

Flipflops					
D-Flipflop	$Q_{t+1} = D$				das Signal wird <b>D</b> -urchgeschleift um einen Takt verzögert ausgegeben
	D	Q	$Q_{t+1}$		
	0	0	0		
	0	1	0		
	1	0	1		
1	1	1			
T-Flipflop	$Q_{t+1} = (T \wedge \neg Q) \vee (\neg T \wedge Q) = T \text{ XOR } Q$				
	T	Q	$Q_{t+1}$		
	0	0	0		
	0	1	1		
	1	0	1		
1	1	0			
J-K-Flipflop	$Q_{t+1} = (J \wedge \neg Q) \vee (\neg K \wedge Q)$				Jump and Kill Flipflop  Tel.-Nr: 0100 NAND  erst das eine und dann nicht das andere
	J	K	Q	$Q_{t+1}$	
	0	0	0	0	
	0	0	1	1	
	0	1	0	0	
	0	1	1	0	
	1	0	0	1	
	1	0	1	1	
	1	1	0	1	
1	1	1	0		
S-R-Flipflop	$Q_{t+1} = S \vee (\neg R \wedge Q)$				Set and Reset Flipflop  S&R = 1&1 is verboten oder Sylvia Riester Flipflop  ist ein JK-Flipflop mit "weiß-nicht"- "mir doch egal" Funktion bei S&R = 1&1  im KV- als Don't Care
	S	R	Q	$Q_{t+1}$	
	0	0	0	0	
	0	0	1	1	
	0	1	0	0	
	0	1	1	0	
	1	0	0	1	
	1	0	1	1	
	1	1	0	*	
1	1	1	*		
Minimierungsalgorithmen					
Karnaught-Veitch	DNF 1. Graycode-Tabelle aufstellen 2. Einsprimblöcke auslesen 3. Durch Disjunktionen verbinden 4. fertig			KNF 1. Graycode-Tabelle aufstellen 2. Nullblöcke negiert auslesen 3. durch Konjunktionen verbinden 4. fertig	

Quine-McCluskey	<ol style="list-style-type: none"> <li>1. Alle Prim-Eins-Terme aus der Funktion lesen</li> <li>2. untereinander Schreiben, und so in Blöcken zusammenfassen, das die Zeilen in den Blöcken die gleiche Anzahl an Einser besitzen</li> <li>3. Elemente benachbarter Gruppen zusammenfassen, wenn sie sich nur in einem "Literal" unterscheiden, und die strittige Stelle durch ein don't care ersetzen</li> <li>4. Zusammengefasste Zeilen markieren</li> <li>5. Neue Tabelle aufstellen, mit den neuen Zeilen, und wie unter Punkt 2 Beschrieben fortfahren.</li> <li>6. Der Algorithmus ist beendet, wenn nichts mehr zusammengefasst werden kann.</li> <li>7. alle nicht markierten Formelteile sind eventuell Teil der Minimalform</li> <li>8. Primblocktabelle aufstellen, und gucken das jede Spalte überdeckt is, ist eine vollständige Überdeckung erreicht, so haben wir die KNF wenn wir die Blöcke verunden</li> </ol>
-----------------	--

### IEEE-Gleitkommastandard

Codierung:	Single: Erstes Bit = Vorzeichen Nächste 8 Bit = Exponent (wobei: Realer Exponent = Exponent – 127) (Exponent + 127 = Exponent in der IEEE-Darstellung) Nächste 23 Bit = Mantisse (hier werden 24 Bit codiert, da die Darstellung in der Mantisse erst nach der ersten Validen Ziffer erfolgt. Die Darstellung ist also immer zu interpretieren als 1,[23Bit])	Single: 32 Bit Double: 64 Bit Extendet: 128 Bit
Kleinste darst. Zahl	1 11111111 111111111111111111111111	
Größte darst. Zahl	0 11111111 111111111111111111111111	
Betragsm. kleinste	1/0 00000000 000000000000000000000000	
1	0 01111111 000000000000000000000000	

### ETG

Reihenschaltung	Strom ist Gleich Spannungen addieren sich	
Parallelschaltung	Ströme addieren sich Spannungen sind gleich	
Ohmsches Gesetz	$U = R \cdot I$	Uri
Knotenregel	$I_1 = I_2 + I_3$	$I_1$ Strom über der Spannungsquelle $I_2 I_3$ Strohm in Netzwerkverzweigungen
Maschenregel	$U_0 = U_1 + U_2$	Die Summe der Spannungen is gleich null

Widerstände in Parallelschaltung	$R_{GES} = R_1 R_2 / (R_1 + R_2)$	Produkt durch Summe
Widerstände in Reihenschaltung	$R_{GES} = \text{Summe der Teilwiderstände}$	
Kapazität eines Kondensators	$C = Q / U$	$I = Q / t$
	$C = \epsilon_{00} * \epsilon_r * (A / d)$	
<b>MOSFET</b>		
N-Mos	ohne Kringel	
P-Mos	mit Kringel	
C-Mos	besteht oben aus P-Mos und unten aus N-Mos, (P und N-Mos führen die Gleiche Funktion aus) die beiden Ausgangssignale der Teilschaltungen werden unter einem gemeinsamen Ausgangssignal der C-Mos Schaltung vereint (ohne Gatter dazwischen, gehen direkt ins Signal)	
N-Mos zu P-Mos P-Mos zu N-Mos	alles was in Reihenschaltung ist zu Parallelschaltung machen, alles was in Parralelschaltung ist zu Reihenschaltung machen.(und Transistoren tauschen)	
<b>Pipelining</b>		
	If - Instruction Fetch (Befehl von Speicher holen) Id – Instruction Decode (Anweisung de Ex – Execute (Anweisung ausführen) Mem – Memory (Speicherzugriffe) WB – write back (Register zurück schreiben)	
Strukturkonflikt	Zwei Befehle wollen auf die Selbe Hardwarekomponente (z.B. Prozessor) gleichzeitig zugreifen	Behebbar durch zusätzliche Hardware (zweiter Prozessor oder so=
Datenkonflikt	Können auftreten, wenn ein Ergebnis eines vorrausgegangenen Befehls benötigt, dieses aber noch nicht in den Register geschrieben ist.	Behebbar mit Bypass-Hardware
Steuerungskonflikt	Kann auftreten bei Verzweigungen, da er nicht weiß welcher Befehl als nächster korrekt geladen werden müsste	Reduzierbar durch Branch-Prediction
<b>Fehlerkorrektur</b>		
Einfachfehlerkorrektur		$2^p \geq d + p + 1$   $d = \text{Datenbits}$ $p = \text{Prüfbits}$
Zweifachfehlerkorrektur		$2^{p+1} \geq 2 + (d+p)(d+p+1)$
<b>Carry-Lookahead</b>		
carry- generate	$g_i = a_i \wedge b_i$	
carry-propagate	$v_i = a_i \text{ XOR } b_i$	
carry	$c_{i+1} = g_i \vee p_i c_i$	

