EDA321 Digitalteknik-syntes D2 och E3

GU **DIT795**

Tentamen *(EDA321-0205)*

Tisdag den 15 mars 2011, fm i M-salarna

Examinator
Arne Linde, tel. 772 1683

Tillåtna hjälpmedel

Inga hjälpmedel tillåtna. Detta innefattar även kalkylatorer och alla tabellverk.

Oläsliga eller svårtydda lösningar ger poängavdrag.

Allmänt:

Ett lösningsblad får endast innehålla redovisningsdelar som hör ihop med en uppgift (dock flera deluppgifter).

För full poäng på de uppgifter som omfattar konstruktioner krävs förutom korrekt funktion även en optimal (minimal) eller nära optimal lösning.
Fungerande men onödigt komplicerade lösningar ger varierande poängavdrag beroende på hur mycket lösningen avviker från den optimala.
För samtliga uppgifter gäller, att ofullständiga lösningar eller lösningar innehållande felaktigheter ger poängavdrag även om resultatet är korrekt.

Betygsättning
För godkänt betyg fordras minst 20 p av totalt 50 p.
20p ≤ **betyg 3** < 30p ≤ **betyg 4** < 40p ≤ **betyg 5**

För godkänt slutbetyg på hela kursen fordras godkänt betyg på tentamen och dessutom fordras godkänd laborationskurs.

Lösningar

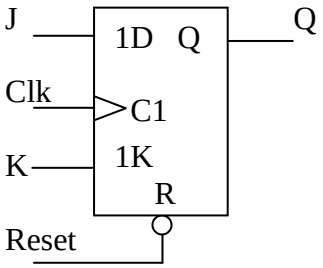
Anslås senast måndag 21 mars, kl 16.00 på kursens hemsida.

Granskning
Av rättning kan göras
onsdag 13/4 kl 11.45 - 12.30 rum E-4102 och fredag 15/4 kl 11.45-12.00 rum E-4102.

1. *Småfrågor (11p)*

a) När du ska koda tillstånden i en tillståndsgraf, vad avgör valet av kod vid implementering i en FPGA? (2p)

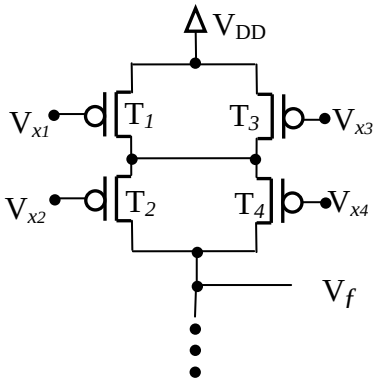
b) Koda JK-vippan enligt figur 1 i VHDL. Reset är asynkron. *(Architeturen räcker)* (3p)



Figur 1: JK-vippa till uppgift 1:c

c) I två process modellen tilldelas en variabel rekorden med insignaler det första som händer i den kombinatoriska processen. Sedan avslutas den kombinatoriska processen med att utsignalerna tilldelas variabeln. Varför gör man så? (3p)

d) Figur 2 visar halva transistornätet för en CMOS krets. Rita den andra halvan som innehåller NMOS transistorerna. (3p)



Figur 2: Transistornät till uppgift 1:e

2. Funktionen nedan är implementerad i en CPLD *(11p)*

$$f = \overline{x_1}\,\overline{x_0} + x_2x_1 + \overline{x_2}\,\overline{x_1}x_0 + \overline{x_2}\,x_1\overline{x_0}$$

a) Ta fram en testvektor som testar ut noden f ”stuck-at-0”. (1p)

b) Ta fram vilka transaktioner som kan leda till hasard. (3p)

c) Ta fram samtliga primimplikanter till funktionen ovan. (3p)

d) Ta fram en minimal hasardfri implementering av funktionen. (2p)

e) Hur kan du modifiera d) för att bibehålla testbarheten. Motivera! (2p)

3. *VHDL (10p)*

VHDL- koden här intill beskriver en arbitrerare.

a) Rita upp en tillståndsgraf utgående från VHDL-koden här intill. (1p)

b) Vilken typ av tillstånds-maskin beskrivs? (1p)

c) Fyll i pulsdiagrammet som finns som bilaga och bifoga den. (4p)

Systemet behöver snabbas upp.

d) Varför reagerar en VHDL implementering med 2 processer snabbare? (1p)
(Givet att det är en bra implementering)

e) Gör en ny implementering av ARCHITECTURE, men med:
- Synkron reset
- 2 processer
- Mealy typ av tillståndsmaskin.
- Enhet 0 skall avbrytas direkt (*dock synkront)* om enhet 3 begär den delade resursen.

Implementeringen skall ge snabbare respons.
I övrigt samma betende.

Redogöra för hur koden här intill skall modifieras.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.ALL;

ENTITY Upg_3 IS
  PORT (R :IN
         std_logic_vector(3 DOWNTO 0);
        Clk,Reset :IN std_logic;
        G :OUT
         std_logic_vector(3 DOWNTO 0));
END Upg_3;

ARCHITECTURE Beha OF Upg_3 IS
TYPE State_Type IS (Start,S0,S1,S2,S3);
SIGNAL Y : State_type;
SIGNAL Cnt :
   std_logic_vector(1 DOWNTO 0);

BEGIN
  PROCESS(Clk,Reset)
  BEGIN
    IF Reset='1' THEN
      Y<=Start;
    ELSIF Clk'event AND Clk='1' THEN
    G<="0000";y <= Start; Cnt <="00";
    CASE y IS
      WHEN Start =>
       IF R(3)='1' THEN Y <= S3;
       ELSIF R(2)='1' THEN Y <= S2;
       ELSIF R(1)='1' THEN Y <= S1;
       ELSIF R(0)='1' THEN Y <= S0;
       END IF;
      WHEN S0 => G(0)<='1';
        Cnt <= Cnt + 1;
        IF (R(0)='1') AND
           (NOT (Cnt = "11")) THEN
          Y<=S0; END IF;
      WHEN S1 => G(1)<='1';
        IF R(1)='1' THEN Y<=S1; END IF;
      WHEN S2 => G(2)<='1';
        IF R(2)='1' THEN Y<=S2; END IF;
      WHEN S3 => G(3)<='1';
        IF R(3)='1' THEN Y<=S3; END IF;
    END CASE;
    END IF;
  END PROCESS;
END Beha;
```

4. *Synkrona sekvensnät (10p)*
En kaffeautomat skall konstrueras.
Den accepterar 1 kr (K – 1 Kr), 5 kr (F – Femma).
Det finns en väljare E där E=1 ger Espresso och E=0 ger vanligt kaffe.
Kaffet kostar 4kr och Espresso 5kr, automaten är utrustad med myntretur för 1kr (Kr).
Insignaler K, F, E och Reset:

K och F är aktiva under en klockcykel när respektive mynt matas in.
(endast ett mynt, bara ett myntinkast).
E ställs in innan köp påbörjas.
Reset : Försätter tillståndsmaskinen i starttillståndet *(asynkron).*

Utsignaler: Ka (matar ut kaffe), Es (matar ut Espresso) och Kr (returnerar en krona).
(Det räcker att utsignalen är aktiv när klockpulsen kommer).

Specifikationen är att automaten skall leverera kaffe när E=0 och erlagt belopp är 4 kr eller mer, samt returnera eventuell växel. Automaten skall leverera Espresso när E=1 och erlagt belopp är 5 kr eller mer, samt returnera eventuell växel. Automaten kan leverera en 1 kr varje klockperiod (förutom kaffe eller Espresso).

Ex : Kunden sätter E=0 och matar in 1kr (K) + 1kr (K) + 1kr (K) + 5kr (F) då blir utsignalen i de följande fyra cyklerna:
(Ka, Kr), (Kr), (Kr) och (Kr).
Därefter befinner sig sekvensnätet i starttillståndet.

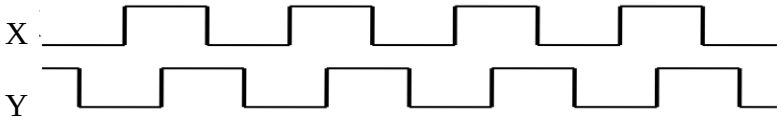
Ni behöver inte ta hänsyn till att någon matar in pengar medans maskinen matar ut Kaffe/Espresso och växel.

a) Rita upp ett tillståndsdiagram enligt specifikationen ovan.
(För full poäng skall den vara av Mealy typ) (4p)

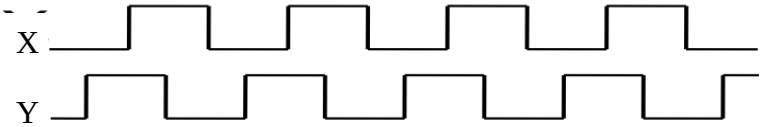
b) Minimera tillståndsdiagrammet från a), om det inte redan är minimalt. (2p)

c) Ställ upp ekvationerna för tillståndsmaskinen kodad enligt ’One hot’+ alla 0. (Starttillståndet kodas med alla tillståndsvariabler 0.)
eller implementera sekvensnätet med VHDL (4p)

5. *Asynkrona sekvensnät (8p, b och c oberoende uppgifter)*
För att bestämma positionen hos en roterande axel används en enkoder. Denna har två utsignaler (X,Y).
Vid medurs vridning



Vid moturs vridning



Konstruera ett asynkront sekvensnät med X och Y som insignaler och en utsignal U som är 0 vid medurs vridning och 1 vid moturs vridning. Ni får anta att man kan försätta nätet i ett starttillstånd som är medurs vridning.

a) Rita upp en tillståndsgraf. *(Behöver inte vara minimal)* (3p)

Flödestabellen nedan som beskriver ett asynkront sekvensnät.
(X, Y insignaler och u_1,u_0 ut signaler)

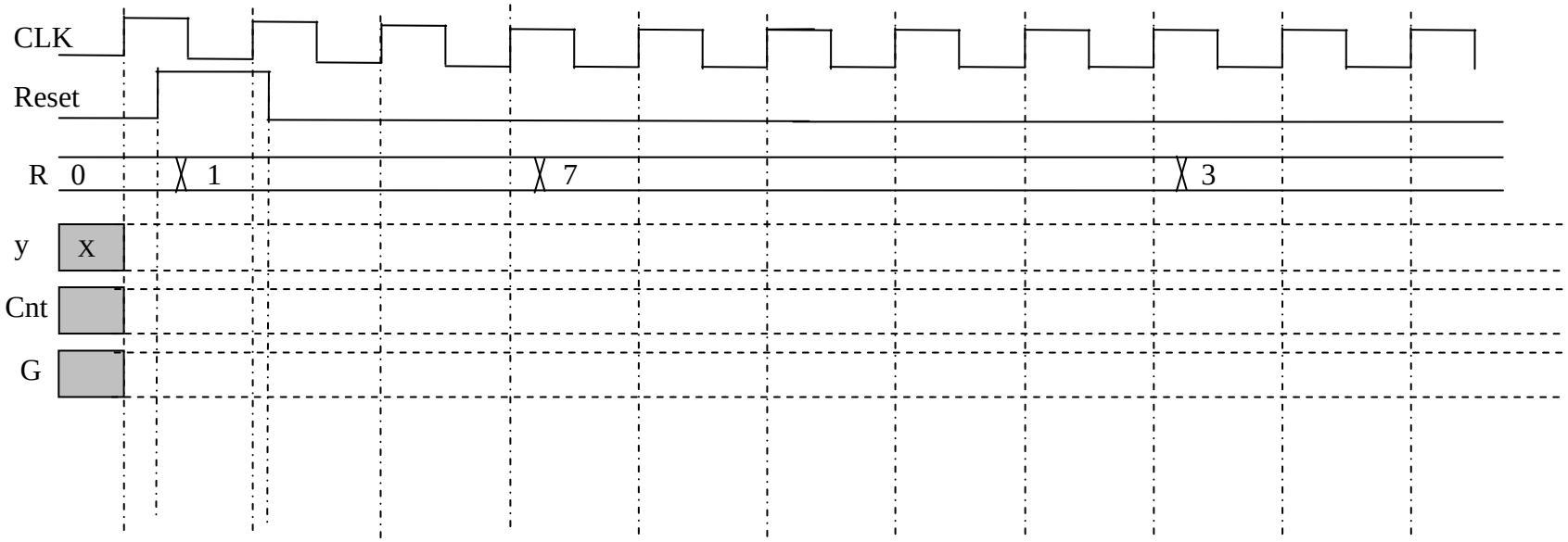
Nuvarande Tillstånd	Nästa tillstånd ($u_1 u_0$)			
	XY			
	00	01	11	10
A	**A(11)**	D(0-)	**A(01)**	C(1-)
B	**B(01)**	D(0-)	**B(11)**	C(1-)
C	A(1-)	-	B(1-)	**C(10)**
D	B(0-)	**D(00)**	A(0-)	-

b) Föreslå en lämplig kodning för tillstånden i tillståndstabellen. (2p)
(Behöver du modifiera tillståndstabellen, för att din kodning skall fungera, skall det klart framgå av lösningen.)

c) Minimera tillståndstabellen ovan för det asynkrona sekvensnätet. (3p)

Kod	Poäng	Sida

Bilaga A : till uppgift 3.



Riv här!

Lösningar:
1 a) En FPGA har många vippor och förhållandevis lite logik, därför är ”One-Hot” kodning lämplig för små och medelstora tillståndsmaskiner.
Stora tillståndsmaskiner försöker man dela ner i mindre som kan implementeras med ”One-Hot”, går inte det så måste man tillgripa snål kodning.

1 b)

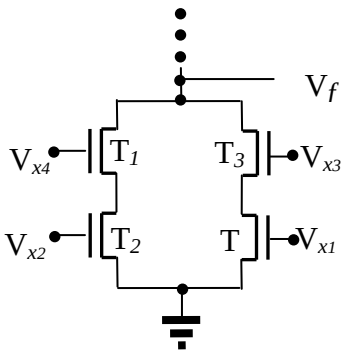
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY JK IS
  PORT ( Clk,Reset,J, K :IN std_logic;
       Q :OUT std_logic);
END JK;

ARCHITECTURE Beha OF JK IS

BEGIN
PROCESS(Clk, Reset)
VARIABLE New_Q : std_logic;
  BEGIN
    IF Reset='0' THEN New_Q:='0';
    ELSIF Clk'event AND Clk='1' THEN
          IF J='1' THEN
                IF K='1' THEN New_Q:= NOT New_Q;
                ELSE New_Q:='1';
                END IF; -- K
          ELSIF K='1' THEN New_Q:='0';
          END IF; -- J
     END IF; -- Reset
    Q <= New_Q;
  END PROCESS;
END Beha;
```

1 c) 1. För att förbättra simulerings effektiviteten, en variabel tilldelning tar direkt medan sen signal tilldelning måste skeduleras.
2. Programmeringstekniskt, variabler beter sig som programmerarna förväntar sig.
3. Vid källkods avlusning så kan man singelsteppa processerna och förstå vad som händer.

1 d) $V_f = (x_2'+x_4')\,(x_1'+x_3')$

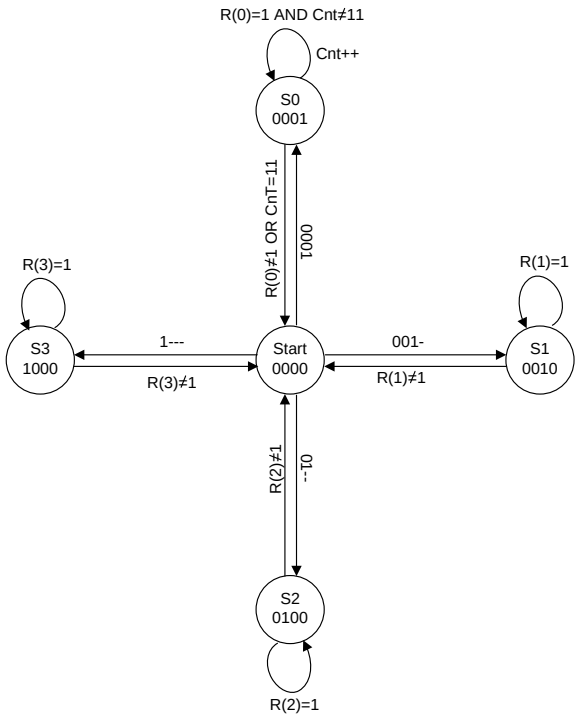
$V_f' = ((x_2'+x_4')\,(x_1'+x_3'))' \rightarrow (x_2'+x_4')' + (x_1'+x_3')' \rightarrow$

$\rightarrow x_2x_4 + x_1x_3$



2 a) Sätt utgången till ett (x_2,x_1,x_0)= -00, 11-. 001 eller 010
2 b) $x_2x_1x_0$ För övergångarna: 000→001, 000→010, 010→110 och 110→1000
2 c) $x_2'x_1'$,x_0',x_2x_1
2 d) $x_2\,x_1 + x_0 + x_2x_1$
2 e) Behövs ej då redundans saknas.

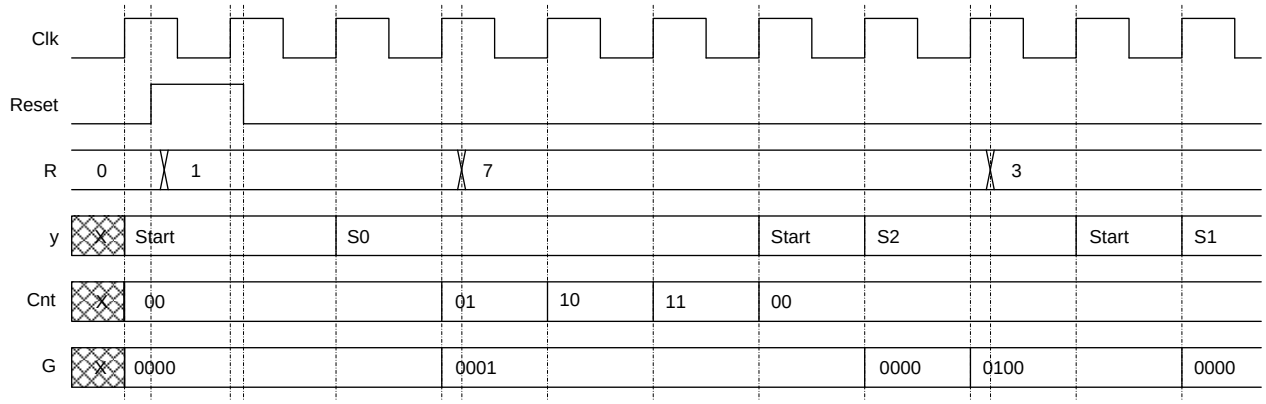
3a) Vi har tillståndsgrafen i *Figur 3a*



Figur 3a Tillståndsgraf

b) Vi har en Mooremaskin då utsignalerna bara beror av aktuellt tillstånd

c) Vi får pulsdiagrammet i *Figur 3b*



Figur 3b Pulsdiagram

d) En kodning med två processer reagerar snabbare då vi gör tillståndsövergången genom att ta ett temporärtillstånd next_state och detta tillstånde tilldelas nytt värde asynkront varför det kan föras över till nästa verkliga tillstånd redan på nästa flank medan kodning med en process och bara en tillståndsvariabel får en fördröjning då vi bara kan trigga på klockflank

e) Med synkron reset, två processer och Mealymodel så får vi VHDL-koden

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY Upg_3e IS
  PORT (R:IN std_logic_vector(3 downto 0);
        Clk:IN std_logic;
        Reset:IN std_logic;
        G:OUT std_logic_vector(3 downto 0));
END Upg_3e;

ARCHITECTURE arch_Upg_3e OF Upg_3e IS
TYPE State_Type IS (Start,S0,S1,S2,S3);
Signal Y:State_type;
SIGNAL Next_Y:State_type;
SIGNAL Cnt:UNSIGNED(1 downto 0);
SIGNAL Next_Cnt:UNSIGNED(1 downto 0);

BEGIN
   mem:PROCESS(Clk)
   BEGIN
      IF Clk'EVENT AND Clk='1' THEN
         IF Reset='1' THEN
            Y<=Start;
            Cnt<="00";
         ELSE
            Cnt<=Next_Cnt;
            Y<=Next_Y;
         END IF;
      END IF;
   END PROCESS mem;

   komb:PROCESS(Y,R,Cnt)
   BEGIN
      G<="0000";
      Next_y <= Start;
      Next_Cnt <="00";
      CASE y IS
         WHEN Start => IF R(3)='1' THEN
                          Next_Y <= S3;
                       ELSIF R(2)='1' THEN
                          Next_Y <= S2;
```

```
                        ELSIF R(1)='1' THEN
                           Next_Y <= S1;
                        ELSIF R(0)='1' THEN
                           Next_Y <= S0;
                        END IF;
         WHEN S0 => G(0)<='1';
                    Next_Cnt <= Cnt + 1;
                    IF (R(3)/='1') AND
                       (R(0)='1') AND
                       (Cnt /= "11") THEN
                        Next_Y<=S0;
                    END IF;
         WHEN S1 => G(1)<='1';
                    IF R(1)='1' THEN
                       Next_Y<=S1;
                    END IF;
         WHEN S2 => G(2)<='1';
                    IF R(2)='1' THEN
                       Next_Y<=S2;
                    END IF;
         WHEN S3 => G(3)<='1';
                    IF R(3)='1' THEN
                       Next_Y<=S3;
                    END IF;
      END CASE;
   END PROCESS;
END Beha;
```

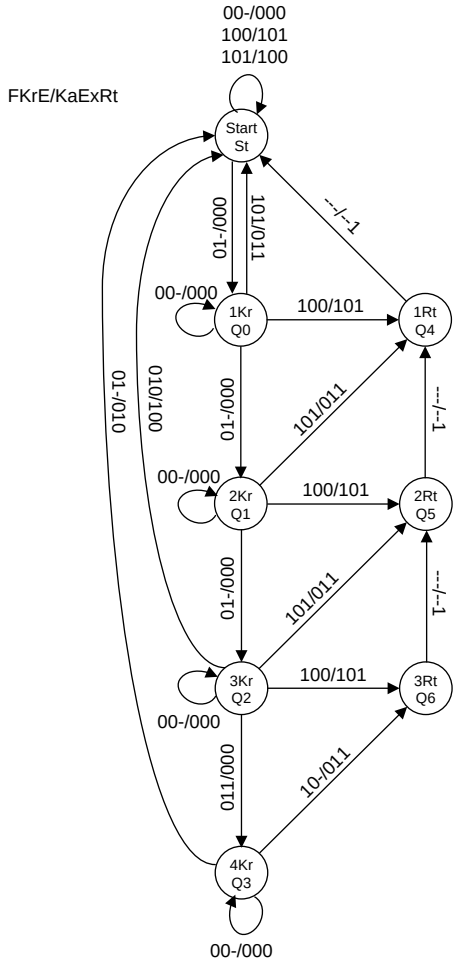
4a) Låt oss se vilka varianter vi har.
För kaffe har vi

Krona + krona + krona + krona → kaffe
Femma → kaffe + en krona retur
Krona + femma → kaffe två kronor retur
Krona + krona + femma → kaffe + 3 kronor retur
Krona + krona+ krona + femma → kaffe + fyra kronor retur

För expresso har vi

Krona + krona + krona + krona + krona → explesso
Femma → expresso
Krona + femma → expresso + en krona retur
Krona + krona + femma → expresso + två kronor retur
Krona + krona + krona + femma → expresso + tre kronor retur
Krona + krona + krona + krona + femma → expresso + fyra kronor retur

Vi kan översätta till tillståndsdiagrammet i *Figur 4a*



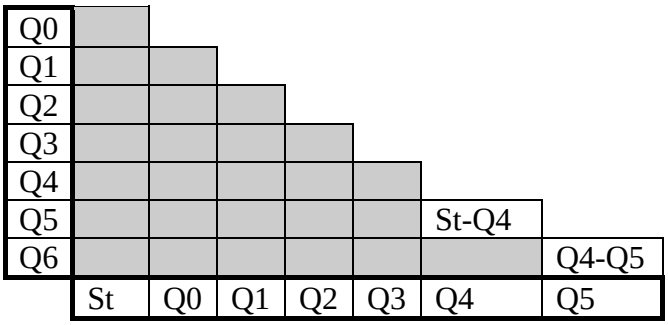
Figur 4a Tillståndsgraf

Vi ställer upp tillståndstabell *Figur 4b*

KaExRt	FKrE							
	000	001	010	011	100	101	110	111
St	St(000)	St(000)	Q0(000)	Q0(000)	St(101)	St(010)	-	-
Q0	Q0(000)	Q0(000)	Q1(000)	Q1(000)	Q4(101)	St(011)	-	-
Q1	Q1(000)	Q1(000)	Q2(000)	Q2(000)	Q5(101)	Q4(011)	-	-
Q2	Q2(000)	Q2(000)	St(100)	Q3(000)	Q6(101)	Q5(011)	-	-
Q3	Q3(000)	Q3(000)	St(010)	St(010)	Q6(011)	Q6(011)	-	-
Q4	St(001)	St(001)	St(001)	St(001)	St(001)	St(001)	-	-
Q5	Q4(001)	Q4(001)	Q4(001)	Q4(001)	Q4(001)	Q4(001)	-	-
Q6	Q5(001)	Q5(001)	Q5(001)	Q5(001)	Q5(001)	Q5(001)	-	-

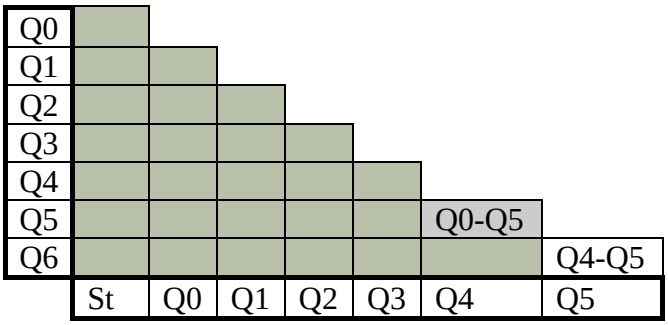
Figur 4b Tillståndstabell

Vi ställer upp relationsgraf *Figur 4c* och markerar de tillståndspar som inte kan slås ihop på grund av skilda utsignaler



Figur 4c Tillståndsgraf

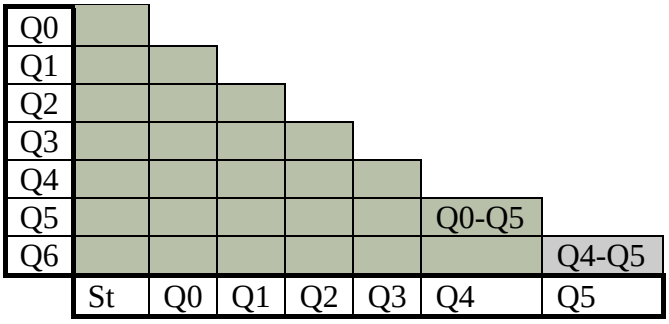
Vi kör ett varv till, *Figur 4d*



Figur 4d Tillståndsgraf

Och måste köra ett varv till, *Figur 4e*

Vår tillståndsgraf går alltså inte att minimera



Figur 4e Tillståndsgraf

Vi börjar med att skriva beteendemässig VHDL-kod direkt utifrån tillståndstabellen

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex4_behavioral IS
  PORT (Reset:IN STD_LOGIC;
        Clk:IN STD_LOGIC;
        F:IN STD_LOGIC;
        Kr:IN STD_LOGIC;
        E:IN STD_LOGIC;
        Ka:OUT STD_LOGIC;
        Ex:OUT STD_LOGIC;
        Rt:OUT STD_LOGIC);
END ex4_behavioral;

ARCHITECTURE arch_ex4_behavioral Of ex4_behavioral IS
TYPE State_Type IS (St,Q0,Q1,Q2,Q3,Q4,Q5,Q6);
Signal Q:State_type;
SIGNAL Next_Q:State_type;
SIGNAL Output:STD_LOGIC_VECTOR(2 downto 0);
SIGNAL Next_Output: STD_LOGIC_VECTOR (2 downto 0);

BEGIN
   mem:PROCESS(Clk)
   BEGIN
      IF Clk'EVENT AND Clk='1' THEN
         IF Reset='1' THEN
            Q<=St;
            Output<=(OTHERS=>'0');
         ELSE
            Output<=Next_Output;
            Q<=Next_Q;
         END IF;
      END IF;
   END PROCESS mem;

   komb:PROCESS(Kr,F,E,Q)
   BEGIN
      Next_Output<=(OTHERS=>'0');
      CASE Q IS
         WHEN St => IF Kr='1' THEN
                       Next_Q<= Q0;
                    ELSIF F='1' THEN
                       Next_Q <= St;
                       IF E='1' THEN
                          Next_Output<="010";
                       ELSE
                          Next_Output<="101";
                       END IF;
                    END IF;
```

```
        WHEN Q0 => IF Kr='1' THEN
                      Next_Q<=Q1;
                   ELSIF F='1' THEN
                      IF E='1' THEN
                         Next_Q<=St;
                         Next_Output<="011";
                      ELSE
                         Next_Q<=Q4;
                         Next_Output<="101";
                      END IF;
                   END IF;
        WHEN Q1 => IF Kr='1' THEN
                      Next_Q<=Q2;
                   ELSIF F='1' THEN
                      IF E='1' THEN
                         Next_Q<=Q4;
                      ELSE
                         Next_Q<=Q5;
                      END IF;
                   END IF;
        WHEN Q2 => IF Kr='1' THEN
                      IF E='1' THEN
                         Next_Q<=Q3;
                      ELSE
                         Next_Q<=St;
                         Next_Output<="100";
                      END IF;
                   ELSIF F='1' THEN
                      IF E='1' THEN
                         Next_Q<=Q5;
                         Next_Output<="011";
                      ELSE
                         Next_Q<=Q6;
                         Next_Output<="101";
                      END IF;
                   END IF;
        WHEN Q3 => IF Kr='1' THEN
                      Next_Q<=St;
                      Next_Output<="010";
                   ELSIF F='1' THEN
                      Next_Q<=Q6;
                      Next_Output<="011";
                  END IF;
        WHEN Q4 => Next_Q<=St;
                   Next_Output<="001";
        WHEN Q5 => Next_Q<=Q4;
                   Next_Output<="001";
        WHEN Q6 => Next_Q<=Q5;
                   Next_Output<="001";
    END CASE;
END PROCESS;
```

```
   Ka<=output(2);
   Ex<=output(1);
   Rt<=output(0);
END arch_ex4_behavioral;
```

och skriver en do-fil för simulering

```
-- ex4_behavioral.do
restart -f -nowave
view signals wave
add wave Clk Reset F Kr E Next_Q Q Next_Output Output Ka
Ex Rt
force Clk 0 0,1 50ns -repeat 100ns
force Reset 0
force F 0
force Kr 0
force E 0
run 225ns
force Reset 1
run 200ns
force Reset 0
run 200ns
force F 1
run 100ns
force E 1
run 100ns
force F 0
run 100ns
force Kr 1
run 500ns
force Kr 1
run 300ns
force Kr 0
force F 1
run 100ns
force F 0
run 400ns
```

Om vi använder one-hotkodning med bara nollor i starttillståndet så får vi villkoren

$$St^+ = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot \overline{Q_3} \cdot \overline{Q_4} \cdot \overline{Q_5} \cdot \overline{Q_6}$$

$$Q_0^+ = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot \overline{Q_3} \cdot \overline{Q_4} \cdot \overline{Q_5} \cdot \overline{Q_6} \cdot K + Q_0 \cdot \overline{K} \cdot \overline{F}$$

$$Q_1^+ = Q_0 \cdot K + Q_1 \cdot \overline{K} \cdot \overline{F}$$

$$Q_2^+ = Q_1 \cdot K + Q_2 \cdot \overline{K} \cdot \overline{F}$$

$$Q_3^+ = Q_2 \cdot K + Q_3 \cdot \overline{K} \cdot \overline{F}$$

$$Q_4^+ = Q_0 \cdot F \cdot \bar{E} + Q_1 \cdot F \cdot E + Q_5$$

$$Q_5^+ = Q_1 \cdot F \cdot \bar{E} + Q_2 \cdot F \cdot E + Q_6$$

$$Q_6^+ = Q_2 \cdot F \cdot \bar{E} + Q_3 \cdot F \cdot E$$

$$Ka = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot \overline{Q_3} \cdot \overline{Q_4} \cdot \overline{Q_5} \cdot \overline{Q_6} \cdot F \cdot \bar{E} +$$

$$+Q_0 \cdot F \cdot \bar{E} + Q_1 \cdot F \cdot \bar{E} + Q_2 \cdot F \cdot \bar{E} + Q_2 \cdot K \cdot \bar{E}$$

$$Ex = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot \overline{Q_3} \cdot \overline{Q_4} \cdot \overline{Q_5} \cdot \overline{Q_6} \cdot F \cdot E +$$

$$+Q_0 \cdot F \cdot E + Q_1 \cdot F \cdot E + Q_2 \cdot F \cdot E + Q_2 \cdot K \cdot \bar{E} + Q_3 \cdot K + Q_3 \cdot F$$

$$Kr = \overline{Q_0} \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot \overline{Q_3} \cdot \overline{Q_4} \cdot \overline{Q_5} \cdot \overline{Q_6} \cdot F \cdot \bar{E} +$$

$$+Q_0 \cdot F + Q_1 \cdot F + Q_2 \cdot F + Q_3 \cdot F + Q_4 + Q_5 + Q_6$$

och vi skriver även VHDL från dessa villkor

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ex4 IS
  PORT (Reset:IN STD_LOGIC;
        Clk:IN STD_LOGIC;
        F:IN STD_LOGIC;
        Kr:IN STD_LOGIC;
        E:IN STD_LOGIC;
        Ka:OUT STD_LOGIC;
        Ex:OUT STD_LOGIC;
        Rt:OUT STD_LOGIC);
END ex4;

ARCHITECTURE arch_ex4 OF ex4 IS
SIGNAL Q:STD_LOGIC_VECTOR(6 DOWNTO 0);

BEGIN
  PROCESS(Clk,Reset)
  BEGIN
     IF Reset='1' THEN
        Q<=(OTHERS=>'0');
     ELSIF Clk'event AND Clk='1' THEN
        Q(0)<=(NOT(Q(6)) AND NOT(Q(5)) AND
              NOT(Q(5)) AND NOT(Q(4)) AND
              NOT(Q(3)) AND NOT(Q(2)) AND
              NOT(Q(1)) AND NOT(Q(0)) AND Kr) OR
               (Q(0) AND NOT(Kr) AND NOT(F));
        Q(1)<=(Q(0) AND Kr) OR
```

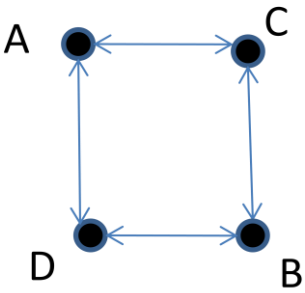
```
                (Q(1) AND NOT(Kr) AND NOT(F));
            Q(2)<=(Q(1) AND Kr) OR
                (Q(1) AND NOT(Kr) AND NOT(F));
            Q(3)<=(Q(2) AND Kr) OR
                (Q(3) AND NOT(Kr) AND NOT(F));
            Q(4)<=(Q(0) AND F AND NOT(E)) OR
                (Q(1) AND F AND E) OR Q(5);
            Q(5)<=(Q(1) AND F AND NOT(E)) OR
                (Q(2) AND F AND E) OR Q(6);
            Q(6)<=(Q(2) AND F AND NOT(E)) OR
                (Q(3) AND F AND E);
            Ka<=(NOT(Q(6)) AND NOT(Q(5)) AND
                NOT(Q(5)) AND NOT(Q(4)) AND
                NOT(Q(3)) AND NOT(Q(2)) AND
                NOT(Q(1)) AND NOT(Q(0)) AND F AND NOT(E)) OR
                (Q(0) AND F AND NOT(E)) OR
                (Q(1) AND F AND NOT(E)) OR
                (Q(2) AND F AND NOT(E)) OR
                (Q(2) AND Kr AND NOT(E));
            Ex<=(NOT(Q(6)) AND NOT(Q(5)) AND
                NOT(Q(5)) AND NOT(Q(4)) AND
                NOT(Q(3)) AND NOT(Q(2)) AND
                NOT(Q(1)) AND NOT(Q(0)) AND F AND E) OR
                (Q(0) AND F AND E) OR
                (Q(1) AND F AND E) OR
                (Q(2) AND F AND E) OR
                (Q(3) AND Kr) OR
                (Q(3) AND F);
            Rt<=(NOT(Q(6)) AND NOT(Q(5)) AND
                NOT(Q(5)) AND NOT(Q(4)) AND
                NOT(Q(3)) AND NOT(Q(2)) AND
                NOT(Q(1)) AND NOT(Q(0)) AND F AND NOT(E)) OR
                (Q(0) AND F) OR
                (Q(1) AND F) OR
                (Q(2) AND F) OR
                (Q(3) AND F) OR Q(4) OR Q(5) OR Q(6);
          END IF;
        END PROCESS;

    END arch_ex4;
```

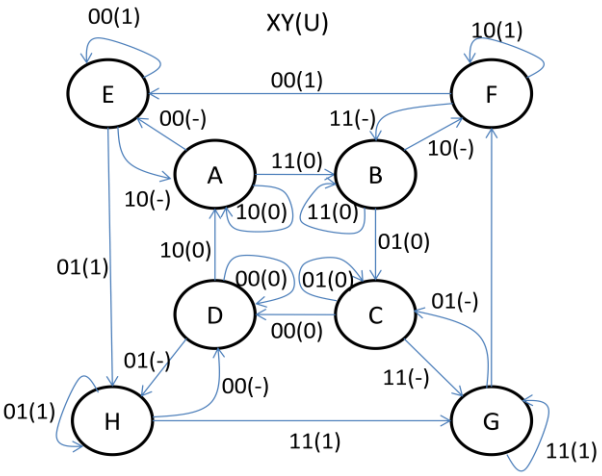
Vi simulerar även denna kod med en `do`-fil

```
-- ex4.do
restart -f -nowave
view signals wave
add wave Clk Reset F Kr E Q Ka Ex Rt
force Clk 0 0,1 50ns -repeat 100ns
force Reset 0
force F 0
force Kr 0
```

```
force E 0
run 225ns
force Reset 1
run 200ns
force Reset 0
run 200ns
force F 1
run 100ns
force E 1
run 100ns
force F 0
run 100ns
force Kr 1
run 500ns
force Kr 1
run 300ns
force Kr 0
force F 1
run 100ns
force F 0
run 400ns
```

6 a)

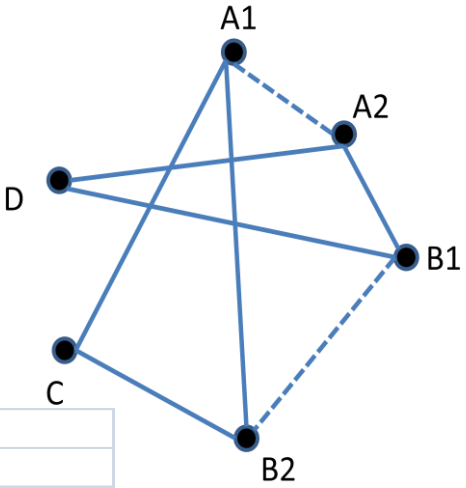




6 b) A 00
C 01
B 11
D 10

6 c)

δ(λ)	$x_1x_0(u_1u_0)$			
	0	1	11	10
A_1	**A_1(11)**	D(--)	-	C(1-)
A_2	-	D(0-)	**A_2(01)**	C(--)
B_1	**B_1(01)**	D(0-)	-	C(--)
B_2	-	D(--)	**B_2(11)**	C(1-)
C	A_1(1-)	-	B_2(1-)	**C(10)**
D	B_1(0-)	**D(00)**	A_2(0-)	-



A2					
B1	X				
B2		X			
C		X	X		
D	X			X	X
	A1	A2	B1	B2	C

δ(λ)	$x_1x_0(u_1u_0)$			
	0	1	11	10
A_{1B2C}	**A_{1B2C}(11)**	A_{2B1D}(--)	**A_{1B2C}(11)**	**A_{1B2C}(10)**
A_{2B1D}	**A_{2B1D}(01)**	**A_{2B1D}(00)**	**A_{2B1D}(01)**	A_{1B2C}(--)