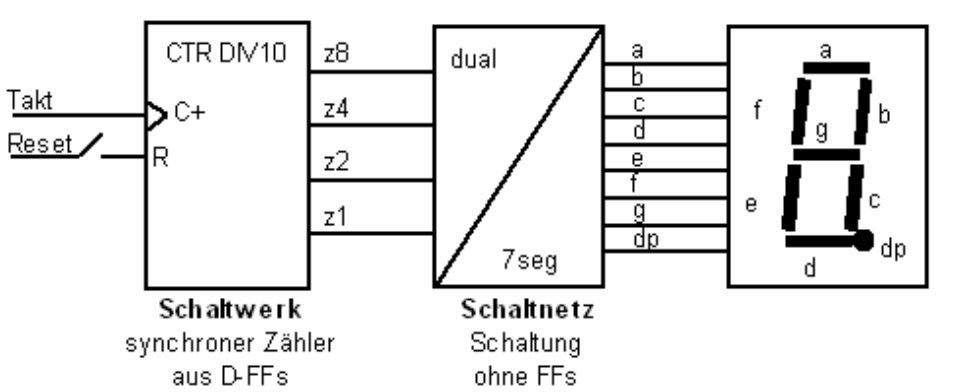
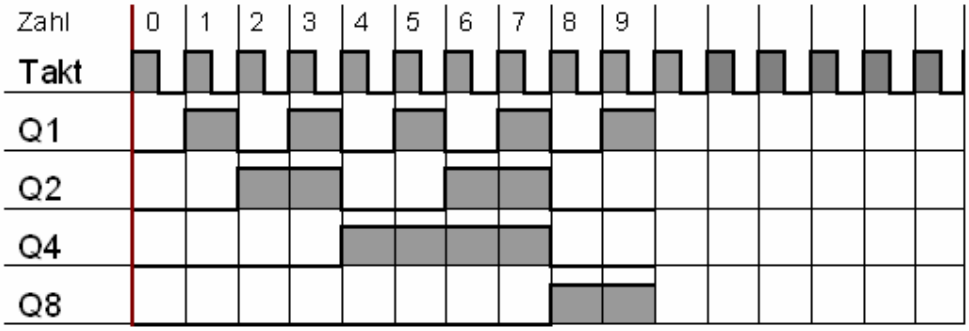
Synchroner Dezimalzähler mit Codewandler und 7-Segmentanzeige

Das Blockschaltbild zeigt einen Dezimalzähler, der mit der ansteigenden Taktflanke von 0 bis 9 zählt. Die FFs des Zählers werden synchron geschaltet, das Rücksetzen erfolgt taktunabhängig, also asynchron. Der Codewandler ist intern ohne FFs aufgebaut.

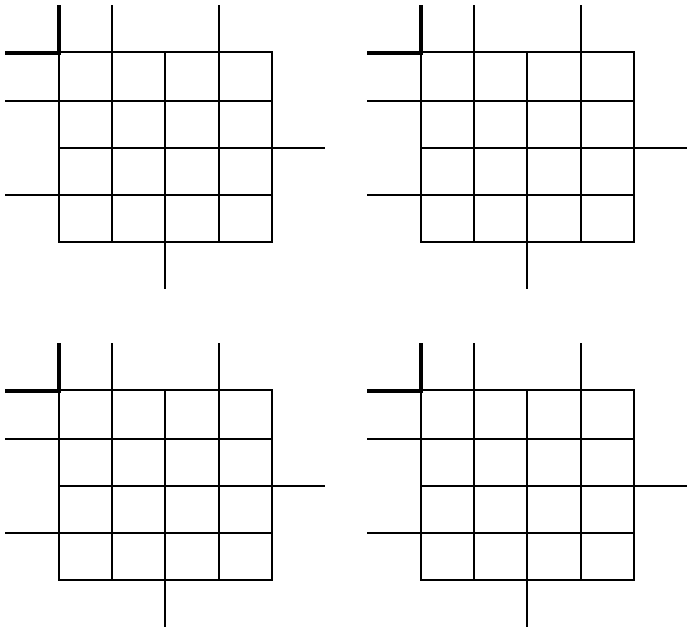


☞ **Signal-Zeitdiagramm (Ablaufdiagramm)**



☞ **Zustands-Übergangstabelle und KV-Diagramme**

					8D	4D	2D	1D
	vor dem Takt				Nach dem Takt			
Nr	Q8	Q4	Q2	Q1	Q8	Q4	Q2	Q1
0	0	0	0	0				
1	0	0	0	1				
2	0	0	1	0				
3	0	0	1	1				
4	0	1	0	0				
5	0	1	0	1				
6	0	1	1	0				
7	0	1	1	1				
8	1	0	0	0				
9	1	0	0	1				
10	1	0	1	0				
11	1	0	1	1				
12	1	1	0	0				
13	1	1	0	1				
14	1	1	1	0				
15	1	1	1	1				

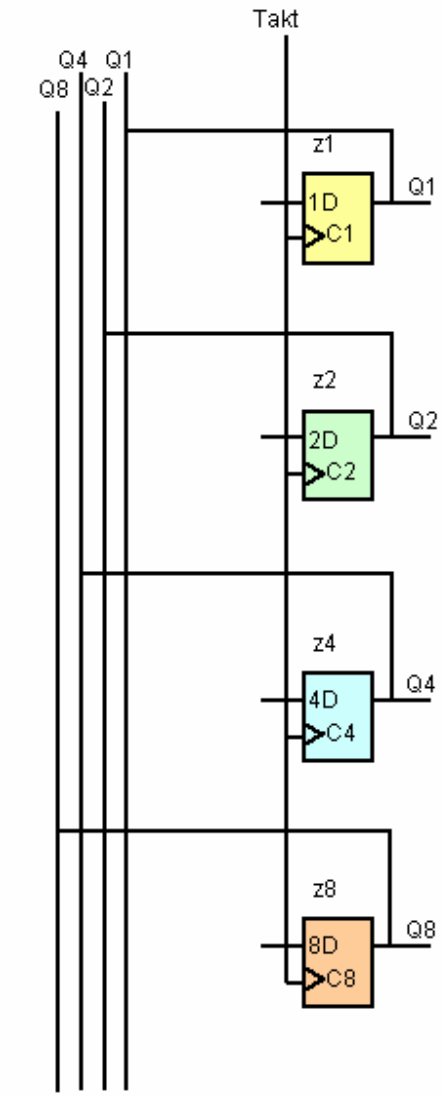


☞ Funktionsgleichungen

1D = **2D =**

4D = **8D =**

☞ Schaltung



☞ Arbeitsauftrag

Programmieren Sie die Aufgabe des Blockschaltbildes, wobei Synchronzähler und Codewandler in Form einer TRUTH_TABLE dargestellt werden sollen. Das taktunabhängige Rücksetzsignal (reset) soll mit dem Taster Pin 15 realisiert werden.

Programm-Listing des synchronen Dezimalzählers

```
MODULE syn_dezimal

TITLE 'syn_dezimal'     // Dezimalzähler mit WT , reset und 7-Seg

@dcset;                 // Compleranweisung: Don't cares berücksichtigen

declarations            // Ein- und Ausgänge

takt pin 89;            // Takteingang pin 89
reset pin 15;           // asynchroner Reset
z8, z4, z2, z1 pin 61, 60, 59, 58 istype 'buffer,reg'; //LEDs
a,b,c,d,e,f,g pin 67, 65, 66, 70, 72, 69, 71 istype 'buffer,com'; //7-Seg
x = .x.;

equations
z8.clk = takt;
z4.clk = takt;
z2.clk = takt;
z1.clk = takt;

z8.ar = reset;
z4.ar = reset;
z2.ar = reset;
z1.ar = reset;

Truth_Table             // Synchronzähler
// vor dem Takt :> nach dem Takt
 ([z8,z4,z2,z1] :> [z8,z4,z2,z1])
  [0, 0, 0, 0 ] :> [ 0, 0, 0, 1];       // 0 -> 1
  [0, 0, 0, 1 ] :> [ 0, 0, 1, 0];       // 1 -> 2
  [0, 0, 1, 0 ] :> [ 0, 0, 1, 1];       // 2 -> 3
  [0, 0, 1, 1 ] :> [ 0, 1, 0, 0];       // 3 -> 4
  [0, 1, 0, 0 ] :> [ 0, 1, 0, 1];       // 4 -> 5
  [0, 1, 0, 1 ] :> [ 0, 1, 1, 0];       // 5 -> 6
  [0, 1, 1, 0 ] :> [ 0, 1, 1, 1];       // 6 -> 7
  [0, 1, 1, 1 ] :> [ 1, 0, 0, 0];       // 7 -> 8
  [1, 0, 0, 0 ] :> [ 1, 0, 0, 1];       // 8 -> 9
  [1, 0, 0, 1 ] :> [ 0, 0, 0, 0];       // 9 -> 0


Truth_Table             // Codewandler
([z8,z4,z2,z1] -> [a, b, c, d, e, f, g])        // Ein- Ausgänge
 [0 ,0, 0, 0 ] -> [1, 1, 1, 1, 1, 1, 0];        // 0
 [0 ,0, 0, 1 ] -> [0, 1, 1, 0, 0, 0, 0];        // 1
 [0 ,0, 1, 0 ] -> [1, 1, 0, 1, 1, 0, 1];        // 2
 [0 ,0, 1, 1 ] -> [1, 1, 1, 1, 0, 0, 1];        // 3
 [0 ,1, 0, 0 ] -> [0, 1, 1, 0, 0, 1, 1];        // 4
 [0 ,1, 0, 1 ] -> [1, 0, 1, 1, 0, 1, 1];        // 5
 [0 ,1, 1, 0 ] -> [1, 0, 1, 1, 1, 1, 1];        // 6
 [0 ,1, 1, 1 ] -> [1, 1, 1, 0, 0, 0, 0];        // 7
 [1 ,0, 0, 0 ] -> [1, 1, 1, 1, 1, 1, 1];        // 8
 [1 ,0, 0, 1 ] -> [1, 1, 1, 1, 0, 1, 1];        // 9

Test_Vectors
            ([takt, reset] -> [z8, z4, z2, z1]);
@repeat 5 { [.c.,     0 ] -> [x , x , x , x ];}
@repeat 2 { [.c.,     1 ] -> [x , x , x , x ];}
@repeat 12{ [.c.,     0 ] -> [x , x , x , x ];}

END
```

Ergebnis der FUNCTIONAL-SIMULATION