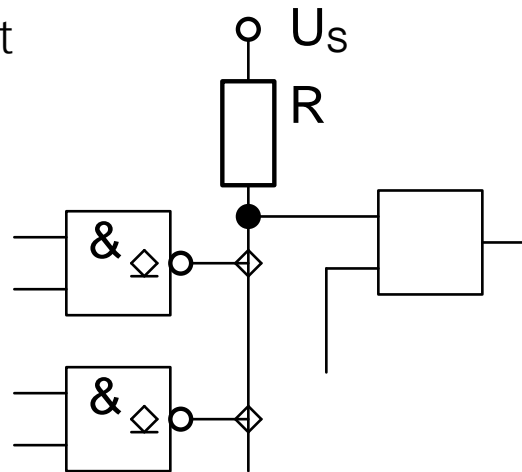
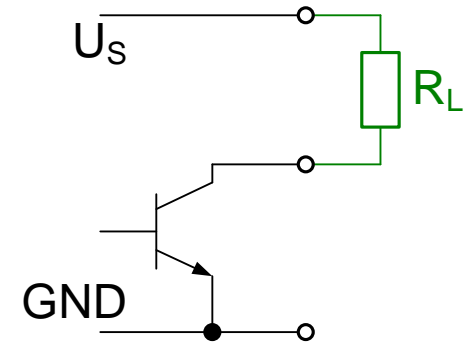
- Die Datenübernahme mit Gray-Code funktioniert nur wenn
 - Daten eine feste Reihenfolge haben
 - Der Übernahmetakt schneller als der Quelltakt ist
- Für allgemeine Anwendungen werden **FIFO-Speicher** eingesetzt
 - FIFO = „First-In-First-Out“
- Speicher mit zwei Schnittstellen („Dual-Port-RAM“)
 - Eingang mit Quelltakt
 - Ausgang mit Zieltakt
- Ansteuerung des FIFOs muss Taktübergang berücksichtigen
 - FIFOs sind als Bauelemente oder Schaltungsbeschreibung verfügbar
 - Bild zeigt FIFO aus Bibliothek des Herstellers Altera
 - **Fragen:**
 - Welche Bedeutung haben die I/Os?
 - Welche Bedeutung hat der horizontale Strich?



(Screenshot: Altera)

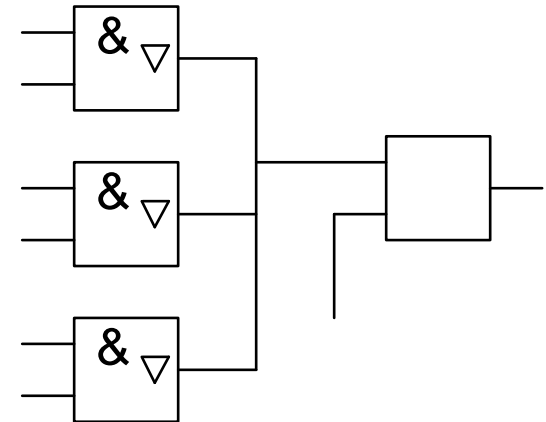
6.8 Spezielle Ein-/Ausgangsstrukturen – Open-Kollektor

- Bei der Ausgangsstufe kann der Pfad zur Versorgungsspannung U_S durch einen externen Lastwiderstand R_L ersetzt werden.
 - Bezeichnung: **Open-Kollektor**
- Dies erlaubt eine Zusammenschaltung mehrerer TTL-Bausteine.
- Wenn **alle** Ausgänge „**high**“ sind (also Transistor sperrt), ist der gemeinsame Ausgang „high“.
- Wenn **ein** Ausgang „**low**“ ist (also Transistor offen), ist der gemeinsame Ausgang „low“.
 - Die Zusammenschaltung ist eine UND-Verknüpfung.
 - Bezeichnung: **Wired-AND**
- Ein Open-Kollektor-Ausgang wird durch ein Symbol ähnlich einer Raute dargestellt.



Tri-State Ausgänge

- Eine andere Möglichkeit zum Zusammenschalten mehrerer Ausgänge ist, inaktive Ausgänge **hochohmig** zu schalten.
 - Bei einem hochohmigen Ausgang sind beide Pfade im Ausgangstreiber (nach U_S und GND) gesperrt.
- Jeder Ausgang kann drei Zustände einnehmen (engl. „Tri-state“):
 - Null („0“)
 - Eins („1“)
 - Hochohmig („Z“)
- Ein Tri-State-Ausgang wird durch ein auf der Spitze stehendes Dreieck dargestellt.
- Durch die Steuerung muss sichergestellt werden, dass stets nur ein Ausgang aktiv ist.
- Tristate-Ausgänge eignen sich auch für bidirektionale Datenübertragung.



Beispiel: Verbindung von CPU und RAM.

- Zum Schreiben gibt die CPU Daten an das RAM.
- Zum Lesen holt die CPU Daten aus dem RAM.

Schmitt-Trigger

- Werden digitale Signale durch Spannungspegel dargestellt, gibt es einen Übergangsbereich
 - Dieser Übergangsbereich wird normalerweise zügig durchlaufen
- Probleme können auftreten, wenn der Übergangsbereich langsam durchlaufen wird und/oder mit Rauschen überlagert ist
 - Ein **Schmitt-Trigger** am Eingang behebt diese Probleme
 - Die Schaltschwelle hat eine **Hysterese**, ist also abhängig vom aktuellen Ausgangswert
 - Bei einer ‚0‘ ist eine „deutliche 1“ erforderlich
 - Bei einer ‚1‘ ist eine „deutliche 0“ erforderlich

