– Lösungen –
8. DigitaleSchaltungen

- Elektronik für Informatiker -

von den Grundlagen bis zur Mikrocontroller-Applikation

Manfred Rost Sandro Wefel

23. November 2021

https://doi.org/10.1515/9783110609066

Verlag: De Gruyter Oldenbourg

Anmerkung: Bildnummern und Seitenzahlen beziehen sich auf die 2. Auflage des Buches.

8 Kombinatorische und sequentielle Digitalschaltungen

8.1 Auf Seite 279 wurde bereits gezeigt, dass man durch Invertierung der einzelnen Bits von Y und durch Addition einer 1 den Wert $-Y$ erhält. Somit wird das Ergebnis der Subtraktion korrekt sein, insofern der Carry-Ripple-Addierer (CRA) n-Bit Zweierkomplementzahlen und den eingehenden Wert c_{in} korrekt addiert und der Wert $-Y$ und das Ergebnis der Addition mit n Stellen darstellbar sind.

Um zu zeigen, dass der CRA korrekt addiert, müssen wir zuerst zeigen, dass ein Volladierer (FA) drei Bits x,y und c_{in} korrekt addiert, also die Funktion $\mathbb{B}^3 \to \mathbb{B}^2 : f(x, y, c_{in}) := (c_{out}, s)$ mit $2 \cdot c_{out} + s = x + y + c_{in}$ berechnet. Der FA (siehe Abb. 8.8) berechnet $c_{out} = x \wedge y \vee c_{in} \wedge (x \oplus y)$ und $s = x \vee y \oplus c_{in}$. (Zur Unterscheidung zwischen dezimaler Algebra und der binären Verknüpfung wurde hier $\vee$ und $\wedge$ für die Binärverknüpfung gewählt.) Durch Prüfen der Belegungen erkennen wir schnell, dass der FA die Gleichung $2 \cdot c_{out} + s = x + y + c_{in}$ erfüllt und somit drei binäre Werte korrekt addiert.

Die Addition von Binärzahlen der Länge n können wir analog zur Funktionswiese des CRA konstruktiv erläutern. Wir beginnen mit der Addition zweier Bits und erhalten eine Summe s und einen Übertrag c. Es gilt $2 \cdot c + s = x + y$. Setzen wir dafür einen Volladdierer ein und gilt $c_{in} = 0$, so entspricht das der Addition zweier Binärzahlen der Länge 1. Die Addition zweier beliebiger Zahlen der Länge l können wir nun rekursiv über die Addition zweier Zahlen der Länge $l - 1$ erläutern. Davon ausgehend, dass wir einen korrekten Addierer der Länge $l - 1$ nutzen, addieren wir damit die niederwertigen $l - 1$ Bits und erhalten eine Summe $(s_{l-2}, \ldots, s_0)$, also ein Teilergebnis der Länge $l - 1$ und einen Übertrag s_{l-1} mit der Wertigkeit 2^{l-1}. Wir addieren diesen Übertrag zu den Ziffern der Summanden mit der gleichen Wertigkeit (x_l und y_l) und erhalten damit das Ergebnis. Diese Addition entspricht wieder der Verknüpfung mit einem FA. Somit erhalten wir für jede Stelle das Ergebnis mit der entsprechenden Wertigkeit und einen ausgehenden Übertrag s_l.

Die Interpretation einer Zahl X in Zweierkomplement-Darstellung (Anhang A.5.2) mit n Stellen vor dem Komma[1] ist

$$\phi(X) = \sum_{i=0}^{n-2} x_i \cdot 2^i - x_{n-1} \cdot 2^{n-1}$$

[1] Wir betrachten erst mal nur Zahlen ohne Nachkommastellen. Die gesamte Betrachtung ist auch für Zahlen mit Nachkommastellen gültig, es muss lediglich darauf geachtet werden, dass bei der Umwandlung von Y zu $-Y$ nicht der Wert 1, sondern der Wert 2^{-k} aufaddiert wird.

Wenn der Addierer Zweierkomplement korrekt addiert, dann muss die Addition von X und $-X$ als Ergebnis den Wert 0 liefern

$$\begin{aligned}
X + (-X) &= \sum_{i=0}^{n-2} x_i \cdot 2^i - x_{n-1} \cdot 2^{n-1} + \left(\sum_{i=0}^{n-2} \overline{x_i} \cdot 2^i - \overline{x_{n-1}} \cdot 2^{n-1} + 1 \right) \\
&= \sum_{i=0}^{n-2} x_i \cdot 2^i - x_{n-1} \cdot 2^{n-1} + \left(\sum_{i=0}^{n-2} (1 - x_i) \cdot 2^i - (1 - x_{n-1}) \cdot 2^{n-1} + 1 \right) \\
&= \sum_{i=0}^{n-2} x_i \cdot 2^i - x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} (-x_i) \cdot 2^i + \sum_{i=0}^{n-2} 2^i - 2^{n-1} + x_{n-1} \cdot 2^{n-1} + 1 \\
&= \sum_{i=0}^{n-2} 2^i - 2^{n-1} + 1 \\
&= 2^{n-1} - 1 - 2^{n-1} + 1 \\
&= 0
\end{aligned}$$

8.2 Wir gehen von der Korrektheit des Verfahrens der Addition im Zweierkomplement für den Fall aus, dass das Ergebnis mit der gegebenen Anzahl von Stellen darstellbar ist. Einen Beweis dazu findet man z.B. in [BM08].

Untersuchen wir zuerst die Fälle, in denen ein Überlauf auftreten kann. Die vorderste (linke) Stelle einer Zweierkomplementzahl kodiert das Vorzeichen, mit einer 1 für negative und einer 0 für positive Zahlen. Eine Bereichsüber- bzw. -unterschreitung tritt bei der Addition genau dann auf, wenn das Ergebnis negativ sein sollte, aber positiv ist und umgekehrt.

Betrachten wir dazu die vorderste Stelle der Addition zweier positiver n-Bit-Zahlen A und B, mit $S = A + B$. Das Ergebnis muss in dem Fall positiv sein: $s_{n-1} = 0$. Für die jeweiligen Bits $0 \leq i < n$ der Summe gilt:

$$s_i = (a_i + b_i + c_{i-1}) \mod 2$$

und der jeweilige Übertrag (Carry):

$$c_i = \lfloor (a_i + b_i + c_{i-1})/2 \rfloor$$

Für die vorderste Stelle $n - 1$ gilt somit

$$\begin{aligned}
s_{n-1} &= (a_{n-1} + b_{n-1} + c_{n-2}) \mod 2 \\
&= (0 + 0 + c_{n-2}) \mod 2 \\
&= c_{n-2}
\end{aligned}$$

Es gilt $c_{n-2} = \lfloor (a_{n-2} + b_{n-2} + c_{n-3})/2 \rfloor$. Sind z.B. die Bits a_{n-2} und b_{n-2} auf 1 gesetzt, ist $c_{n-2} = 1$ und damit $s_{n-1} = 1$. Die Addition dieser beiden positiven Zahlen

liefert ein negatives Ergebnis und damit einen Überlauf. Analog gilt es für die Addition zweier negativer Zahlen.

Werden zwei Zahlen mit unterschiedlichem Vorzeichen addiert, so ist der Betrag des Ergebnisses in jedem Fall kleiner als der größere Betrag der beiden Zahlen selbst. Somit ist das Ergebnis in jedem Fall in der gegebenen Bitlänge darstellbar. Wenn die Addition im Zweierkomplement korrekt ist, worauf wir uns hier berufen, dann kann es keinen Überlauf geben.

8.3 Der Wert s_n, der ja auch c_{n-1} entspricht (siehe Abb. 8.9) ist der Übertrag an der vorderen Stelle. Die Behauptung ist somit, es leigt kein Überlauf bzw. Unterlauf vor, wenn $c_{n-1} = s_{n-1}$ gilt.

Betrachten wir dazu die vorherige Aufgabe. Ein Überlauf tritt nur bei gleichem Vorzeichen auf. Für den Fall positiver Zahlen, also $a_{n-1} = b_{n-1} = 0$ wäre es der Fall, wenn das Ergebnis von $c_{n-2} = 1$ ist. Dann ist das Summenbit

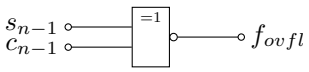
$$s_{n-1} = a_{n-1} = b_{n-1} + c_{n-2} \mod 2 = 0 + 0 + 1 \mod 2 = 1$$

Der Übertrag ist aber

$$c_{n-1} = \lfloor (a_{n-1} + b_{n-1} + c_{n-2})/2 \rfloor = \lfloor (0 + 0 + 1)/2 \rfloor = 0$$

und somit gilt $c_{n-1} \neq s_{n-1}$. Analog können wir den Beweis für zwei negative Zahlen mit $a_{n-1} = b_{n-1} = 1$ und der Übertrag $c_{n-2} = 0$, der zu einem Unterlauf führt, erbringen.

Die zu ergänzende Schaltung muss lediglich die beiden Werte c_{n-1} und s_{n-1} vergleichen und bei Gleicheit den Wert 1 ausgeben. Dafür genügt ein XNOR-Gatter:



8.4 Ein n-Bit Carry-Ripple Addierer benötigt (ohne Subtrahierer-Zusatzschaltung) insgesamt n Volladierer. Ein Volladdierer besteht aus 5 Gattern (vgl. Abb. 8.7 und Abb. 8.8). Somit gilt für den Platzbedarf, ohne dabei die Verdrahtung zu berücksichtigen:

$$A(\text{CRA}(n)) = 5 \cdot n$$

Für den Aufbau eines CSA der Bitbreite n werden drei CSA der Breite $n/2$ benötigt. Hinzu kommen die die Multiplexer. Ein einfacher Multiplexer benötigt 3 Gatter (vgl. Abb. 8.3). Somit gilt für den Platzbedarf, ohne dabei die Verdrahtung zu berücksichtigen:

$$A(\text{CSA}(n)) = 3 \cdot A(\text{CSA}(n/2)) + 3 \cdot n$$

Der Platzbedarf für ein CSA der Bitbreite 1 ist der Platzbedarf eines Volladdierers, also $A(\text{CSA}(1)) = 5$.

Wir brauchen eine Methode zur Lösung der Rekursion. Für die Laufzeitabschätzung rekursiv definierter Funktionen der Form $T(n) = a \cdot T(\frac{n}{b}) + f(n)$ nutzt der Informatiker u.a. die Mastermethode (siehe z.B. [CLRS10]) zur Bestimmung asymptotischer Schranken, insofern die Methode anwendbar ist. Hier ist gegeben: $a = 3, b = 2$ und $f(n) = 3 \cdot n$. Es gilt

$$f(n) = \mathcal{O}\left(n^{\log_b a - \varepsilon}\right) \quad \text{also} \quad 3 \cdot n = \mathcal{O}\left(n^{\log_2 3 - \varepsilon}\right)$$

für ein $\varepsilon > 0$, da mit $\varepsilon \approx 0,03$ gilt $n^{1,53\ldots - \varepsilon} = n^{1,5}$ und $3 \cdot n = \mathcal{O}\left(n^{1,5}\right)$. Wir können den ersten Fall des Mastertheorems anwenden, womit folgt:

$$A(\text{CSA}(n)) = \Theta(n^{1,53\ldots})$$

Im Vergleich dazu liegt $A(\text{CRA}(n)) = \Theta(n)$, wächst also nur linear in der Größe.

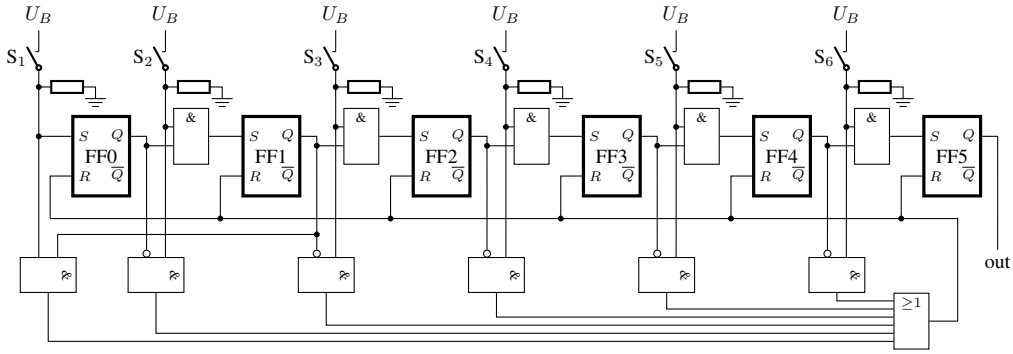
8.5 Ein CSA ist rekursiv definiert. Die Rekursion terminiert bei der Addition einer Bitbreite von 1 Bit, für die ein Volladdierer (FA) verwendet wird. Für alle anderen Rekursionsstufen kommt als längster Pfad die Verbindung über den Multiplexer (MUX) hinzu. Die Pfadlänge über den MUX beträgt 2 Gatter, über den FA 3 Gatter. Die Anzahl der Rekursionsstufen für ein n-Bit CSA beträgt $\lceil \log_2 n \rceil$. Die Pfadlänge des kritischen Pfades beträgt $2 \cdot \log_2 n + 3$ Gatter.

8.6 Ein Vergleich zweier Zahlen kann über die Subtraktion erfolgen. Um zu prüfen, ob die Zahl $A > B$ ist, ziehen wir die Zahl A von B ab. Ist das Ergbnis negativ, also das vorderste Bit des Ergebnisses bzw. das Flag `neg` auf 1 gesetzt, dann gilt $A > B$. Bei der Subtraktion kann natürlich ein Überlauf auftreten, falls z.B. B negativ war und A postiv. Somit muss zusätzlich das Flag `ovfl` ausgewertet werden.

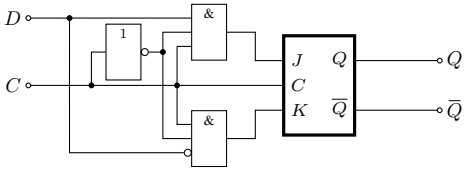
Für die Auswertung $A = B$ benötigen wir entweder eine Möglichkeit zu testen, ob das Ergebnis der Subtraktion den Wert 0 hat oder wir führen eine weitere Subtraktion aus. Gilt weder $A > B$, noch $B > A$, so muss $A = B$ gelten.

8.7 In der Zweierkomplement-Darstellung gibt es nur eine Darstellung der Zahl 0, nämlich die Darstellung, in der alle Bits auf 0 gesetzt sind. Wir benötigen einen schnellen (parallelen) Test aller Bits auf 0. Dazu können wir eine Schaltung analog zum Compare-Baustein einsetzen (vgl. Abb 8.12). Zuerst wird jedes Bit des Ergebnisses C mittels eines XNOR-Bausteins mit dem Wert 0 verglichen. Das Ergebnis ist 1, wenn das betrachtete Bit von C gleich 0 ist. Anschließend sammeln wir diese Ergebnisse in einer 2:1-Reduktion jeweils mittels einer AND-Verküfung auf. Das Endergebnis, welches über das Flag ausgeben wird ist genau dann 1, wenn der Wert von C 0 ist. Die Pfadlänge ist logarithmisch, bezogen auf die Bitbreite n. Alternativ kann auch ein AND-Gatter mit n Eingängen in der ALU integriert werden, z.B. als spezielles CMOS-Gatter.

8.8 Ähnlich einem Schieberegister können wir eine Schaltung aufbauen, bei der ein Wert 1 vom ersten FF zum letzten FF und damit zum Ausgang durchgeschoben wird. Dabei wird der Wert nur „weitergeschoben“, wenn die richtige Taste gedrückt wird. Somit muss der Wert vom vorherigen FF „UND“ die die richtigen Taste gedrückt werden. Ist das vorherige FF noch noch nicht aktiv „UND“ wird dann die somit falsche Taste gedrückt, dann müssen alle FFs wieder in den Ausgangszustand zurück versetzt werden (Reset). Das kann bei bezogen auf die Reihenfolge falschen Tasten „ODER“ bei bei nicht verwendeten Taste auftreten. Wir benötigen FFs, AND und OR-Gatter.

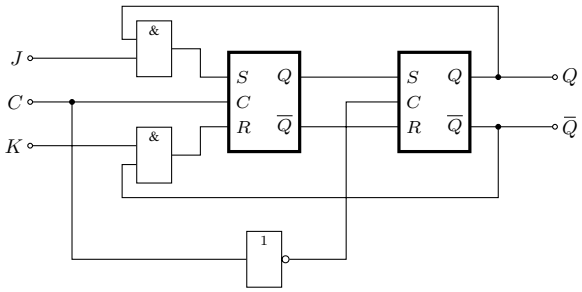


8.9 Wird der K-Eingang eines JK-FFs mit dem invertierten Wert des J-Eingangs belegt, so erhält man ein D-FF. Zur Umwandlung des pegel- in ein flankengesteuertes FF kann z.B. das Prinzip aus Abb. 8.28 angewendet werden. Dazu schalten wir ein Verzögerungsgatter (Inverter) vor die AND-Gatter, welche das D-Signal einmal direkt für Eingang J und einmal invertiert für den Eingang K bereitstellen. Durch die Verzögerung des Inverters entsteht bei der ansteigenden Taktflanke entweder am J- oder am K-Eingang eine kurze Signalspitze, passend zum D-Wert. Der Ausgang schaltet auf den D-Wert um.



8.10 Entsprechend der Abb. 8.38 kann das System einfach aus zwei RS-Latches aufgebaut werden. Lediglich der Master-Latch muss etwas erweitert werden, da neben den Eingangswerten R und S (hier J und K) noch die rückgekoppelten Werte des Slave-Latches

verarbeiten muss. Diese können wir jedoch einfach mit J und K durch zusätzliche AND-Gatter verknüpfen.



8.11 Damit die Wertübernahme am Master ohne Verzögerung bezogen auf den Takt erfolgt, also kein Taktversatz (engl. *clock skew*) auftritt, muss der Master den Takt sofort erhalten. Der Slave hingegen darf in seine aktive Phase erst dann eintreten bzw. seine aktive Flanke erst dann erhalten, wenn der Master den Wert gespeichert hat.

Dies kann z.B. dadurch erreicht werden, dass der Master H-aktiv und der Slave L-aktiv ist. Um einen L-aktiven Slave zu erhalten, kann man z.B. ein H-aktives FF nehmen, den Takt invertieren und somit bewusst verzögert den invertierten Takt an den Slave anlegt. Diese Verzögerung hat den Effekt, dass der Slave erst aktiv wird, wenn der Master bereits inaktiv ist, d.h. das Wirkintervall ist bereits vorüber, wenn der Slave aktiv wird und man in den Zeitraum des Kippintervalls eintritt.

8.12

-	0	1
0	0	1
1	1	0

·	0	1
0	0	0
1	0	1

/	0	1
0	-	0
1	-	1

Die Subtraktion (modulo 2) entspricht der XOR-Verknüfung, analog zur die Addition.

Die Multiplikation entspricht einer AND-Verknüfung.

Die Division entspricht einer Multiplikation mit dem inversen Element des Divisors, wobei es in $GF(2)$ nur das Inverse zur 1 gibt: $x \cdot (-1) \mod 2$. Da man auch auch nicht durch 0 dividieren darf, ist das Ergebnis an der Stelle auch nicht wichtig. Wir können uns eine passende Belegung aussuchen und z.B. die Identität verwenden, also $A/B = A$.

8.13 Betrachten wir jeweils die Folge bei Initialisierung mit (1,0,0,0) oder hexadezimal 8.

Externe Rückkopplung
Fibonacci-LFSR

S	Eingabe				Hex
	x	x^2	x^3	x^4	
1	0	1	0	0	4
2	1	0	1	0	a
3	0	1	0	1	5
4	0	0	1	0	2
5	0	0	0	1	1
6	1	0	0	0	8

Interne Rückkopplung
Galois-LFSR

S	Eingabe				Hex
	x	x^2	x^3	x^4	
1	0	1	0	0	4
2	0	0	1	0	2
3	0	0	0	1	1
4	1	0	1	0	a
5	0	1	0	1	5
6	1	0	0	0	8

Jeweils nach 6 Schritten wiederholt sich die Eingabe.

Bei abweichender Initialisierung, z.B. (1,1,1,0) können auch andere Folgen entstehen. Diese können auch in der Länge abweichen, wie die kürzeren Folgen bei der Initialisierung mit (1,1,0,1) im Fibonacci-LFSR oder mit (1,0,0,1) im Galois-LFSR.

8.14 Betrachten wir wieder jeweils die Folge bei Initialisierung mit (1,0,0,0) oder hexadezimal 8.

Externe Rückkopplung
Fibonacci-LFSR

S	Eingabe				Hex
	x	x^2	x^3	x^4	
0	1	1	0	0	c
1	0	1	1	0	6
2	0	0	1	1	3
3	0	0	0	1	1
4	0	0	0	0	0
5	1	0	0	0	8

Interne Rückkopplung
Galois-LFSR

S	Eingabe				Hex
	x	x^2	x^3	x^4	
0	0	1	0	0	4
1	0	0	1	0	2
2	0	0	0	1	1
3	1	0	1	0	a
4	0	1	0	1	5
5	1	0	0	0	8

Jeweils nach 6 Schritten wiederholt sich die Eingabe.

8.15 Es handelt sich um ein 4-Bit LFSR. Bei Betrachtung der Folge erkennt man, dass ein Wert offensichtlich vom ersten zum letzten FF durchgestellt wird, sich also nicht beim Übergang zwischen zwei FFs verändert. Somit bestimmt jeweils nur die Änderung am Eingangswert über die Folge. Das entspricht einer externen Rückkopplung.

Wir betrachten den Eingangswert $f_x^{(1)}$ als Funktion der vorhergehenden Werte

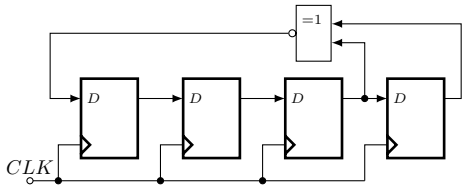
$$f_0^{(1)} = (c_1 \cdot f_x^{(0)}) \oplus (c_2 \cdot f_{x^2}^{(0)}) \oplus (c_3 \cdot f_{x^3}^{(0)}) \oplus f_{x^4}^{(0)}$$

wobei in diesem Falle für $\oplus$ (XOR) auch ein $\overline{\oplus}$ (XNOR) stehen kann, je nach verwendetem Gatter. Es gibt 8 Kombinationen von c_1 bis c_3, die getestet werden können. Für XOR

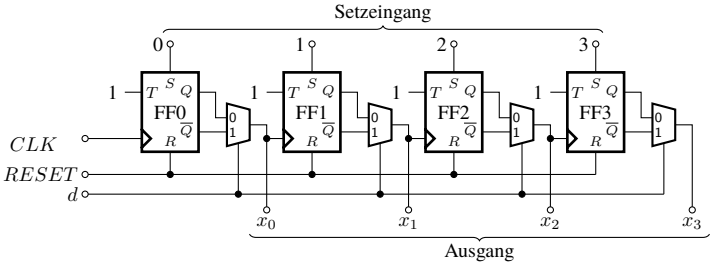
findet sich keine Belegung zur Erfüllung dieser Folge. Für XNOR gibt es die Belegung $c_1 = 0, c_2 = 0, c_3 = 1$, also

$$f_0^{(1)} = f_{x^3}^{(0)} \overline{\oplus} f_{x^4}^{(0)}$$

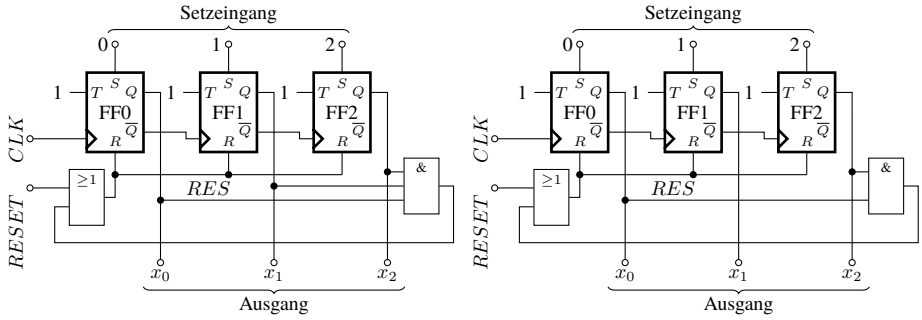
Das enstpricht dem Polynom $p_e = x^4 + x^3 + 1$ und der folgenden Schaltung.



8.16 Bei einem asynchronen Zähler ist immer ein Taktversatz zu erwarten und dessen Ausweitung in gewissen Grenzen erlaubt. Somit sollte das Einbauen eines Multiplexers vor den nachfolgenden Eingang kein Problem darstellen.

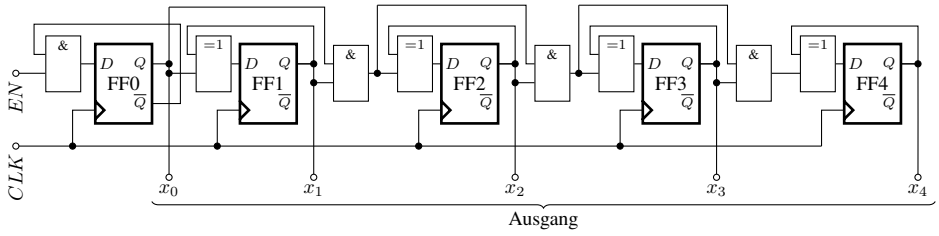


8.17 Für n=7 und n=5 genügt ein 3-Bit-Zähler. Für n=7 muss ein Reset bei der Bitkombination $(x_0, x_1, x_2) = (1, 1, 1)$ und für n=5 bei der Bitkombination $(x_0, x_1, x_2) = (1, 0, 1)$ ausgelöst werden.

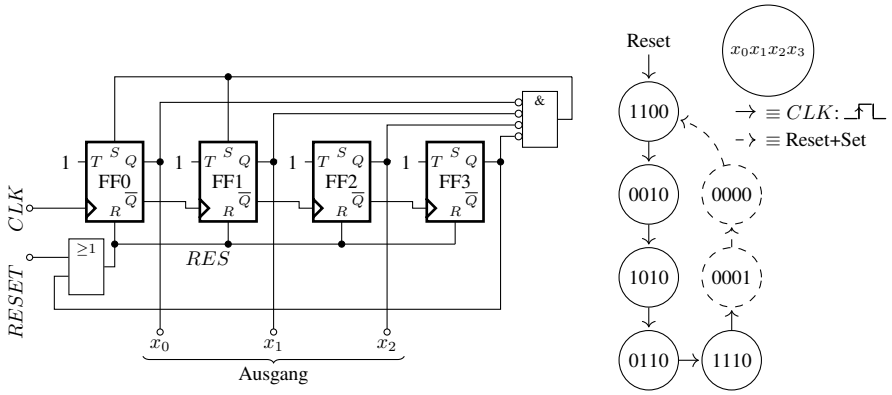


(a) n=7 **(b)** n=5

8.18 Der Eingang D einer Stufe ergibt sich aus der konjunktiven Verknüpfung der Ausgänge der vorhergehenden Stufen exklusiv-Oder verknüpft mit dem eigenen Ausgang. Wir benötigen noch ein Enable-Signal. Im Gegensatz zum Toggle-FF besitzt das D-FF keinen Enable-Eingang, wir können es also nicht nach Abb. 8.58 umsetzen. Zwischen Takteingang und Taktsignal dürfen wir auch keine Gatter einfügen. Somit muss die Aktivierung über die Rückkopplung gesteuert werden. Hier hilft folgende Beobachtung; Beharrt der Ausgang des ersten FF auf einem Pegel 0, so werden die nachfolgenden Gatter nicht schalten. Wir müssen somit lediglich die Rückkopplung von $\overline{Q}$ am FF0, z.B. über AND-Gate unterbrechen und somit an Q dauerhaft den Wert 0 ausgeben.

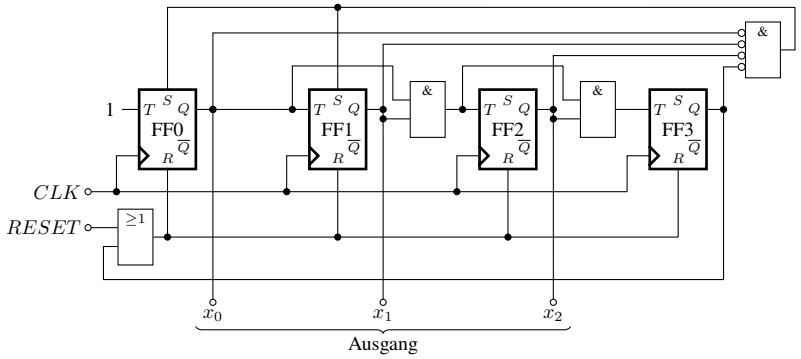


8.19 Wir benötigen einen modulo-8-Zähler, der bei Erreichen des Wertes 8 das System auf den Wert 3 initialisiert, also zuerst ein Reset auslöst und dann den Wert 3 setzt. Das Auslösen des Resets erfolgt analog zu Aufgabe 8.17, hier aber bei Erreichen der 8, also wenn der Ausgang von FF3 auf 1 wechselt. Nach einem Reset liegen alle FF-Ausgänge auf 0. Dies wird detektiert und sofort über die 3 die asynchronen Setzeingänge die benötigten FF (0 und 1) wieder auf 1 gesetzt. Das „Setzen“ eines FF entspricht dem Anlegen einer 1 am asynchronen Eingang S. Durch dieses Setzen des Wertes 3 liegt keine 0 am Ausgang an und die Setzeingänge gehen wieder auf 0, bis erneut ein Reset ausgelöst wird.

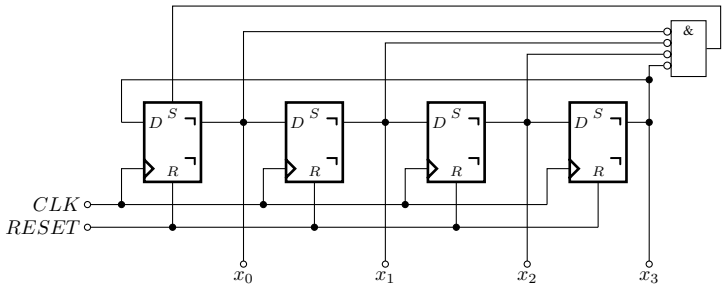


(a) Schaltung **(b)** Zustandsübergänge

8.20 Dazu erweitern wir z.B. eine Schaltung aus Abb. 8.58, Abb. 8.60 oder aus Abb. 8.69 um die Reset-Logik entsprechend der vorhergehenden Aufgabe.



8.21 Entsprechend der vorhergehenden Aufgabe können wir auch hier wieder die S-Eingänge verwenden. Nach der Aufgabenstellung kann dies taktgesteuert erfolgen oder asynchron, da der S-Eingang auf ein asynchrones Reset reagiert.



8.22 Betrachten wir dazu die Abbildung des asynchronen BCD-Zählers und dessen Zeitverhalten. Hier ist deutlich zu sehen, dass die Sprungantwort des FFs zeitlich verzögert zum Takt erfolgt, insbesondere bei dem letzten FF zu den Zeitpunkten 75 ns und 95 ns. Diese Verzögerung ergibt sich aus der Addition der Laufzeiten der FFs. Die Summe darf die Periodendauer nicht überschreiten, da andernfalls bei der nächsten Taktflanke das Zählergebnis noch nicht an den Ausgängen anliegt. Ein Teiler, der einen bestimmten Zählwert erfragt, würde dann zu einem falschen Zeitpunkt bzw. gar nicht schalten.

8.23 Beim Ringzähler wird der Ausgangswert eines FF mit jedem Takt in das nächste FF und vom letzten zurück in das erste FF geschoben. Ein Zähler mit 8 FFs würde an einem beliebigen Ausgang beim Laden eines Muster 10000000 jeweils innerhalb von 8 Takten einmal den Wert 1 für die Dauer eines Taktes ausgeben.

Um nach 4 Takten genau für einen Takt den Wert 1 auszugeben, können wir das Muster 10001000 laden und einen beliebigen Ausgang verwenden. Das Verhältnis zwischen t_{high} und $t_{periode}$ ist 1:4, entspricht also dem geforderten Tastgrad.

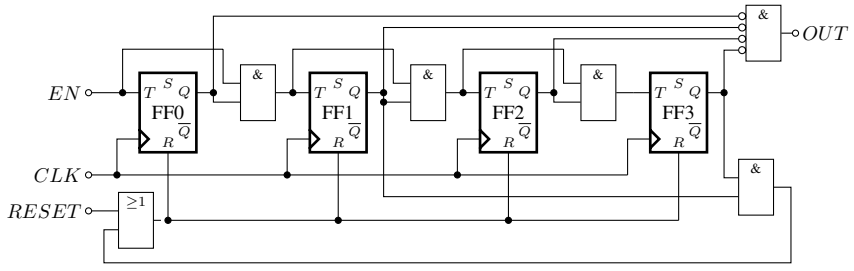
8.24 Mit der Initialisierung 101 liegt am Ausgang $\overline{Q}$ des letzten FFs der Wert 0. Demzufolge wird mit dem ersten Takt eine 0 am Eingang des ersten FF eingelesen. Die neue Belegung ändert ist 010 und beim nächsten Takt wieder 101. Es folgt mit jedem Takt ein Wechsel zwischen den beiden Belegungungen. Demzufolge wiederholt sich der Wert an einem beliebigen Ausgang aller zwei Takte. Es ist ein 2:1-Teiler mit einem Tastgrad 1:2.

8.25 Betrachten wir die entstehende Folge:

11001100
11100110
11110011
01111001
00111100
10011110
11001111
01100111
00110011
00011001
00001100
10000110
11000011
01100001
00110000
10011000
11001100

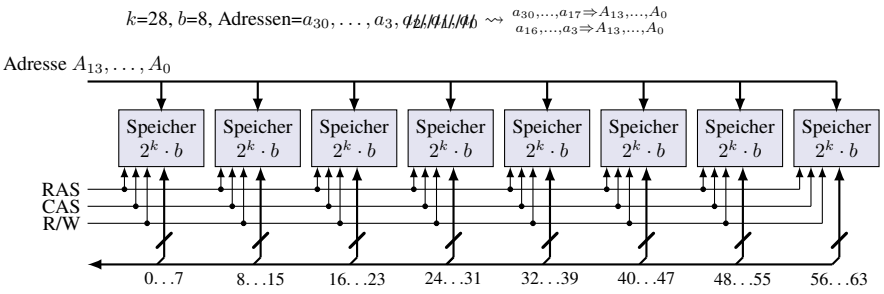
Wir können uns jetzt einen beliebigen Ausgang aussuchen. Nehmen wir den ersten Ausgang, also die erste Spalte der Tabelle. Die sich wiederholende Folge ist 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1. Bei den ersten 6 Belegungen vermutet man einen Teiler 4:1. Dann folgt allerdings nicht der gewohnte Wechsel, sondern eine Pause von zwei Takten. Das gleiche gilt für die nächsten 4 Takte. Wie können kein Teilungsverhältnis und somit auch keinen Tastgrad angeben.

8.26 Analog zur Aufgabe 8.20 können wir wieder einen synchronen Zähler nutzen. Der Reset erfolgt bei Schritt 12. Dafür brauchen wir einen 4-Bit Zähler. Leider können wir an keinem Ausgang das Erreichen des Zustands 11 direkt ablesen. Wir benötigen eine Zusatzschaltung, die z.B. bei genau einem der 11 stabilen Zustände am Ausgang den Wert 0 ausgibt. Das könnte z.B. der Zustand 0000 sein. Dieser Zustand wird über ein AND-Gatter ermittelt.



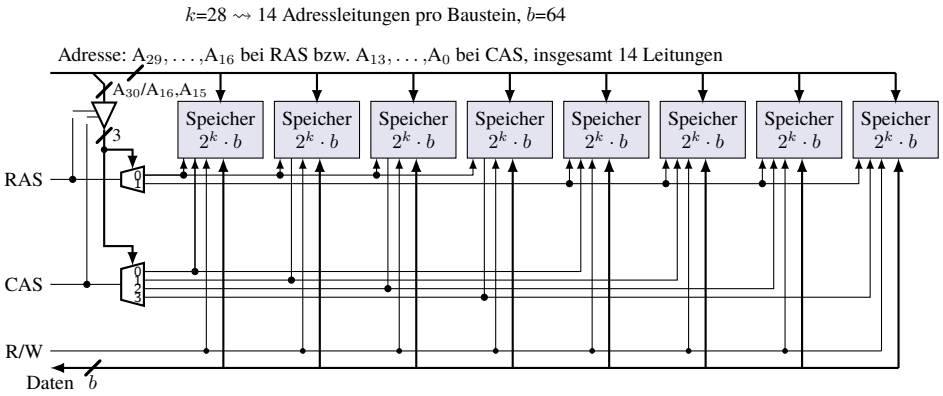
Nur für die Dauer einer Taktperiode, beim Zustand 0000 liegt der Ausgang auf 1. Die anderen 10 Zustände liegt er auf 0. Der Tastgrad ist 1:11.

8.27 (a) Die Datenbusbreite der geforderten 64 Bit wird durch die Zusammenschaltung der 8 Bausteine erreicht. Demzufolge hat jeder Bausteine die Datenbusbreite von $\frac{64}{8}$ Bit $=$ 8 Bit. Für die Adressierung von 2 Gibibyte $= 2^{31}$Byte werden 31 Adressleitungen benötigt, da die Adressierung byteweise erfolgt. Allerdings wird bei einem adressierten Byte immer der komplette 64 Byte-Block ausgegeben, in dem sich das adressierte Byte befindet. Demzufolge sind nur die oberen 28 Bit der Adresse, also $a_{30}, \ldots, a_3$ zur Ansteuerung erforderlich. Diese 28 Bit passen zur Speichergröße eines Bausteins, da jeder Baustein $\frac{1}{8}$ der Gesamtkapazität, also 256 Mebibyte beiträgt. Die 28 Bit der Adresse sind nun noch pro Baustein auf dessen Speichermatrix in Zeilen und Spalten aufzuteilen. Nutzen wir z.B. die Aufteilung in ein Quadrat der Kantenlänge 2^{14}, so benötigen wir für die 28 Bit-Adresse jeweils 14 Leitungen für Zeilen und Spalten.



(a) Vergrößerung der Datenbusbreite

(b) Jeder einzelne Baustein liefert die geforderten 64 Bit für den Datenbus. Der gesamte Adressraum wird aber erst durch Zusammenschaltung der 8 Bausteine erreicht. Jeder Bausteine hat wieder eine Speicherkapazität von 256 Mebibyte, wird aber nur bei bestimmten Adressen aktiviert.
Wir benötigen 31 Adressen für die Adressierung des gesamten Speichers. Eine mögliche Variante ist die Aufteilung in zwei mal 16 Bit, also insgesamt 32 Adressleitungen. Davon nutzen wir den höherwertigen Anteil zur Auswahl des Bausteins durch die Ansteuerung der Multiplexer, also z.B. die Bits 31 und 30 bei RAS und die Bits 15 und 14 bei CAS, von denen wir allerdings nur die Bits 30, 15 und 14 für die drei Bits zur Ansteuerung der Multiplexer benötigen. Die Bits 0 bis 13 und 16 bis 29 liefern zusammen die 28-Bit-Adresse des ausgewählten Bausteins.



(b) Vergrößerung des Adressraumes

8.28 Im normalen Rechnerbetrieb werden sehr häufig Werte im RAM abgelegt, es wird also eine sehr hohe Anzahl schreibender Zugriffe erforderlich sein, welche zu einem schnellen Verschleiß des Flash-Speichers führen. Um eine hohe Schreibgeschwindigkeit zu erreichen, müssen die Flash-Zellen einzeln angesteuert werden bzw. sollte das Löschen nicht blockweise erfolgen. Das erfordert relativ große Strukturen und viel Platz, wobei die Schreibgeschwindigkeit aufgrund des erforderlichem Löschens immer noch relativ gering gegenüber statischem oder dynamischen RAM ist.

8.29 Eine Lage besitzt eine Fläche von $1\,\text{cm}^2$. Bei einer Strukturgröße von 18 nm kann maximal folgende Anzahl von Zellen pro Lage untergebracht werden.

$$\text{NAND-Flash: } \frac{1\,\text{cm}^2}{4 \cdot 18\,\text{nm}^2} \approx 1388 \cdot 10^9 \text{ Bit} \approx 165\,\text{GiB}$$

$$\text{NOR-Flash: } \frac{1\,\text{cm}^2}{10 \cdot 18\,\text{nm}^2} \approx 555 \cdot 10^9 \text{ Bit} \approx 66\,\text{GiB}$$

$$\text{DRAM: } \frac{1\,\text{cm}^2}{8 \cdot 18\,\text{nm}^2} \approx 694 \cdot 10^9 \text{ Bit} \approx 82\,\text{GiB}$$

$$\text{SRAM: } \frac{1\,\text{cm}^2}{100 \cdot 18\,\text{nm}^2} \approx 55 \cdot 10^9 \text{ Bit} \approx 6{,}6\,\text{GiB}$$

Die Werte für den Speicher sind jeweils mit 10 zu multiplizieren. Das Ergebnis stellt eine obere Grenze dar. Der reale Werte wird wesentlich darunter liegen, da ein enormer Platzbedarf für Ansteuerung, Signalleitungen, Energieversorgung, etc. besteht.