CHALMERS

Department of Computer Science and Engineering

2017-01-11

EDA322 Digital konstruktion
Några uppgifter om tillståndsmaskiner
Lösningar

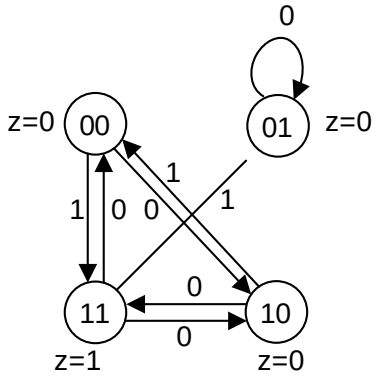
Uppgifterna är hämtade ur Brown, Vranesic: Fundamentals of Digital Logic with VHDL Design ed 3, kapitel 8.

1 Ta fram en krets uppbyggd av D-vippor som implementerar tillståndstabellen i *Figur E1*. Skriv VHDL-kod för implementeringen

Present state		*Next state*				*Output*
		w=0		*w=1*		
y_2	y_1	y_2	y_1	y_2	y_1	*z*
0	0	1	0	1	1	0
0	1	0	1	0	0	0
1	0	1	1	0	0	0
1	1	1	0	0	1	1

Figur E1 Tillståndstabell

Vi ritar en tillståndsgraf, *Figur S1a*



Figur S1a Tillståndgraf från Figur E1

Vi skall implementera med hjäp av D-vippor så vi tecknar D-vippans tillståndstabell, *Figur S1b*

Present state	*D-flipflop* *D*	*Next state*
0	0	0
0	1	1
1	0	0
1	1	1

Figur S1b D-vippans tillståndstabell

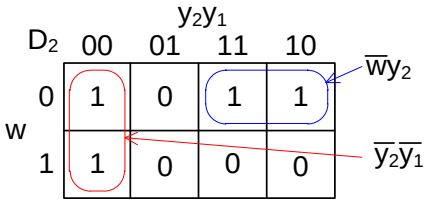
Vi inför D-vippor i tillståndtabellen utifrån D-vippans sanningstabell i *Figur S1a* och får*, Figur S1c*

1 forts.

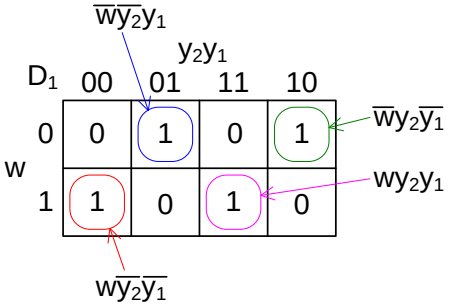
Present state		*D flipflops*				*Next state*				*Output*
		w=0		*w=1*		*w=0*		*w=1*		
Q_2	Q_1	D_2	D_1	D_2	D_1	Q_2	Q_1	Q_2	Q_1	*z*
0	0	1	0	1	1	1	0	1	1	0
0	1	0	1	0	0	0	1	0	0	0
1	0	1	1	0	0	1	1	0	0	0
1	1	0	0	0	1	1	0	0	1	1

Figur S1c Tillståndstabell med D-vippor

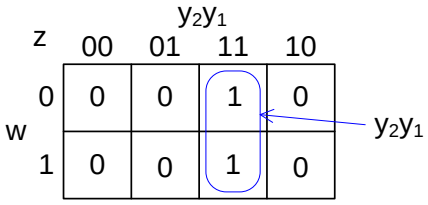
Vi använder Karnaughdiagram för att bestämma villkoren för D-ingångarna, *Figur S1d-f*



Figur S1d Karnaughdiagram för D_2



Figur S1e Karnaughdiagram för D_1



Figur S1f Karnaughdiagram för z

Vi får

$$D_2 = \overline{y_2} \cdot \overline{y_1} + \overline{w} \cdot y_2$$

$$D_1 = w \cdot y_2 \cdot y_1 + w \cdot \overline{y_2} \cdot \overline{y_1} + \overline{w} \cdot y_2 \cdot \overline{y_1} + \overline{w} \cdot \overline{y_2} \cdot x_1 = w \oplus y_2 \oplus y_1$$

$$z = y_2 \cdot y_1$$

Låt oss skriva VHDL-kod och testa resultatet.
Vi börjar med att skapa en D-flipflop som vi kan använda som komponent.

1 forts.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY D_flipflop IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        D:IN STD_LOGIC;
        Q:OUT STD_LOGIC);
END D_flipflop;

ARCHITECTURE arch_D_flipflop OF
                        D_flipflop IS
BEGIN
   D_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         Q <= '0';
      ELSIF rising_edge(Clock) THEN
         IF (D='1') THEN
            Q <= '1';
         ELSE
            Q <= '0';
         END IF;
      END IF;
   END PROCESS D_proc;
END arch_D_flipflop;
```

Vi simulerar ned en do-fil

```
restart -f -nowave
view signals wave
add wave Clock Resetn D Q
force Clock 0 0, 1 50ns -repeat 100ns
force D 0
force Resetn 0
run 225ns
force Resetn 1
run 200ns
force D 1
run 200ns
force D 0
run 200ns
force D 1
run 200ns
force Resetn 0
```

1 forts.

```
run 200ns
force Resetn 1
run 200ns
```

Nu kan vi skriva kod för tillståndsmaskinen

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S1 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S1;

ARCHITECTURE arch_S1 OF S1 IS
   COMPONENT D_flipflop IS
      PORT(Clock:IN STD_LOGIC;
           Resetn:IN STD_LOGIC;
           D:IN STD_LOGIC;
           Q:OUT STD_LOGIC);
   END COMPONENT D_flipflop;
   SIGNAL D1_signal:STD_LOGIC;
   SIGNAL D2_signal:STD_LOGIC;
   SIGNAL Q1_signal:STD_LOGIC;
   SIGNAL Q2_signal:STD_LOGIC;
   SIGNAL D_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
   SIGNAL Q_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
   D_flipflop_comp_1:
   D_flipflop
   PORT MAP(Clock => Clock,
            Resetn =>Resetn,
            D => D1_signal,
            Q => Q1_signal);

   D_flipflop_comp_2:
   D_flipflop
   PORT MAP(Clock => Clock,
            Resetn =>Resetn,
            D => D2_signal,
            Q => Q2_signal);
```

1 forts.

```
    D1_signal <= w XOR (Q2_signal XOR Q1_signal);
    D2_signal <= (NOT(Q2_signal) AND NOT(Q1_signal)) OR
                 (NOT(w) AND Q2_signal);
    D_vector_signal <= D2_signal & D1_signal;
    Q_vector_signal <= Q2_signal & Q1_signal;

    z <= Q2_signal AND Q1_signal;

END arch_S1;
```

I koden har vi lagt in vektorerna `D_vector_signal` och `Q_vector_signal` för att lättare se tillstånden.
Vi simulerar med en ny `do`-fil

```
restart -f -nowave
view signals wave
add wave Clock Resetn w D2_signal D1_signal Q2_signal
add wave Q1_signal -radix binary D_vector_signal Q_vector_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force w 0
force Resetn 0
run 225ns
force Resetn 1
run 400ns
force w 1
run 400ns
force w 0
run 400ns
```

---

2 Upprepa *uppgift 1* med hjälp av JK-vippor

Vi har samma tillståndstabell som i uppgift 1 men ska nu implementera med JK-vippor.
Figur S2a ger JK-vippornas tillståndstabell

Present state	*JK-flipflop*		*Next state*
	J	*K*	
0	0	x	0
0	1	x	1
1	x	1	0
1	x	0	1

Figur S2a JK-vippans tillståndstabell

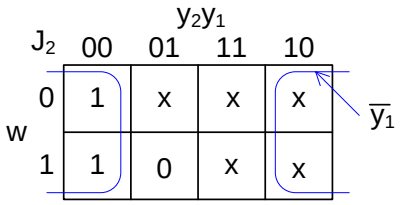
2 forts.

Vi kompletterar nu tillståndstabellen i *Figur E1* med JK-vippor, Figur S2b

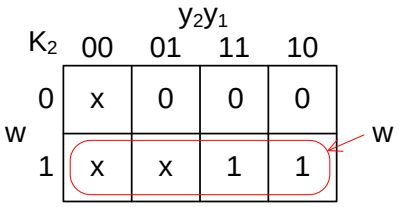
Present state		*D flipflops*								*Next state*				*Output*
		w=0				*w=1*				*w=0*		*w=1*		
Q_2	Q_1	J_2	K_2	J_1	K_1	J_2	K_2	J_1	K_1	Q_2	Q_1	Q_2	Q_1	z
0	0	1	x	0	x	1	x	1	x	1	0	1	1	0
0	1	x	0	x	0	0	x	x	1	0	1	0	0	0
1	0	x	0	1	x	x	1	0	x	1	1	0	0	0
1	1	x	0	x	1	x	1	x	0	1	0	0	1	1

Figur S2b Tillståndstabell med D-vippor

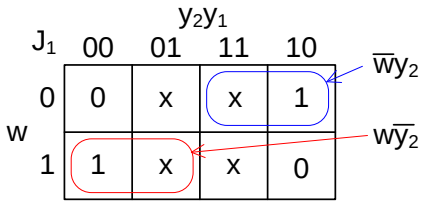
Vi använder Karnaughdiagram för att ta fram de logiska villkoren, *Figur S2c-f*



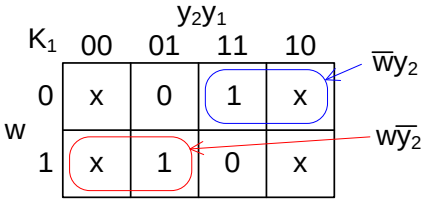
Figur S2c Karnaughdiagram för J_2



Figur Sd Karnaughdiagram för K_2



Figur S2e Karnaughdiagram för J_1



Figur S2f Karnaughdiagram för K_1

Vi får

$$J_2 = \overline{y_1}$$

$$K_2 = w$$

$$J_1 = w \cdot \overline{y_2} + \overline{w} \cdot y_2 = w \oplus y_2$$

$$K_1 = J_1 = w \cdot \overline{y_2} + \overline{w} \cdot y_2 = w \oplus y_2$$

Utsignalen z påverkas inte av vilka vippor vi väljer utan blir den samma som i *exempel 1* dvs

2 forts.

$z = x_2 \cdot x_1$

Låt oss åter skriva VHDL-kod och börjar då med en JK-vippa.

```
ENTITY JK_flipflop IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        J:IN STD_LOGIC;
        K:IN STD_LOGIC;
        Q:OUT STD_LOGIC);
END JK_flipflop;

ARCHITECTURE arch_JK_flipflop OF JK_flipflop IS
   SIGNAL Q_signal:STD_LOGIC;
   SIGNAL JK_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
   JK_signal <= J&K;
   JK_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         Q_signal <= '0';
      ELSIF rising_edge(Clock) THEN
         CASE JK_signal IS
            WHEN "00" =>
               Q_signal <= Q_signal;
            WHEN "01" =>
               Q_signal <= '0';
            WHEN "10" =>
               Q_signal <= '1';
            WHEN "11" =>
               Q_signal <= NOT(Q_signal);
            WHEN OTHERS =>
         END CASE;
      END IF;
   END PROCESS JK_proc;
   Q <= Q_signal;
END arch_JK_flipflop;
```

Som vi simulerar med en do-fil.

```
restart -f -nowave
view signals wave
add wave Clock Resetn J K
add wave -radix binary JK_signal Q_signal Q
force Clock 0 0, 1 50ns -repeat 100ns
```

2 forts.

```
force J 0
force K 0
force Resetn 0
run 225ns
force Resetn 1
run 200ns
force J 1
run 200ns
force K 1
run 200ns
force J 0
run 200ns
force Resetn 0
run 200ns
force Resetn 1
run 200ns
```

och nu över till tillståndsmaskinen

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S2 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S2;

ARCHITECTURE arch_S2 OF S2 IS
   COMPONENT JK_flipflop IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        J:IN STD_LOGIC;
        K:IN STD_LOGIC;
        Q:OUT STD_LOGIC);
   END COMPONENT JK_flipflop;
   SIGNAL J1_signal:STD_LOGIC;
   SIGNAL K1_signal:STD_LOGIC;
   SIGNAL J2_signal:STD_LOGIC;
   SIGNAL K2_signal:STD_LOGIC;
   SIGNAL Q1_signal:STD_LOGIC;
   SIGNAL Q2_signal:STD_LOGIC;
   SIGNAL JK1_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
   SIGNAL JK2_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
   SIGNAL Q_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
```

2 forts.

```
BEGIN
   JKflipflop_comp_1:
   JK_flipflop
   PORT MAP(Clock => Clock,
            Resetn =>Resetn,
            J => J1_signal,
            K => K1_signal,
            Q => Q1_signal);

   JK_flipflop_comp_2:
   JK_flipflop
   PORT MAP(Clock => Clock,
            Resetn =>Resetn,
            J => J2_signal,
            K => K2_signal,
            Q => Q2_signal);

   J2_signal <= NOT(Q1_signal);
   K2_signal <= w;
   J1_signal <= w XOR Q2_signal;
   K1_signal <= J1_signal;
   JK1_vector_signal <= J1_signal & K1_signal;
   JK2_vector_signal <= J2_signal & K2_signal;
   Q_vector_signal <= Q2_signal & Q1_signal;
   z <= Q2_signal AND Q1_signal;
END arch_S2;
```

Vi har åter knutit ihop signaler i vektorer för att lättare se dom i simuleringen som vi gör med en do-fil.

```
restart -f -nowave
view signals wave
add wave Clock Resetn w J1_signal K1_signal
add wave J2_signal K2_signal Q1_signal Q2_signal
add wave -radix binary JK1_vector_signal
add wave Q_vector_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force w 0
force Resetn 0
run 225ns
force Resetn 1
run 400ns
force w 1
run 400ns
force w 0
run 400ns
```

3 Upprepa *uppgift 1* med hjälp av T-vippor

Vi skall implementera med hjäp av T-vippor så vi teck-nar T-vippans tillståndstabell, *Figur S3a*

Present state	*T-flipflop* *D*	*Next state*
0	0	0
0	1	1
1	0	1
1	1	0

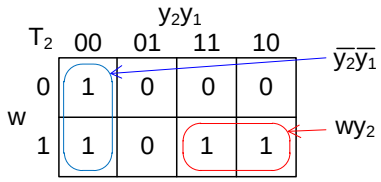
Figur S3a T-vippans tillståndstabell

Vi inför D-vippor i tillståndtabellen utifrån D-vippans sanningstabell i *Figur E1* och får, *Figur S3b*

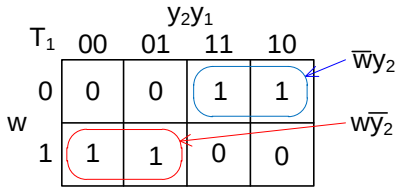
Present state		*T flipflops*				*Next state*				*Output*
		w=0		*w=1*		*w=0*		*w=1*		
Q_2	Q_1	T_2	T_1	T_2	T_1	Q_2	Q_1	Q_2	Q_1	*z*
0	0	1	0	1	1	1	0	1	1	0
0	1	0	0	0	1	0	1	0	0	0
1	0	0	1	1	0	1	1	0	0	0
1	1	0	1	1	0	1	0	0	1	1

Figur S3b Tillståndstabell med D-vippor

Vi använder Karnaughdiagram för att bestämma villkoren för D-ingångarna, *Figur S1c-e*



Figur S3c Karnaughdiagram för T_2



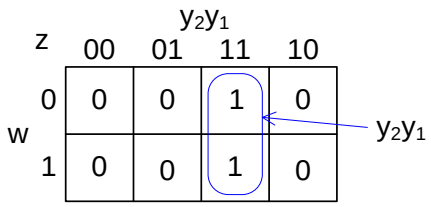
Figur S3d Karnaughdiagram för T_1

Vi får

$$T_2 = \overline{y_2} \cdot \overline{y_1} + w \cdot y_2$$

$$T_1 = \overline{w} \cdot y_2 + w \cdot \overline{y_2}$$

$$z = y_2 \cdot y_1$$



Figur S3e Karnaughdiagram för z

Låt oss åter skriva VHDL-kod och börjar då med en T-vippa.

3 forts.

```
-- T_flipflop.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY T_flipflop IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        T:IN STD_LOGIC;
        Q:OUT STD_LOGIC);
END T_flipflop;

ARCHITECTURE arch_T_flipflop OF T_flipflop IS
   SIGNAL Q_signal:STD_LOGIC;
BEGIN
   T_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         Q_signal <= '0';
      ELSIF rising_edge(Clock) THEN
         IF (T='1') THEN
            Q_signal <= NOT(Q_signal);
         ELSE
            Q_signal <= Q_signal;
         END IF;
      END IF;
   END PROCESS T_proc;
   Q <= Q_signal;
END arch_T_flipflop;
```

som vi simulerar med en do-fil

```
-- T_flipflop.do
restart -f -nowave
view signals wave
add wave Clock Resetn T Q_signal Q
force Clock 0 0, 1 50ns -repeat 100ns
force T 0
force Resetn 0
run 225ns
force Resetn 1
run 200ns
force T 1
run 200ns
force T 0
```

3 forts.

```
run 200ns
force T 1
run 200ns
force Resetn 0
run 200ns
force Resetn 1
run 200ns
```

och nu över till tillståndsmaskinen

```
-- S3.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S3 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S3;

ARCHITECTURE arch_S3 OF S3 IS
   COMPONENT T_flipflop IS
      PORT(Clock:IN STD_LOGIC;
           Resetn:IN STD_LOGIC;
           T:IN STD_LOGIC;
           Q:OUT STD_LOGIC);
   END COMPONENT T_flipflop;
   SIGNAL T1_signal:STD_LOGIC;
   SIGNAL T2_signal:STD_LOGIC;
   SIGNAL Q1_signal:STD_LOGIC;
   SIGNAL Q2_signal:STD_LOGIC;
   SIGNAL T_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
   SIGNAL Q_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
   T_flipflop_comp_1:
   T_flipflop
   PORT MAP(Clock => Clock,
            Resetn =>Resetn,
            T => T1_signal,
            Q => Q1_signal);
```

3 forts.

```
    T_flipflop_comp_2:
    T_flipflop
    PORT MAP(Clock => Clock,
             Resetn =>Resetn,
             T => T2_signal,
             Q => Q2_signal);

    T1_signal <= (NOT(w) AND Q2_signal) OR
                 (w AND NOT(Q2_signal));
    T2_signal <= (NOT(Q2_signal) AND NOT(Q1_signal)) OR
                 (w AND Q2_signal);
    T_vector_signal <= T2_signal & T1_signal;
    Q_vector_signal <= Q2_signal & Q1_signal;
    z <= Q2_signal AND Q1_signal;

END arch_S3;
```

som vi simulerar med en do-fil

```
-- ex8_1_logic_T.do
restart -f -nowave
view signals wave
add wave Clock Resetn w T2_signal T1_signal Q2_signal
add wave Q1_signal -radix binary T_vector_signal Q_vector_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force w 0
force Resetn 0
run 225ns
force Resetn 1
run 400ns
force w 1
run 400ns
force w 0
run 400ns
```

---

4 Lös *uppgift 1* med beteendemässig VHDL-kod

---

4 forts.

Då vi skall använda beteendemässig kod så återgår vi till vår ursprungliga tillståndstabell som vi upprepar i *Figur S4a*. Vi ersätter tillståndsvariablerna y_2 och y_1 med tillståndsnamn*, Figur S4b*.
Om vi använder tre processer så får vi koden

Present state		*Next state*				*Output*
		w=0		*w=1*		
y_2	y_1	y_2	y_1	y_2	y_1	*z*
0	0	1	0	1	1	0
0	1	0	1	0	0	0
1	0	1	1	0	0	0
1	1	1	0	0	1	1

Figur S4a Tillståndstabell

Tillstånds-variabler		*Tillstånd*
y_2	y_1	
0	0	A
0	1	B
1	0	C
1	1	D

Figur S4b Tillstånds-tilldelning

```
-- S4.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S4 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S4;

ARCHITECTURE arch_S4 OF S4 IS
   TYPE state_type IS (A,B,C,D);
   SIGNAL state_signal:state_type;
   SIGNAL next_state_signal:state_type;
BEGIN
   state_transition_proc:
      PROCESS(Resetn,Clock)
      BEGIN
         IF (Resetn='0') THEN
            state_signal<=A;
         ELSIF rising_edge(Clock) THEN
            state_signal<=next_state_signal;
         END IF;
   END PROCESS state_transition_proc;

   stateflow_proc:
   PROCESS(state_signal,w)
   BEGIN
      CASE state_signal IS
         WHEN A =>
            IF w = '1' THEN
               next_state_signal <= D;
            ELSE
               next_state_signal <= C;
            END IF;
         WHEN B =>
            IF w = '1' THEN
               next_state_signal <= B;
```

4 forts.

```
               ELSE
                  next_state_signal <= B;
                  END IF;
            WHEN C =>
               IF w = '1' THEN
                  next_state_signal <= A;
               ELSE
                  next_state_signal <= D;
               END IF;
            WHEN D =>
               IF w = '1' THEN
                  next_state_signal <= B;
               ELSE
                  next_state_signal <= C;
               END IF;
         END CASE;
   END PROCESS stateflow_proc;

   assignment_proc:
   PROCESS(state_signal,w)
   BEGIN
      IF (state_signal=D) THEN
         z <= '1';
      ELSE
         z <= '0';
      END IF;
   END PROCESS assignment_proc;
END arch_S4;
```

med do-filen

```
-- S4.do
restart -f -nowave
view signals wave
add wave Clock Resetn w
add wave state_signal next_state_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force w 0
force Resetn 0
run 225ns
force Resetn 1
force w 0
run 100ns
force w 1
run 100ns
force w 0
run 100ns
```

4 forts.

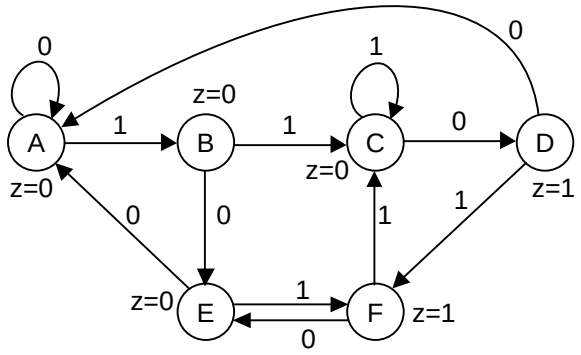
```
force w 1
run 400ns
force w 0
run 200ns
force w 1
run 200ns
force w 0
run 200ns
force w 1
run 400ns
```

---

5 Ta fram en tillståndsdiagrammet för en tillståndsmaskin av Mooretyp som ger utsignalen *z* värdet ett (1) om vi via ingången *w* har detekterat någon av sekvenserna 110 eller 101. Sekvenserna kan vara överlappande

---

Vi skall få en etta (1) ut från vår tillståndsmaskin om vi har identifierat någon av serierna 110 eller 101 hos insignalen. Serierna får vara överlappande. Vi ritar tillståndsgraf för en lämplig Mooremaskin, *Figur S5a* och tecknar tillståndstabellen, *Figur S5b*.



Figur S5a Tillståndsgraf för Mooremaskin

Låt oss skriva VHDL-kod

```
-- ex8_5.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S5 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S5;

ARCHITECTURE arch_S5 OF S5 IS
   TYPE state_type IS (A,B,C,D,E,F);
   SIGNAL state_signal:state_type;
   SIGNAL nextState_signal:state_type;
BEGIN
   state_transition_proc:
```

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	A	B	0
B	E	C	0
C	D	C	0
D	A	F	1
E	A	F	0
F	E	C	1

Figur S5b Tillståndstabell för Mooremaskin

5 forts.

```
  PROCESS(Resetn,Clock)
    BEGIN
       IF (Resetn='0') THEN
          state_signal<=A;
       ELSIF rising_edge(Clock) THEN
          state_signal<=nextState_signal;
       END IF;
    END PROCESS state_transition_proc;

    flow_proc:
    PROCESS(state_signal,w)
    BEGIN
       CASE state_signal IS
           WHEN A =>
             IF w = '1' THEN
                nextState_signal <= B;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN B =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
          WHEN C =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= D;
             END IF;
           WHEN D =>
             IF w = '1' THEN
                nextState_signal <= F;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN E =>
             IF w = '1' THEN
                nextState_signal <= F;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN F =>
             IF w = '1' THEN
                nextState_signal <= C;
```

5 forts.

```
            ELSE
               nextState_signal <= E;
            END IF;
         WHEN OTHERS =>
      END CASE;
   END PROCESS flow_proc;

   signal_assignment_proc:
   PROCESS(state_signal)
   BEGIN
      IF ((state_signal=D) OR
          (state_signal=F)) THEN
         z <= '1';
      ELSE
         z <= '0';
      END IF;
   END PROCESS signal_assignment_proc;
END arch_S5;
```

Vi skriver en do-fil för simulering

```
-- S5.do
restart -f -nowave
view signals wave
add wave Clock Resetn w state_signal nextState_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force w 0
force Resetn 0

run 225ns
force Resetn 1
force w 0
run 100ns
force w 1
run 100ns
#1
force w 0
run 100ns
#10
force w 1
run 400ns
#101111
force w 0
run 100ns
#1011110->1
force w 1
```

5 forts.

```
run 200ns
#101111011
force w 0
run 100ns
#1011110110->1
force w 1
run 400ns
#101111011001111->1000
```

---

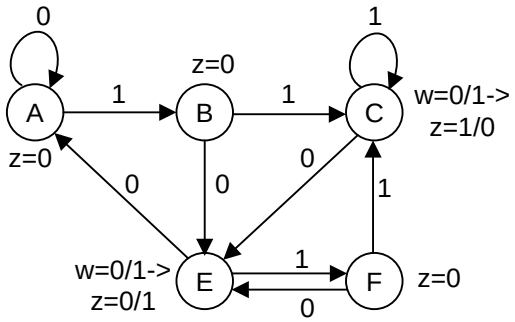
6 Upprepa *uppgift 5* med hjälp av en tillståndsmaskin av Mealytyp

---

Vi ritar tillståndsgraf, *Figur S6a* och översätter till tillståndstabell, *Figur S6b*

Preset state	*Next state*		*Output z*	
	w=0	*w=1*	*w=0*	*w=1*
A	A	B	0	0
B	E	C	0	0
C	E	C	1	0
E	A	F	0	1
F	E	C	0	0

Figur S6b Tillståndstabell för Mealymaskin



Figur S6a Tillståndsgraf för Mealymaskin

Vi skriver VHDL-kod

```
-- S6a.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S6a IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S6a;

ARCHITECTURE arch_ S6a OF S6a IS
   TYPE state_type IS (A,B,C,E,F);
   SIGNAL state_signal:state_type;
   SIGNAL
nextState_signal:state_type;
BEGIN
   state_transition_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         state_signal<=A;
      ELSIF rising_edge(Clock) THEN
         state_signal<=nextState_signal;
      END IF;
   END PROCESS state_transition_proc;

   flow_proc:
```

6 forts.

```
 PROCESS(state_signal,w)
    BEGIN
       CASE state_signal IS
           WHEN A =>
             IF w = '1' THEN
                nextState_signal <= B;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN B =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
          WHEN C =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
           WHEN E =>
             IF w = '1' THEN
                nextState_signal <= F;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN F =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
          WHEN OTHERS =>

       END CASE;
    END PROCESS flow_proc;

    signal_assignment_proc:
    PROCESS(state_signal,w)
    BEGIN
       IF (((state_signal=C) AND (w='0'))OR
           ((state_signal=E) AND (w='1'))) THEN
          z <= '1';
       ELSE
          z <= '0';
       END IF;
```

5 forts.

```
   END PROCESS signal_assignment_proc;
END arch_ S6a;
```

Vi ser ur tabellen i *Figur S66b* att tillstånd *B* och *F* är likadana varför vi kan slå ihop dem till ett enda tillstånd. Vi behåller tillstånd *B* och tar bort tillstånd *E* och får *Figur S6c* som ger tillståndsgrafen i *Figur S6d* som vi åter skriver VHDL-kod för

Present state	*Next state*		*Output z*	
	w=0	*w=1*	*w=0*	*w=1*
A	A	B	0	0
B	E	C	0	0
C	E	C	1	0
E	A	B	0	1

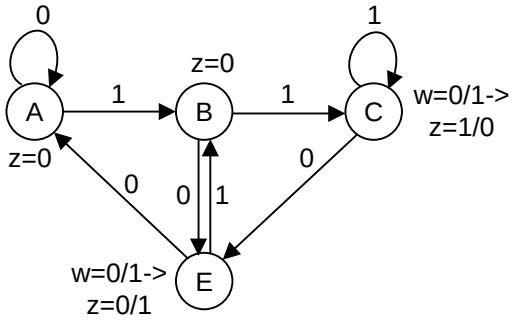
Figur S6c Modifierad tillståndsgraf för Mealymaskin

```
-- St.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY S6 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S6;

ARCHITECTURE arch_S6 OF 6 IS
   TYPE state_type IS (A,B,C,E);
   SIGNAL
state_signal:state_type;
   SIGNAL
nextState_signal:state_type;
BEGIN
   state_transition_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         state_signal<=A;

      ELSIF rising_edge(Clock) THEN
         state_signal<=nextState_signal;
      END IF;
   END PROCESS state_transition_proc;
```



Figur S6cdModifierad tillståndsgraf för Mealymaskin

6 forts.

```
    flow_proc:
    PROCESS(state_signal,w)
    BEGIN
       CASE state_signal IS
           WHEN A =>
             IF w = '1' THEN
                nextState_signal <= B;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN B =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
          WHEN C =>
             IF w = '1' THEN
                nextState_signal <= C;
             ELSE
                nextState_signal <= E;
             END IF;
          WHEN E =>
             IF w = '1' THEN
                nextState_signal <= B;
             ELSE
                nextState_signal <= A;
             END IF;
          WHEN OTHERS =>

       END CASE;
    END PROCESS flow_proc;

    signal_assignment_proc:
    PROCESS(state_signal,w)
    BEGIN
       IF (((state_signal=C) AND (w='0')) OR
           ((state_signal=E) AND (w='1'))) THEN
          z <= '1';
       ELSE
          z <= '0';
       END IF;
    END PROCESS signal_assignment_proc;
 END arch_ex8_6_v2;
```

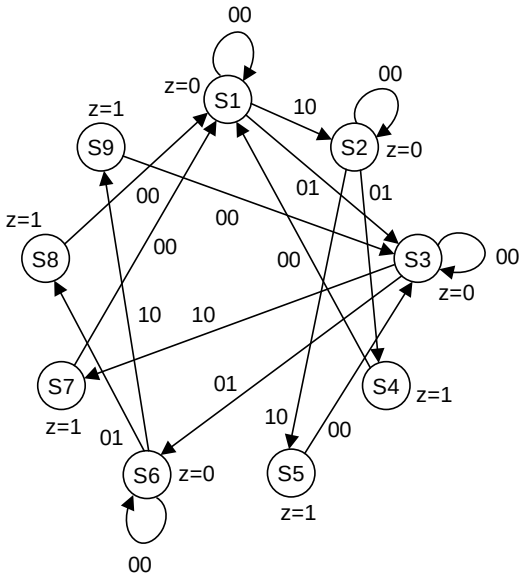
Vi kan åter simulera med samma do-fil som i *exempel S5*.

7 Minimera tillståndstabellen i *Figur E7* med de två insignalerna *D* och *N* och utsignalen *z*

Present state	*Next state*				*Output*
	DN				
	00	*01*	*10*	*11*	*z*
S1	S1	S3	S2	-	0
S2	S2	S4	S5	-	0
S3	S3	S6	S7	-	0
S4	S1	-	-	-	1
S5	S3	-	-	-	1
S6	S6	S8	S9		0
S7	S1	-	-	-	1
S8	S1	-	-	-	1
S9	S3	-	-	-	1

Figur E7

Vi ritar tillstådsgraf*, Figur S7a*



Figur S7a Tillståndsgraf för Figur E7

7 forts.

Vi börjar med att ställa upp en tabell över alla möjliga tillståndsekvivalenter, *Figur S7b*

S2								
S3								
S4								
S5								
S6								
S7								
S8								
S9								
	S1	S2	S3	S4	S5	S6	S7	S8

Figur S7b Tabell 1 för tillståndsminimering med alla möjliga tillståndsekvivalenter

I tabellen dimmar vi alla tillståndsekvivalenter som är omöjliga på grund av att tillstånden har olika utsignaler, *Figur S7c*

S2								
S3								
S4								
S5								
S6								
S7								
S8								
S9								
	S1	S2	S3	S4	S5	S6	S7	S8

Figur S7c Tabell 2 för tillståndsminimering

För att två tillstånd skall vara ekvivalenta så måste även de efterföljande tillstånden, dvs de tillstånd till vilka vi går från aktuellt tillstånd vara ekvivalenta. Vi skriveri de odimmade rutorna in de par av efterföljande tillstånd som måste vara ekvivalenta för att de två aktuella tillstånden skall vara ekvivalenta. Ingår de två aktuella tillstånden i paret av efterföljande tillstånd så behöver vi inte skriva in dem eftersom det är dessa som vi vill slå ihop. Vi behöver inte heller skriva in ett par som redan är en ekvivalens, dvs är ett och samma tillstånd, *Figur S7d*.

S2	S3,S4 S2,S5							
S3	S3,S6 S2,S7	S4,S6 S5,S7						
S4								
S5				S1,S3				
S6	S3,S8 S2,S9	S4,S8 S5,S9	S6,S8 S7,S9					
S7					S1,S3			
S8					S1,S3			
S9				S1,S3			S1,S3	S1,S3
	S1	S2	S3	S4	S5	S6	S7	S8

Figur S7d Tabell 2 för tillståndsminimering

7 forts.

Vi fortsätter genom att dimma ut ekvivalenter där vi har skrivit in tillståndspar som vi redan har dömt som inte ekvivalenta genom att vi har dimmat ut dem, *Figur S7e*.

S2	S3,S4 S2,S5							
S3	S3,S6 S2,S7		S4,S6 S5,S7					
S4								
S5					S1,S3			
S6	S3,S8 S2,S9		S4,S8 S5,S9	S6,S8 S7,S9				
S7						S1,S3			
S8						S1,S3			
S9					S1,S3			S1,S3	S1,S3
	S1		S2	S3	S4	S5	S6	S7	S8

Figur S7e Tabell 2 för tillståndsminimering

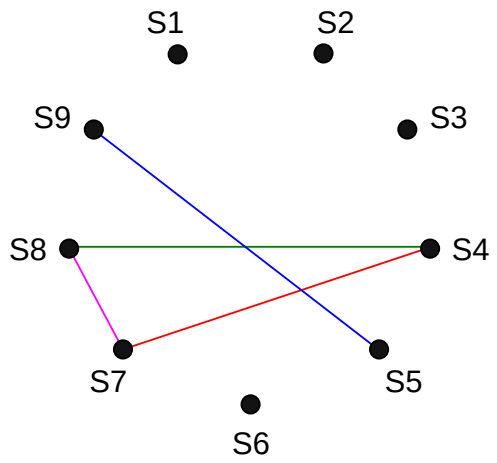
Detta har nu givit upphov till nya dimningar, dvs tillstånd som inte kan vara ekvivalenter och vi kan upprepa föregående uteslutning av ekvivalenter med hjälp av de nya dimningarna, *Figur S7f*.Vi kommer inte längre och vi ritar relationsgraf, *Figur S7g*.

S2	S3,S4 S2,S5							
S3	S3,S6 S2,S7	S4,S6 S5,S7						
S4								
S5				S1,S3				
S6	S3,S8 S2,S9	S4,S8 S5,S9	S6,S8 S7,S9					
S7						S1,S3			
S8						S1,S3			
S9				S1,S3				S1,S3	S1,S3
	S1	S2	S3	S4		S5	S6	S7	S8

Figur S7f Tabell 2 för tillståndsminimering

Vi ser att vi kan slå ihop tillstånd S4, S7 och S8 samt tillstånd S5 och S9. Låt oss behålla tillstånd S4 och S5. Efter dessa sammanslagningar ser vi också att vi kan slå ihop tillstånd S2 och S6, vi behåller S2 och tecknar den minimerade tillståndstabellen, *Figur S7h*.

Låt oss först rita upp den nya tillståndsgrafen, *Figur S7i*

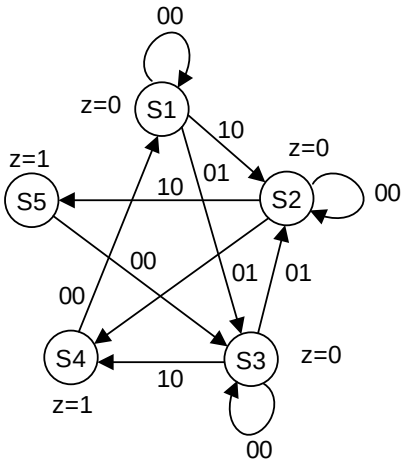


Figur S7g Relationsgraf

7 forts.

Present state	*Next state* DN				*Output*
	00	01	10	11	z
S1	S1	S3	S2	-	0
S2	S2	S4	S5	-	0
S3	S3	S2	S4	-	0
S4	S1	-	-	-	1
S5	S3	-	-	-	1

Figur S7h Minimerad tillståndstabell



Figur S7i Minimerad tillståndsgraf

Låt oss göra en implementering med D-vippor

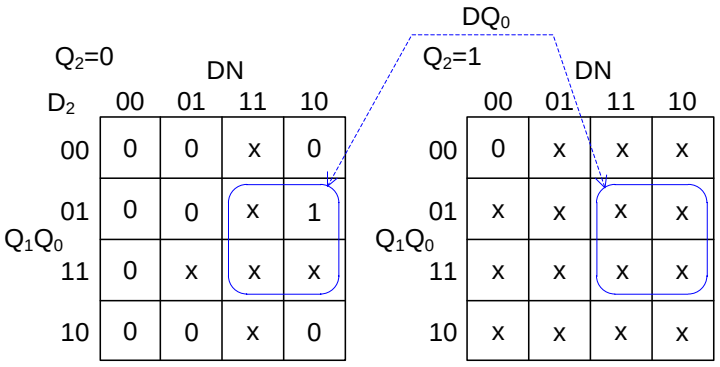
Vi ger tillstånden värden samt inför D-vippor, *Figur S7j*

Present state	*D-flipflops* $D_2D_1D_0$				*Next state* $Q_2Q_1Q_0$				*Output*
	DN				DN				
$Q_2Q_1Q_0$	00	01	10	11	00	01	10	11	z
000	000	010	001	---	000	010	001	---	0
001	001	011	100	---	001	011	100	---	0
010	010	001	011	---	010	001	011	---	0
011	000	---	---	---	000	---	---	---	1
100	010	---	---	---	010	---	---	---	1

Figur S7j Tillståndstabell

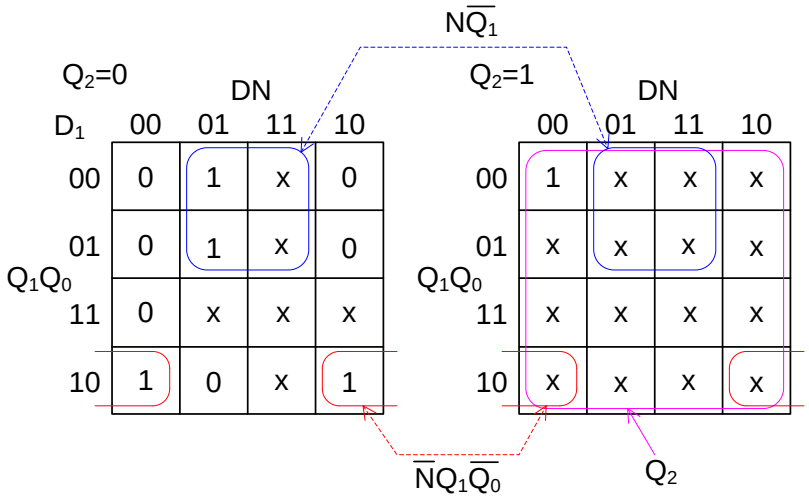
Här har vi fem variabler så det är hanterligt att logik-minimera med hjälp av Karnaughdiagram, *Figur S7k-n*

$$D_2 = D \cdot Q_0$$



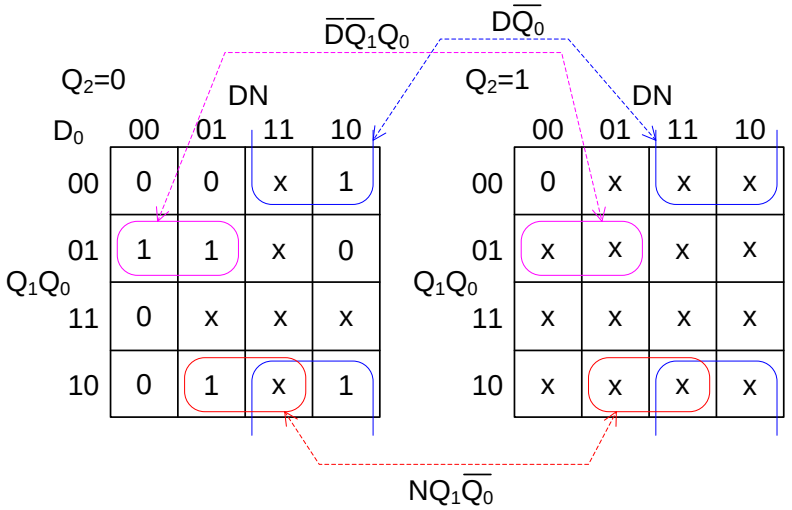
Figur S7k Karnaughdiagram för D_2

7 forts.



Figur S7l Karnaughdiagram för D_1

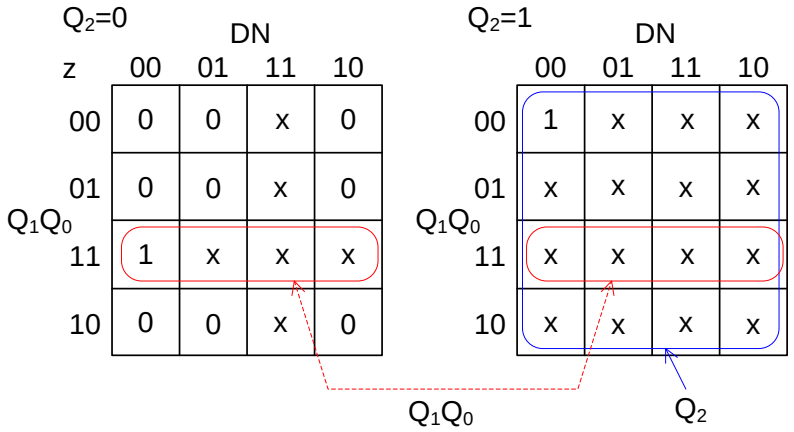
$$D_1 = Q_2 + N \cdot \overline{Q_1} + \overline{N} \cdot Q_1 \cdot \overline{Q_0}$$



Figur S7m Karnaughdiagram för D_0

$$D_0 = D \cdot \overline{Q_0} + N \cdot Q_1 \cdot \overline{Q_0} + \overline{D} \cdot \overline{Q_1} \cdot Q_0$$

7 forts.



$$z = Q_2 + Q_1 \cdot Q_0$$

Figur S7n Karnaughdiagram för utsignalen z

För att kunna testa kretsen så måste vi se vad våra logik har gjort av don´t care-tillstånden. Vi uppdaterar tillståndstabellen, *Figur S7o*.

Present state	*D-flipflops*				*Next state*				*Output*
	$D_2D_1D_0$				$Q_2Q_1Q_0$				
	DN				DN				
$Q_2Q_1Q_0$	00	01	10	11	00	01	10	11	z
000	000	010	001	011	000	010	001	011	0
001	001	011	100	110	001	011	100	110	0
010	010	001	011	001	010	001	011	001	0
011	000	000	100	100	000	000	100	100	1
100	010	010	011	011	010	010	011	011	1
101	011	011	110	110	011	011	110	110	1
110	010	011	011	011	010	011	011	011	1
111	010	010	110	110	010	010	110	110	1

Figur S7o Tillståndstabell med realiserade värden för don´t care-tillstånd

Det är möjligt att en annan tilldelning av värden till tillstånden skulle ge en enklare implementering. Vi går inte in på detta

---

8 Skriv VHDL-kod för både den ominimerade och den minimerade tillståndstabellen i *uppgift 7*

Vi börjar med den ominimerade tillståndsmaskinen från *figur S7a*

```
-- S8_omin.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

8 forts.

```
ENTITY S8_omin IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        D:IN STD_LOGIC;
        N:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S8_omin;

ARCHITECTURE arch_ S8_omin OF S8_omin IS
   TYPE state_type IS (S1,S2,S3,S4,S5,S6,S7,S8,S9);
   SIGNAL state_signal:state_type;
   SIGNAL next_state_signal:state_type;
   SIGNAL DN_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
   DN_vector_signal <= D & N;
   state_transition_proc:
      PROCESS(Resetn,Clock)
      BEGIN
         IF (Resetn='0') THEN
            state_signal<=S1;
         ELSIF rising_edge(Clock) THEN
            state_signal<=next_state_signal;
         END IF;
   END PROCESS state_transition_proc;

   stateflow_proc:
   PROCESS(state_signal,DN_vector_signal)
   BEGIN
      CASE state_signal IS
         WHEN S1 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S1;
            ELSIF (DN_vector_signal = "01") THEN
               next_state_signal <= S3;
            ELSIF (DN_vector_signal = "10") THEN
               next_state_signal <= S2;
            END IF;
         WHEN S2 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S2;
            ELSIF (DN_vector_signal = "01") THEN
               next_state_signal <= S4;
            ELSIF (DN_vector_signal = "10") THEN
               next_state_signal <= S5;
            END IF;
```

8 forts.

```
            WHEN S3 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S3;
               ELSIF (DN_vector_signal = "01") THEN
                  next_state_signal <= S6;
               ELSIF (DN_vector_signal = "10") THEN
                  next_state_signal <= S7;
                END IF;
            WHEN S4 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S1;
               END IF;
            WHEN S5 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S3;
               END IF;
            WHEN S6 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S6;
               ELSIF (DN_vector_signal = "01") THEN
                  next_state_signal <= S8;
               ELSIF (DN_vector_signal = "10") THEN
                  next_state_signal <= S9;
               END IF;
            WHEN S7 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S1;
               END IF;
            WHEN S8 =>
               IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S1;
               END IF;
            WHEN S9 =>
                IF (DN_vector_signal = "00") THEN
                  next_state_signal <= S3;
               END IF;
         END CASE;
      END PROCESS stateflow_proc;

      assignment_proc:
      PROCESS(state_signal,DN_vector_signal)
      BEGIN
         IF ((state_signal=S4) OR
             (state_signal=S5) OR
             (state_signal=S7) OR
             (state_signal=S8) OR
             (state_signal=S9)) THEN
            z <= '1';
```

8 forts.

```
      ELSE
          z <= '0';
       END IF;
    END PROCESS assignment_proc;
END arch_S8_omin;
```

Vi simulerar med en do-fil

```
-- S8_omin.do
restart -f -nowave
view signals wave
add wave Clock Resetn D N
add wave -radix binary DN_vector_signal
add wave state_signal next_state_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force D 0
force N 0
force Resetn 0
run 225ns
force Resetn 1
force D 0
run 400ns
force D 1
run 400ns
force D 0
run 400ns
force N 1
run 400ns
force N 0
run 400ns
force D 1
run 400ns
force D 0
run 400ns
force N 1
run 400ns
```

Vi övergår till den minimerade tillståndsmaskinen

```
-- S8_min.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

8 forts.

```
ENTITY S8_min IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        D:IN STD_LOGIC;
        N:IN STD_LOGIC;
        z:OUT STD_LOGIC);
END S8_min;

ARCHITECTURE arch_S8_min OF S8_min IS
   TYPE state_type IS (S1,S2,S3,S4,S5);
   SIGNAL state_signal:state_type;
   SIGNAL next_state_signal:state_type;
   SIGNAL DN_vector_signal:STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
   DN_vector_signal <= D & N;
   state_transition_proc:
      PROCESS(Resetn,Clock)
      BEGIN
         IF (Resetn='0') THEN
            state_signal<=S1;
         ELSIF rising_edge(Clock) THEN
            state_signal<=next_state_signal;
         END IF;
   END PROCESS state_transition_proc;

   stateflow_proc:
   PROCESS(state_signal,DN_vector_signal)
   BEGIN
      CASE state_signal IS
         WHEN S1 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S1;
            ELSIF (DN_vector_signal = "01") THEN
               next_state_signal <= S3;
            ELSIF (DN_vector_signal = "10") THEN
               next_state_signal <= S2;
            ELSE
               next_state_signal <= S1;
            END IF;
         WHEN S2 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S2;
            ELSIF (DN_vector_signal = "01") THEN
               next_state_signal <= S4;
            ELSIF (DN_vector_signal = "10") THEN
               next_state_signal <= S5;
```

8 forts.

```
            ELSE
               next_state_signal <= S1;
            END IF;
         WHEN S3 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S3;
            ELSIF (DN_vector_signal = "01") THEN
               next_state_signal <= S2;

            ELSIF (DN_vector_signal = "10") THEN
               next_state_signal <= S4;
            ELSE
               next_state_signal <= S1;
            END IF;
         WHEN S4 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S1;
            ELSE
               next_state_signal <= S1;
            END IF;
         WHEN S5 =>
            IF (DN_vector_signal = "00") THEN
               next_state_signal <= S3;
            ELSE
               next_state_signal <= S1;
            END IF;
      END CASE;
   END PROCESS stateflow_proc;

   assignment_proc:
   PROCESS(state_signal,DN_vector_signal)
   BEGIN
      IF ((state_signal=S4) OR
          (state_signal=S5)) THEN
         z <= '1';
      ELSE
         z <= '0';
      END IF;
   END PROCESS assignment_proc;
END arch_S8_min;
```

Vi simulerar åter med en do-fil

8 forts.

```
-- S8_min.do
restart -f -nowave
view signals wave
add wave Clock Resetn D N
add wave -radix binary DN_vector_signal
add wave state_signal next_state_signal z
force Clock 0 0, 1 50ns -repeat 100ns
force D 0
force N 0
force Resetn 0
run 225ns
force Resetn 1
force D 0
run 400ns
force D 1
run 400ns
force D 0
run 400ns
force N 1
run 400ns
force N 0
run 400ns
force D 1
run 400ns
force D 0
run 400ns
force N 1
run 400ns
```

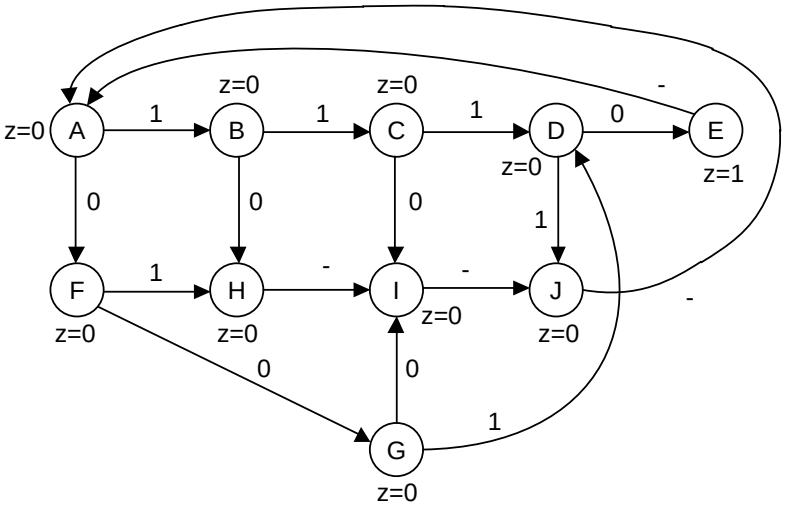
---

9 Ta fram tillståndstabellen för en tillståndsmaskin som skall detektera en sekvens av fyra värden på ingången *w*. Utsignalen *z* ska bli ett (1) om sekvensen är 0010 eller 1110. I övriga fall ska utsignalen bli noll (0). Efter en sekvens om fyra värden ska tillståndsmaskenen börja om med en ny sekvens, dvs sekvenserna kan inte vara överlappande. Minimera tillståndstabellen

---

9 forts.

Vi översätter funktionsbeskrivningen, att vi skall få en etta (1) som utsignal om vi har haft pulssekvensen 0010 eller pulssekvensen 1110 och i annat fall en nolla (0), till tillståndsgrafen för en tillståndsmaskin av Mooretyp, *Figur S9a*. Efter fyra klockpulser skall vi alltid återgå till starttillståndet



Figur S9a Tillståndsgraf

Från tillståndsgrafen tecknar vi tillståndstabellen, *Figur S9b*, som vi även skriver som ett partionsuttryck

$$P_1 = (A, B, C, D, E, F, G, H, I, J)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0
G	I	D	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9b Tillståndstabell

9 forts.

Vi börjar tillståndsminimeringen med att dela in tabellen i tillstånd som skiljer sig genom att de ger olika utsignal. Vi ser att här är det tillstånd *E* som skiljer sig från de övriga och vi får ett nytt partionsuttryck samt en ny tabell, *Figur S9c*

$$P_2 = (A, B, C, D, F, G, H, I, J)(E)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
D	E	J	0
F	G	H	0
G	I	D	0
H	I	I	0
I	J	J	0
J	A	A	0
E	A	A	1

Figur S9c minimeringstabell 1

Vi fortsätter med att titta på vilka tillstånd vi går till från nuvarande tillstånd då *w=0* respektive då *w=1*. För att vi skall kunna slå ihop tillstånden så måste efterföljande tillstånd vid respektive insignal tillhöra samma partion. Vi ser att tillstånd *D* skiljer sig från de andra tillstånden i den första gruppen genom att ge en övergång till den andra gruppen. Vi får ett nytt partionsuttryck och en ny tabell, *Figur S9d*

$$P_3 = (A, B, C, D, F, G, H, I,)(D)(E)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
F	G	H	0
G	I	D	0
H	I	I	0
I	J	J	0
J	A	A	0
D	E	J	0
E	A	A	1

Figur S9d minimeringstabell 2

9 forts.

Vi har nu fått nya partioner som vi kan granska på samma sätt. Vi ser att tillstånden *C* och *G* går till en annan partion än de andra tillstånden i den första gruppen. De går dock till samma grupp vilket innebär att de har en egen partion. Vi får ett nytt partionsuttryck och en ny tabell, *Figur S9e*

$$P_4 = (A, B, F, H, 1, J)(C, G)(D)(E)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
F	G	H	0
H	I	I	0
I	J	J	0
J	A	A	0
C	I	D	0
G	I	D	0
D	E	J	0
E	A	A	1

Figur S9e minimeringstabell 3

Vi fortsätter på samma sätt och ser att i den första partionen går tillstånd *B* och *F* till andra partioner, de går dock inte till samma partioner så de får var och en läggas i en egen ny partion. Vi får ett nytt partionsuttryck och en ny tabell, *Figur S9f*

$$P_5 = (A, H, I, J)(B)(C, G)(D)(E)(F)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
H	I	I	0
I	J	J	0
J	A	A	0
B	H	C	0
C	I	D	0
G	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0

Figur S9f minimeringstabell 4

9 forts.

Vi fortsätter och ser att tillstånd *A* går till en annan partion än vad övriga tillstånd i den första partionen går. *A* får alltså en egen partion och vi får ett nytt partionsuttryck och en ny tabell, *Figur S9g*

$$P_5 = (A)(B)(C, G)(D)(E)(F)(H, I, J)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
G	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9g minimeringstabell 5

Vi ser nu att i den sista partionen går tillstånd *J* till en annan partion än övriga tillstånd och den får en egen partion. Vi får ett nytt partionsuttryck och en ny tabell, *Figur S9h*

$$P_6 = (A)(B)(C, G)(D)(E)(F)(H, I)(J)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
G	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9 minimeringstabell 6

9 forts.

Vi kan nu se att tillstånd *H* och *I* går till olika partioner så de får delas upp i två partioner. Vi skriver ett nytt partionsuttryck och tecknar en ny tabell, *Figur S9i*

$$P_7 = (A)(B)(C, G)(D)(E)(F)(H)(I)(J)$$

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
G	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9i minimeringstabell 7

Vi kommer nu inte längre och kan konstatera att vi kan slå samman tillstånd *C* med tillstånd *G*. Behåller vi tillstånd *C* så får vi tillståndstabellen i *Figur S9j*

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
D	E	J	0
E	A	A	1
F	C	H	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9j Minimerad tillståndstabell

9 forts.

Vi skall nu se på en annan metod för tillståndsminimering som är lite enklare att hantera med penna och papper. Vi upprepar den ursprungliga tillståndstabellen för att ha den tillgänglig, *Figur S9k*.

Pre-sent state	*Next state*		*Out-put*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
D	E	J	0
E	A	A	1
F	G	H	0
G	I	D	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9k Tillståndstabell

Vi ställer upp en tabell över alla möjliga tillståndsekvivalenter, *Figur S9l*

B								
C								
D								
E								
F								
G								
H								
I								
J								
	A	B	C	D	E	F	G	H	I

Figur S9l Tabell 1 för tillståndsminimering med alla möjliga tillståndsekvivalenter

I tabellen dimmar vi alla tillståndsekvivalenter som är omöjliga på grund av att tillstånden har olika utsignaler, vi ser enkelt att tillstånd *E* har avvikande utsignal från de andra tillstånden, *Figur S9m*

9 forts.

B								
C								
D								
E								
F								
G								
H								
I								
J								
	A	B	C	D	E	F	G	H	I

Figur S9m Tabell 2 för tillståndsminimering

För att två tillstånd skall vara ekvivalenta så måste även de efterföljande tillstånden, dvs de tillstånd till vilka vi går från aktuellt tillstånd vara ekvivalenta. Vi skriver i de odimmade rutorna in de par av efterföljande tillstånd som måste vara ekvivalenta för att de två aktuella tillstånden skall vara ekvivalenta. Ingår de två aktuella tillstånden i paret av efterföljande tillstånd så behöver vi inte skriva in dem eftersom det är dessa som vi vill slå ihop. Vi behöver inte heller skriva in ett par som redan är en ekvivalens, dvs är ett och samma tillstånd, *Figur S9n*

B	B,C F,H							
C	B,D F,I	C,D H,I						
D	B,J E,F	C,J E,H	D,J E,I					
E								
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I	A,J
	A	B	C	D	E	F	G	H	I

Figur S9n Tabell 3 för tillståndsminimering

Vi fortsätter genom att dimma ut ekvivalenter där vi har skrivit in tillståndspar som vi redan har dömt som inte ekvivalenta genom att vi har dimmat ut dem, *Figur S9o*

9 forts.

B	B,C F,H								
C	B,D F,I	C,D H,I							
D	B,J E,F	C,J E,H	D,J E,I						
E									
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J					
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I			
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I		
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J	
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I	A,J
	A	B	C	D	E	F	G	H	I

Figur s9o Tabell 4 för tillståndsminimering

Detta har nu givit upphov till nya dimningar, dvs tillstånd som inte kan vara ekvivalenter och vi kan upprepa föregående uteslutning av ekvivalenter med hjälp av de nya dimningarna, *Figur S9p*

B	B,C F,H								
C	B,D F,I		C,D H,I						
D	B,J E,F		C,J E,H	D,J E,I					
E									
F	B,H F,G		C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I		C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I		C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J		C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F		A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I
	A		B	C	D	E	F	G	H

Figur E8.11p Tabell 5 för tillståndsminimering

9 forts.

Detta ger upphov till ytterligare tillstånd som inte kan vara ekvivalenter och vi kör ett varv till, *Figur S9q*

B	B,C F,H							
C	B,D F,I	C,D H,I						
D	B,J E,F	C,J E,H	D,J E,I					
E								
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I	A,J
	A	B	C	D	E	F	G	H	I

Figur S9q Tabell 6 för tillståndsminimering

Detta ger upphov till ytterligare nya icke-ekvivalenta tillstånd, *Figur S9r*

B	B,C F,H							
C	B,D F,I	C,D H,I						
D	B,J E,F	C,J E,H	D,J E,I					
E								
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I	A,J
	A	B	C	D	E	F	G	H	I

Figur S9r Tabell 7 för tillståndsminimering

9 forts.

Vi kör ett varv till, *Figur S9s*

B	B,C F,H							
C	B,D F,I	C,D H,I						
D	B,J E,F	C,J E,H	D,J E,I					
E								
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I
	A	B	C	D	E	F	G	H

Figur S9s Tabell 8 för tillståndsminimering

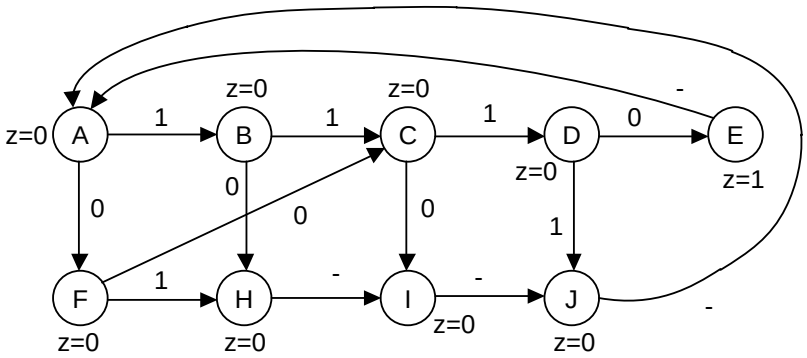
och ett varv till, *Figur S9t*

B	B,C F,H							
C	B,D F,I	C,D H,I						
D	B,J E,F	C,J E,H	D,J E,I					
E								
F	B,H F,G	C,H G,H	D,H G,I	E,G H,J				
G	B,D F,I	C,D H,I		D,J E,I		D,H G,I		
H	B,I F,I	C,I H,I	D,I	E,I I,J		G,I H,I	D,I	
I	B,J F,J	C,J H,J	D,J I,J	E,J		G,J H,J	D,J I,J	I,J
J	A,B A,F	A,C A,H	A,D A,I	A,E A,J		A,G A,H	A,D A,I	A,I
	A	B	C	D	E	F	G	H

Figur S9t Tabell 9 för tillståndsminimering

9 forts.

eftersom det enda kvarstående möjliga ekvivalenspar saknar efterföljande par som måste vara ekvivalenter så kommer vi inte längre och kan dra slutsats att vi kan slå ihop tillstånd *C* och tillstånd *G* och vi får den minimerade tillståndstabellen, *Figur S9u*, som är likadan som *Figur S9j* som vi fick från den första metoden.



Figur S9v Förenklad tillståndsgraf

Låt oss rita upp den förenklade tillståndsgrafen, *Figur Sv*

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	F	B	0
B	H	C	0
C	I	D	0
D	E	J	0
E	A	A	1
F	C	H	0
H	I	I	0
I	J	J	0
J	A	A	0

Figur S9u Minimerad tillståndstabell

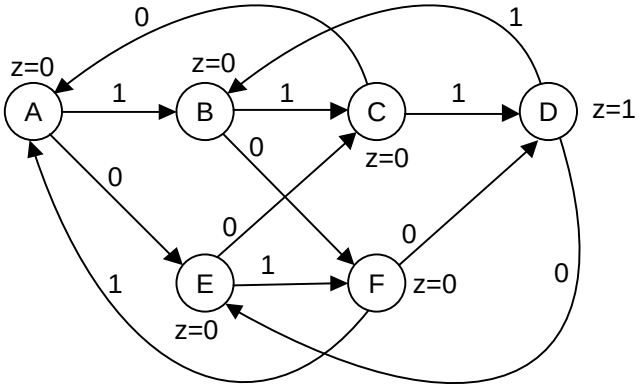
---

10 Ta fram och minimera tillståndstabellen för en tillståndsmaskin som detekterar om antalet ettor (1) i en trebitars sekvens på ingången w är udda eller jämnt. Udda antal ettor skall sätta utsignalen z till ett (1). Trebitarssekvenserna är kan inte vara överlappande

---

10 forts.

Vi får tillståndsdiagrammet i *Figur S10a*



Figur S10a Tillståndsdiagram

Från tillståndsgrafen tecknar vi tillståndstabellen, *Figur S10b*, och börjar tillståndsmimimeringen med att ställa upp en tabell över alla möjliga tillståndsekvivalenter, *Figur E8.12c*

Present state	*Next state*		*Output*
	w=0	*w=1*	*z*
A	E	B	0
B	F	C	0
C	A	D	0
D	E	B	1
E	C	F	0
F	D	A	0

Figur S10b Tillståndstabell

B					
C					
D					
E					
F					
	A	B	C	D	E

Figur S10c Tabell för tillståndsminimering med alla möjliga tillståndsekvivalenter

10 forts.

I tabellen dimmar vi alla ekvivalenter som är omöjliga på grund av att tillstånden har olika utsignaler, vi ser enkelt att tillstånd *D* har enutsignal som avviker från övriga tillstånd, *Figur S10d*

B					
C					
D					
E					
F					
	A	B	C	D	E

Figur S10d Tabell 2 för tillståndsminimering

För att två tillstånd skall vara ekvivalenta så måste även de efterföljande tillstånden, dvs de tillstånd till vilka vi går från aktuellt tillstånd vara ekvivalenta. Vi skriver i de odimmade rutorna in de par av efterföljande tillstånd som måste vara ekvivalenta för att de två aktuella tillstånden skall vara ekvivalenta. Ingår de två aktuella tillstånden i paret av efterföljande tillstånd så behöver vi inte skriva in dem eftersom det är dessa som vi vill slå ihop. Vi behöver inte heller skriva in ett par som redan är en ekvivalens, dvs är ett och samma tillstånd, *Figur S10e*.

B	B,C E,F				
C	A,E B,D	A,F C,D			
D					
E	B,F C,E	C,F	A,C D,F		
F	A,B D,E	A,C D,F	A,D		A,F C,D
	A	B	C	D	E

Figur S10e Tabell 3 för tillståndsminimering

Vi fortsätter genom att dimma ut ekvivalenter där vi har tillståndspar som vi redan har dömt som inte ekvivalenta genom att vi har dimmat ut dem, *Figur S10f*

B	B,C E,F				
C	A,E B,D	A,F C,D			
D					
E	B,F C,E	C,F	A,C D,F		
F	A,B D,E	A,C D,F	A,D		A,F C,D
	A	B	C	D	E

Figur S10f Tabell 4 för tillståndsminimering

10 forts.

Detta har nu givit upphov till nya dimningar, dvs tillstånd som inte är ekvivalenter och vi kan upprepa föregående uteslutning av ekvivalenter med hjälp av de nya dimningarna, *Figur S10g*.

Vi ser att alla möjliga ekvivalenter är bortdimmade och vi kan dra slutsatsen att vi hade en minimal tillståndstabell från början

B	B,C E,F				
C	A,E B,D	A,F C,D			
D					
E	B,F C,E	C,F	A,C D,F		
F	A,B D,E	A,C D,F	A,D		A,F C,D
	A	B	C	D	E

Figur E8.12g Tabell 5 för tillståndsminimering

---

11 Minimera tillståndstabellen i *figur E11*

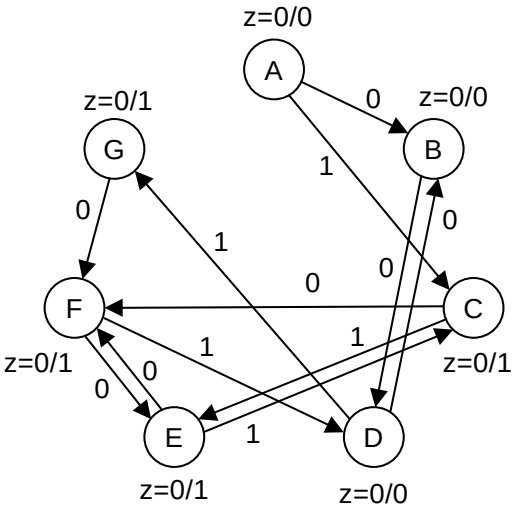
Present state	*Next state*		*Output*	
	$w_1=0$	$w_1=1$	$w_1=0$	$w_1=1$
A	B	C	0	0
B	D	-	0	0
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	-	0	1

Figur E11 Modifierad tillståndstabell

---

11 forts.

Låt oss först rita tillståndsdiagram, *Figur S11a*



Figur S11a Tillståndsgraf

Vi börjar vår tillståndsminimering med att ställa upp en tabell över alla möjliga tillståndsekvivalenter, *Figur S11b*

B						
C						
D						
E						
F						
G						
	A	B	C	D	E	F

Figur S11b Tabell för tillståndsminimering

I tabellen dimmar vi alla ekvivalenter som är omöjliga på grund av att tillstånden har olika utsignal, *Figur S11c*

B						
C						
D						
E						
F						
G						
	A	B	C	D	E	F

Figur S11c Tabell 2 för tillståndsminimering

För att två tillstånd skall vara ekvivalenta så måste även de efterföljande tillstånden, dvs de tillstånd till vilka vi går från aktuellt tillstånd vara ekvivalenta. Vi skriver i de odimmade rutorna in de par av efterföljande tillstånd som måste vara ekvivalenta för att de två aktuella tillstånden skall vara ekvivalenta.

11 forts.

Ingår de två aktuella tillstånden i paret av efterföljande tillstånd så behöver vi inte skriva in dem eftersom det är dessa som vi vill slå ihop. Vi behöver inte heller skriva in ett par som redan är en ekvivalens, dvs är ett och samma tillstånd, *Figur S11d*.

Vi fortsätter genom att dimma ut par som vi redan har dömt som icke ekvivalenta genom att vi har dimmat ut dem, *Figur S11e*.

B	B,D					
C						
D	C,G					
E						
F			E,F D,E		C,D	
G						E,F
	A	B	C	D	E	F

Figur S11d Tabell 3 för tillståndsminimering

B	B,D					
C						
D	C,G					
E						
F			E,F D,E		C,D	
G						E,F
	A	B	C	D	E	F

Figur S11e Tabell 4 för tillståndsminimering

Vi fortsätter på samma sätt, *Figur S11f*.

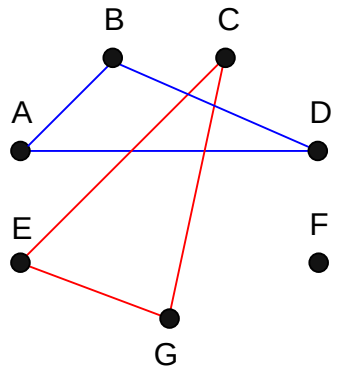
B	B,D					
C						
D	C,G					
E						
F			E,F D,E		C,D	
G						E,F
	A	B	C	D	E	F

Figur S11f Tabell 5 för tillståndsminimering

11 forts.

Vi kommer inte längre i minimeringen och ritar relationsgraf, *Figur S11g*.

Vi ser att vi kan slå ihop tillstånd *A* och B eller tillstånd *B* och *D* samt tillstånd *C*, *E* och *G*. Då vi har slagit ihop tillstånd *C* och *G* så ser vi att även *A* och *D* kan slås ihop och vi slår ihop tillstånd *A*, *B* och *D* till ett tillstånd.
Låt oss behålla tillstånd *A* och *C*. Vi får den minimerade tillståndstabellen, *Figur S11h*.

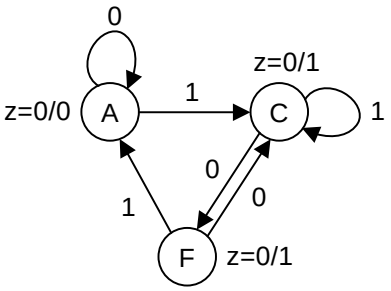


Figur S11g Relationsgraf

Present state	*Next state*		*Output*	
	w_1=0	w_1=1	w_1=0	w_1=1
A	A	C	0	0
C	F	C	0	1
F	C	A	0	1

Figur S11h Minimerad tillståndstabell

Vi ritar ny tillståndsgraf, *Figur S11i*.



Figur S11i Minimerad tillståndsgraf

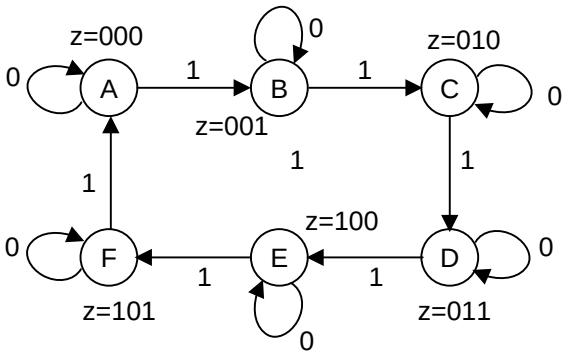
---

12 Använd D-vippor för att konstruera en modulo-6-räknare med sekvensen 0, 1, 2, 3, 4, 5, 0, 1,…. Räknaren räknar klockpulser men bara då ingången *w* är ett (1).

---

12 forts.

Vi har tillståndsgrafen i *Figur S12a*.



Figur S12a Tillståndsgraf

För detta behöver vi tre (3) utsignaler (2^3=8) och även tre (3) vippor. Vi ställer upp tillståndstabellen, *Figur S12b*

Present state	*Next state*		*Output*		
	w=0	*w=1*	z_2	z_1	z_0
A	A	B	0	0	0
B	B	C	0	0	1
C	C	D	0	1	0
D	D	E	0	1	1
E	E	F	1	0	0
F	F	A	1	0	1

Figur S12b Tillståndstabell

Present state	*D-flipflop*	*Next state*
	D	
0	0	0
0	1	1
1	0	0
1	1	1

Figur S12c Tillståndstabell för JK-vippa

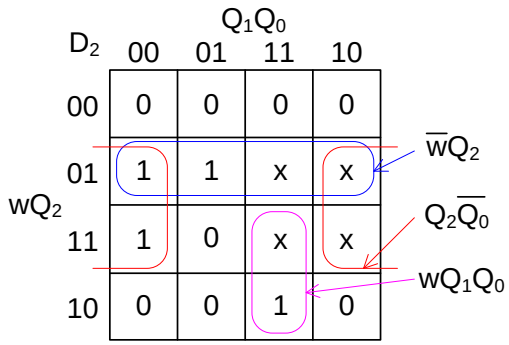
Då vi skall använda D-vippor ställer vi upp tillståndstabellen för dessa, *Figur S12c*, och använder denna tabell för att införa D-vippor i vår tillståndstabell, *S12d*

Present state			*D-flipflops*						*Next state*						*Output*		
			w=0			*w=1*			*w=0*			*w=1*					
Q_2	Q_1	Q_0	D_2	D_1	D_0	D_2	D_1	D_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	z_2	z_1	z_0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	0	1	0	0	1	1	0	1	0	0	1	1	0	1	0
0	1	1	0	1	1	1	0	0	0	1	1	1	0	0	0	1	1
1	0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	0
1	0	1	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1

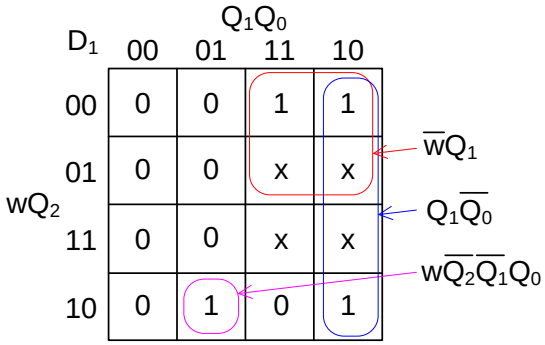
Figur S12d Tillståndstabell med D-vippor

12 forts.

I tabellen behöver vi egentligen inte ha med D-ingångarna eftersom dom har samma värden som Q-utgångarna i Next state.
Vi använder Karnaughdiagram för att bestämma villkoren för våra tre insignaler, *Figur S12e-g*



Figur S12e Karnaughdiagram för D_2

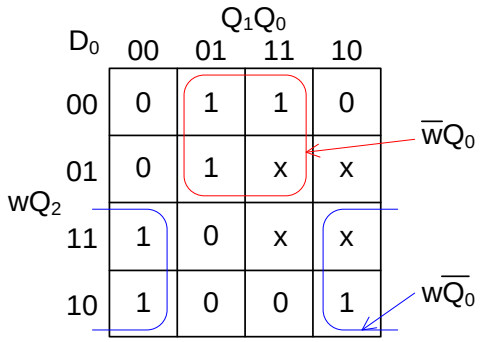


Figur S12f Karnaughdiagram för D_1

$$D_2 = Q_2 \cdot \overline{Q_0} + \overline{w} \cdot Q_2 + w \cdot Q_1 \cdot Q_0$$

$$D_1 = Q_1 \cdot \overline{Q_0} + \overline{w} \cdot Q_1 + w \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot Q_0$$

$$D_0 = \overline{w} \cdot Q_0 + w \cdot \overline{Q_0} = w \oplus Q_0$$



Figur S12g Karnaughdiagram för D_0

Utsignalerna är de samma som vippornas utsignaler.
Det är möjligt att någon annan tilldelning av värden till tillstånden skulle ge en enklare implementering men vi går inte in på detta

---

13 Upprepa *uppgift 12* med JK-vippor

Vi har tillståndstabellen för en JK-vippa i *Figur S13a* och skriver om tillståndstabellen i *Figur S12a* men för att få plats så utelämnar vi kolumnerna med utsignaler då dessa har samma värden som vippornas utsignaler. Vi får *Figur S13b*

Present state	*JK-flipflop*		*Next state*
	J	*K*	
0	0	x	0
0	1	x	1
1	x	1	0
1	x	0	1

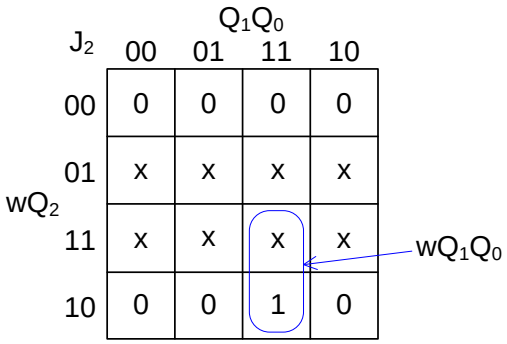
Figur S13a Tillståndstabell för JK-vippa

13 forts.

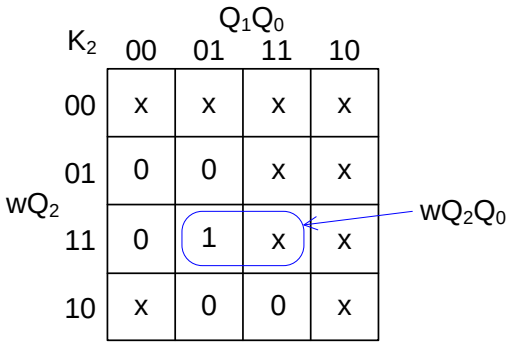
Present state			*JK-flipflops*												*Next state*					
			w=0						*w=1*						*w=0*			*w=1*		
Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0	J_2	K_2	J_1	K_1	J_0	K_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0
0	0	0	0	x	0	x	0	x	0	x	0	x	1	x	0	0	0	0	0	1
0	0	1	0	x	0	x	x	0	0	x	1	x	x	1	0	0	1	0	1	0
0	1	0	0	x	x	0	0	x	0	x	x	0	1	x	0	1	0	0	1	1
0	1	1	0	x	x	0	x	0	1	x	x	1	x	1	0	1	1	1	0	0
1	0	0	x	0	0	x	0	x	x	0	0	x	1	x	1	0	0	1	0	1
1	0	1	x	0	0	x	x	0	x	1	0	x	x	1	1	0	1	0	0	0

Figur S13b Tillståndstabell med D-vippor

Vi ställer upp Karnaughdiagram för våra insignaler J och K till de tre vipporna, *Figur S13c-h*



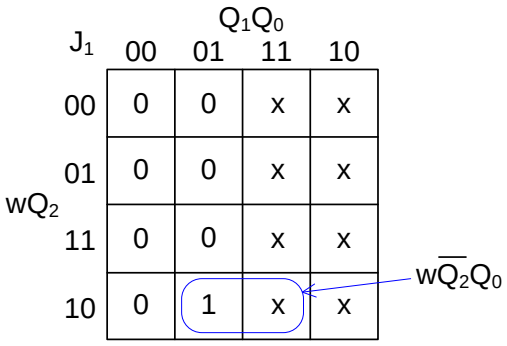
Figur S13c Karnaughdiagram för J_2



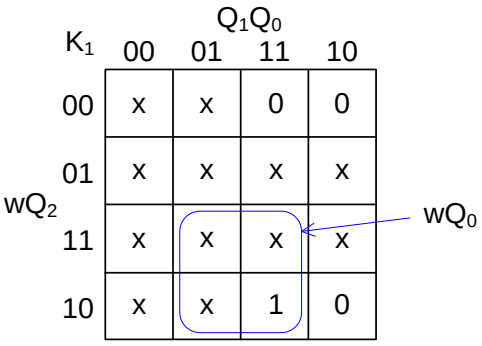
Figur S13d Karnaughdiagram för K_2

$$J_2 = w \cdot Q_1 \cdot Q_0$$

$$K_2 = w \cdot Q_2 \cdot Q_0$$



Figur S13e Karnaughdiagram för J_1

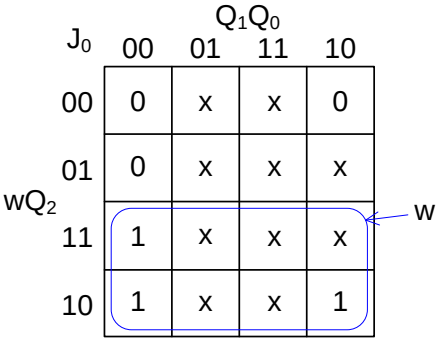


Figur S13f Karnaughdiagram för K_1

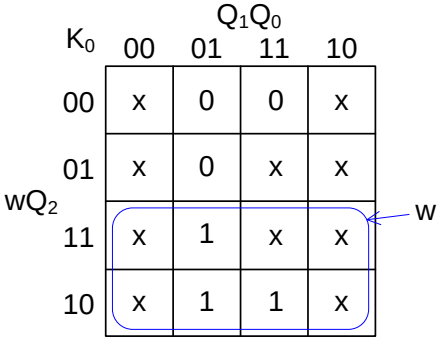
13 forts.

$$J_1 = w \cdot \overline{Q_2} \cdot Q_0$$

$$K_1 = w \cdot Q_0$$



Figur S13 Karnaughdiagram för J_0



Figur S13 Karnaughdiagram för K_0

$$J_0 = K_0 = w$$

Och utsignalerna `z2`, `z1` och `z0` är alltså de samma som JK-vippornas utsignaler `Q2`, `Q1` respektive Q0.
Det är möjligt att någon annan tilldelning av värden till tillstånden skulle ge en enklare implementering men vi går inte in på detta

---

14 Upprepa *uppgift 12* med T-vippor

Vi har tillståndstabellen för en T-vippa i *Figur S14a* och skriver om tillståndstabellen i *Figur S12a* med T-vippor och får *Figur S14b*

Present state	*T-flipflop*	*Next state*
	T	
0	0	0
0	1	1
1	1	0
1	0	1

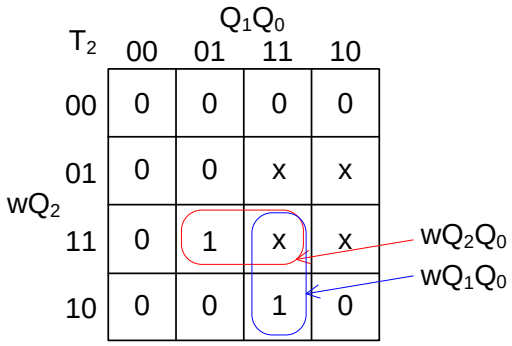
Figur S14a Tillståndstabell för JK-vippa

14 forts.

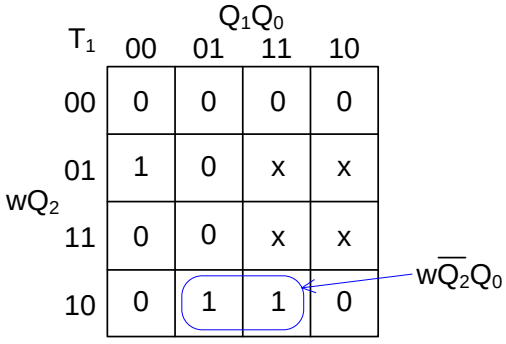
Present state			*T-flipflops*						*Next state*						*Output*		
			w=0			*w=1*			*w=0*			*w=1*					
Q_2	Q_1	Q_0	T_2	T_1	T_0	T_2	T_1	T_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	z_2	z_1	z_0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	1	0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	0	1	0	1	0	0	1	1	0	1	0
0	1	1	0	0	0	1	1	1	0	1	1	1	0	0	0	1	1
1	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	0
1	0	1	0	0	0	1	0	1	1	0	1	0	0	0	1	0	1

Figur S14b Tillståndstabell med D-vippor

Vi använder Karnaughdiagram för att ta fram de logiska uttrycken för T-signalerna, *Figur S15c-e*. Då utsignalerna har samma värden som tillståndsvärdena så behöver vi inte bestämma dessa för sig.



Figur S14c Karnaughdiagram för T_2

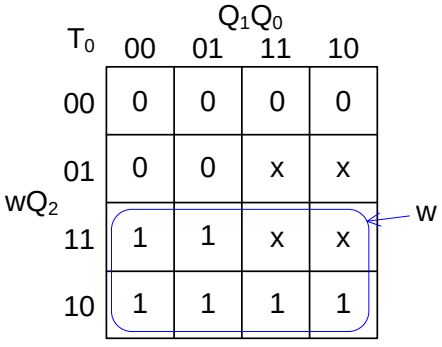


Figur S14d Karnaughdiagram för T_1

$$T_2 = w \cdot Q_1 \cdot Q_0 + w \cdot Q_2 \cdot Q_0$$

$$T_1 = w \cdot \overline{Q_2} \cdot Q_0$$

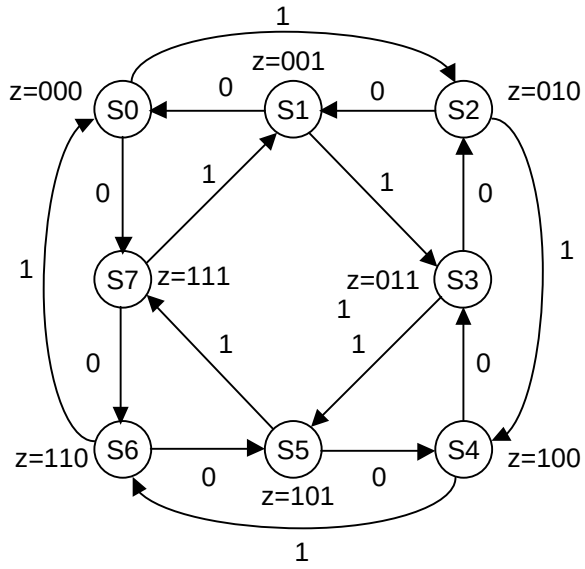
$$T_0 = w$$



Figur S14e Karnaughdiagram för T_0

15 Ta fram tillståndsgraf och tillståndstabell för en räknarliknande krets med åtta (8) tillstånd 0-7. Är ingången w ett (1) så skall kretsen räkna uppåt med steget två (2). Då den har mått maxvärde så skall den göra wrap around, dvs är värdet åtta (8) så skall nästa värde bli noll (0) medan nästa värde skall bli ett (1) om nuvarande värde är nio (9). Är w noll (0) så skall kretsen fungera som en vanlig nedräknare med steget ett som gör wrap around då den når noll (0).
Implementera kretsen med hjälp av D-vippor

Vi ritar tillståndsgraf, *Figur S15a* och tecknar tillståndstabell, *Figur S15b*, där vi har givit tillstånden bokstavsbeteckningar



Figur S15a Tillståndsgraf

Present state	*Next state*		*Count*
	w=0	*w=1*	
S0	S7	S2	0
S1	S0	S3	1
S2	S1	S4	2
S3	S2	S5	3
S4	S3	S6	4
S5	S4	S7	5
S6	S5	S0	6
S7	S6	S1	7

Figur S15b Tillståndstabell

Vi har tillståndstabellen för en D-vippa i *Figur S16c*.
Värden 0 – 7 innebär at vi behöver tre (3) D-vippor (2^3=8). Vi skriver om tillståndstabellen med tilldelade värden på tillstånden och med D-ingångarnas värden, *Figur S16d*

Present state	*D-flipflop*	*Next state*
	D	
0	0	0
0	1	1
1	0	0
1	1	1

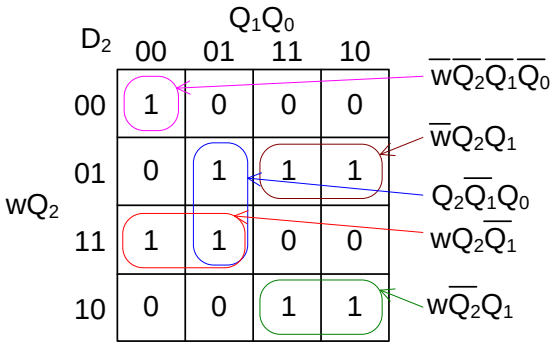
Figur S15c Tillståndstabell för JK-vippa

15 forts.

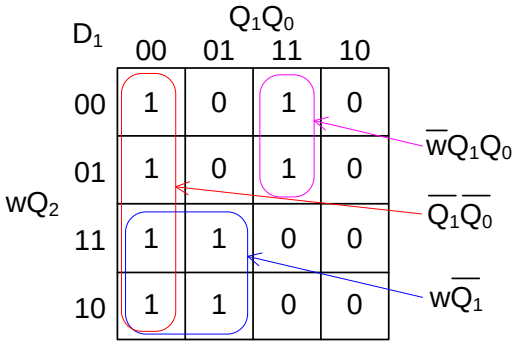
Present state			*D-flipflops*						*Next state*						*Output*		
			w=0			*w=1*			*w=0*			*w=1*					
Q_2	Q_1	Q_0	D_2	D_1	D_0	D_2	D_1	D_0	Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	z_2	z_1	z_0
0	0	0	1	1	1	0	1	0	1	1	1	0	1	0	0	0	0
0	0	1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	1
0	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0
0	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	1	1
1	0	0	0	1	1	1	1	0	0	1	1	1	1	0	1	0	0
1	0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	0	1
1	1	0	1	0	1	0	0	0	1	0	1	0	0	0	1	1	0
1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1

Figur S15d Tillståndstabell med D-vippor

Vi använder Karnaughdiagram för att söka de logiska villkoren för D-vipporna, *Figur S15e-g*. Eftersom räknarutgångarna har samma värden som våra vippors utgångar så behöver vi inga egna uttryck för dessa



Figur S15e Karnaughdiagram för D_2

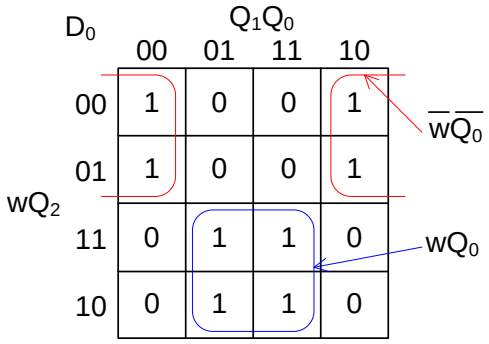


Figur S15f Karnaughdiagram för D_1

$$D_2 = \overline{w}\cdot Q_2\cdot Q_1 + Q_2\cdot\overline{Q_1}\cdot Q_0 + w\cdot Q_2\cdot\overline{Q_1} + w\cdot\overline{Q_2}\cdot Q_1 + \overline{w}\cdot\overline{Q_2}\cdot\overline{Q_1}\cdot\overline{Q_0}$$

$$D_1 = w\cdot\overline{Q_1} + \overline{Q_1}\cdot\overline{Q_0} + \overline{w}\cdot Q_1\cdot Q_0$$

$$D_0 = w\cdot Q_0 + \overline{w}\cdot\overline{Q_0} = \overline{w\oplus Q_0}$$



Figur S15g Karnaughdiagram för D_0

16 Skriv VHDL-kod för tillståndstabellen framtagen i *uppgift 15*

Vi får koden

```
-- S16.vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;
```

16 forts.

```
ENTITY S16 IS
   PORT(Clock:IN STD_LOGIC;
        Resetn:IN STD_LOGIC;
        w:IN STD_LOGIC;
        count:OUT STD_LOGIC_VECTOR(2 DOWNTO 0));
END S16;

ARCHITECTURE arch_S16 OF S16 IS

   SIGNAL count_signal:INTEGER RANGE 0 TO 7;
BEGIN
   count_proc:
   PROCESS(Resetn,Clock)
   BEGIN
      IF (Resetn='0') THEN
         count_signal<=0;
      ELSIF rising_edge(Clock) THEN
         IF (w='1') THEN
            IF (count_signal = 6) THEN
               count_signal <= 0;
            ELSIF (count_signal = 7) THEN
               count_signal <= 1;
            ELSE
               count_signal <= count_signal + 2;
            END IF;
         ELSE
            IF (count_signal = 0) THEN
               count_signal <= 7;
            ELSE
               count_signal <= count_signal -1;
            END IF;
         END IF;
      END IF;
   END PROCESS count_proc;
```

```
   count <= STD_LOGIC_VECTOR(TO_UNSIGNED(count_signal,3));

END arch_S16;
```

Vi simulerar med en do-fil

```
-- S16.do
restart -f -nowave
view signals wave
add wave Clock Resetn w
add wave count_signal count
add wave -radix binary count
add wave -unsigned count
force Clock 0 0, 1 50ns -repeat 100ns
force Resetn 0
force w 0
run 225ns
force Resetn 1
run 1500ns
force w 1
run 1500ns
force w 0
run 500ns
force w 1
run 500ns
```