Tentamen i
Digitala system - EDI610 15hp
varav denna tentamen 4,5hp

Institutionen för elektro- och informationsteknik
Campus Helsingborg, LTH

2016-12-22 8.00 - 13.00

Uppgifterna i tentamen ger totalt 60 poäng. Uppgifterna är inte ordnade på något speciellt sätt. Läs därför igenom alla uppgifter innan du börjar lösa dem. Några uppgifter är uppdelade i deluppgifter. Av totalt 60 möjliga poäng fordras minst 30 för godkänt.

Inga hjälpmedel är tillåtna

Observera!

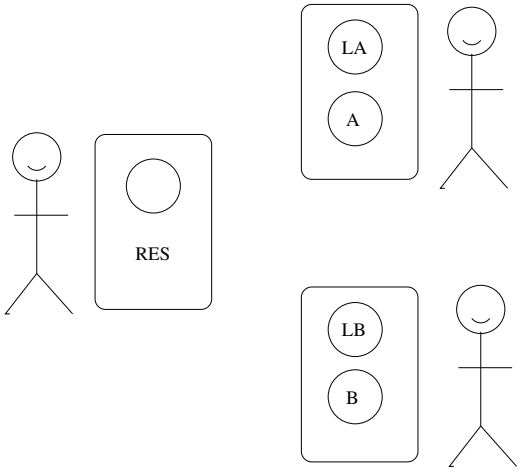
- För att rättning av lösning skall komma i fråga fordras att den är läslig samt klart och tydligt uppställd.
- Glöm inte att skriva logo och koden på alla blad.
- Alla lösa blad ska vara samlade i omslaget
- Lösningarna ska vara numrerade och ordnade i nummerföljd

Lycka till!

1. (a) Omvandla 56_{10} till basen 2 (2 p)
   (b) Omvandla 56_{10} till basen 16 (2 p)
   (c) Vilket är talområdet för ett åtta-bitars tal med teckenbit respektive utan teckenbit? (2 p)
   (d) Negativa tal representaeras ofta med 2-komplement. Skriv 2-komplementrepresentationen av talet 56, där 56 är representerat som ett 8-bitars positivt tal (2 p)
   (e) Utför subtraktionen av $50_{10} - 56_{10}$ binärt och visa tydligt hur du omvandlar svaret till basen 10 (2 p)

2. En 2-bitars ADomvandlare av typen Flash har tre komparatorer som ger '1' ut om dess insignal överstiger referensspänningen. Utsignalen, $x = x_2, x_1, x_0$ antar fyra relevanta värden: $x = 000, x = 001$, $x = 011$, och $x = 111$. Övriga kombinationer är *don't care*. Det finns även en ingång, enable *en*, aktiv hög som vid låg ger $u = 00$ och vid hög ger u dess rätta värden. Konstruera ett kombinatoriskt nät med fyra insignaler, x_i och *en* och två utsignaler: u_1, u_0 som är en binärkodning av de fyra fallen. Visa Karnaugh-diagrammen och uttryck för de minimala uttrycken. (10 p)

3. I ett frågeprogram på TV skall två tävlande, A och B, svara på frågor. De tävlande har vars en knapp och vars en lampa. Programledaren börjar läsa frågan och då en tävlande tror sig kunna svaret trycker den på sin knapp för att tända sin lampa. När dennes lampa är tänd kan inte den andre tävlande tända sin lampa. Att de trycker samtidigt anses inte kunna hända och behandlas som *don't care*. Systemet nollställs (lampan släcks) genom att programledaren trycker på sin resetknapp (`RES`), se figur 1. Konstruera ett minimalt sekvensnät som styr lamporna. (10 p)



Figur 1: Fågetävling

4. Nedan ser du en VHDL-fil (dec16_prov) som beskriver ett sekvensnät.
   (a) Rita tillståndsgrafen för nätet. (3 p)
   (b) Skriv sekvensmaskinen på minimal form (SP-form). Koda tillstånden med NBCD-kod (6 p)
   (c) Är sekvensnätet på Moore- eller Mealyform (1 p)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dec16_prov is
    Port (clock, x, reset : in  STD_LOGIC;
               u : out  STD_LOGIC);
end dec16_prov;
architecture Behavioral of dec16_prov is
type state_type is (s0,s1,s2,s3,s4);
signal present_state, next_state:state_type;
begin
  process(present_state,x,reset)
  begin
    if reset='1' then
      next_state<=s0;
    else
      case present_state is
           when s0 => if x='0' then next_state<=s1;
                      else next_state<=s3;
                      end if;
           when s1 => if x='0' then next_state<=s2;
                      else next_state<=s1;
                      end if;
           when s2 => if x='0' then next_state<=s3;
                      else next_state<=s0;
                      end if;
           when s3 => if x='0' then next_state<=s3;
                      else next_state<=s2;
                      end if;
      end case;
    end if;
  end process;
  process(present_state, x)
    begin
      if present_state=s2 then
         if x='0' then u<='1';
         else u<='0';
         end if;
     else
         u<='0';
     end if;
  end process;
  process(clock)
    begin
      if rising_edge(clock) then
         present_state<=next_state;
      end if;
  end process;
end  architecture Behavioral;
```

5. Antag en processor som har en klockfrekvens på 1kHz och där instruktioner anges i ett 3-adress format. För t.ex. ALU operationer anges instruktioner enligt: Operation (OP) Destination (DES), Operand1 (OP1), Operand2 (OP2) och utförs enligt följande:
DES = OP1 OP OP2. Om assemblerinstruktionen är ADD R1, R2, R3 kommer detta att utföras: R1=R2 ADD R3, d v s innehållet i register 2 (R2) kommer adderas till innehållet i register 3 (R3) och resultatet lagras i register 1 (R1). För programmet som är givet nedan, besvara: (10 p)

   (a) Hur många instruktioner hinner processorn exekvera på 1 sekund om varje instruktion tar 10 klockcykler (bortse från minnesaccesser)?

   (b) Hur många bitar behövs för att lagra en heltalsvariabel med värden mellan 0 och 65534?

   (c) Vilka instruktioner påverkar status-flaggor, t ex Z-flaggan?

   (d) Vilka instruktioner påverkas av status-flaggor, t ex Z-flaggan?

   (e) Vilket är det binära värdet av det som finns lagrat på adress 10010-10101 om lagringen är enligt Big.endian?

ADRESS	INSTRUKTION	FÖRKLARING
00000	XOR R1, R1, R1	R1← R1 XOR R1
00010	ADD R2, R1, 17	R2←R1 + 17
00100	MUL R2, R1, R2	R2←R1*R2
00110	BEZ 10000	IF ZERO BRANCH TO 10000
01000	SUB R2, R2, 1	R2←R2 - 1
01010	BR 10000	BRANCH TO 10000
01100	ADD R1, R1, 1	R1 ← R1 + 1
01110	MUL R1, 5, 2	R1 ← 5 MUL 2
10000	AND R1, R1, R2	R1 ← R1 AND R2
10010	0b00001111	Binärt
10011	0x10	Hexadecimalt
10100	0b0	Binärt
10101	5	Decimalt

6. På en processor som har en pipeline med följande steg: Fetch instruction (FI), Decode instruction (DI), Calculate operand address (CO), Fetch operand (FO), Execute instruction (EI) och Write operand (WO) ska följande program exekvera: (10 p)

LOAD R1, X	Läs in det som finns på adress X till register R1
ADD R2, R4	R2←R2+R4
ADD R4, R3	R4←R4+R3
SUB R2, R4	R2←R2-R4
LOAD R1, X	Läs in det som finns på adress X till register R1
ADD R2, R1	R2←R2+R1
MUL R3, R4	R3←R3*R4
SUB #1, R2	R2←R2-1
ADD R1, R2	R1←R1+R2
BEZ TAR	Branch om zero
MOVE #10, R1	R1←10

   (a) Återge och förklara den första strukturella hazarden som uppkommer

   (b) Återge och förklara den första data hazarden som uppkommer

   (c) Återge och förklara den första kontroll hazarden som uppkommer

   (d) Återge och ändra den första uppkomsten där delay load kan användas (visa hur assemblykoden kan ändras)

   (e) Återge och ändra den första uppkomsten där delayed branching (visa hur assemblykoden ändras)

Svar till tentamen i Digitala system - EDI610 15hp, 2016-12-22

1a $56_{10} = 111000_2$

1b $56_{10} = 38_{16}$

1c (-128 - +127) med teckenbit och (0 - 255) utan teckenbit

1d 2-komplementet av 56 ($= 00111000_2$) är 11001000

1e $50_{10} = 00110010_2$ adderas med 2-komplementet av 56. $00110010 + 11001000 = 11111010$
Det är ett negativt tal som konvertrat till positivt blir $00000110_2 = 6_{10}$ d.v.s talet är -6.

2

x_2	x_1	x_0	u_1	u_0
0	0	0	0	0
0	0	1	0	1
0	1	0	-	-
0	1	1	1	0
1	0	0	-	-
1	0	1	-	-
1	1	0	-	-
1	1	1	1	1

en sättes in i slututtrycket för u_i

u_1		x_1x_0			
		00	01	11	10
x_2	0	0	0	1	-
	1	-	-	1	-

Tabell 1: $u_1 = en(x_1)$

u_0		x_1x_0			
		00	01	11	10
x_2	0	0	1	0	-
	1	-	-	1	-

Tabell 2: $u_0 = en(x_2 + x_1'x_0)$

3

	A	B	q_1	q_0	q_1^+	q_0^+	LA	LB	
Vila	0	0	0	0	0	0	0	0	
	0	0	0	1	0	1	1	0	
	0	0	1	0	1	0	0	1	
	0	0	1	1	-	-	-	-	
B trycker	0	1	0	0	1	0	0	0	B trycker först
	0	1	0	1	0	1	1	0	B trycker men A var först
	0	1	1	0	1	0	0	1	B först och trycker
	0	1	1	1	-	-	-	-	
A trycker	1	0	0	0	0	1	0	0	A trycker först
	1	0	0	1	0	1	1	0	A först och trycker
	1	0	1	0	1	0	0	1	A trycker men B var först
	1	0	1	1	-	-	-	-	
Finns ej	1	1	0	0	-	-	-	-	
	1	1	0	1	-	-	-	-	
	1	1	1	0	-	-	-	-	
	1	1	1	1	-	-	-	-	

'RES' implementeras direkt i uttrycken för q_i.

q_1^+		q_1q_0			
		00	01	11	10
A, B	00	0	0	-	1
	01	1	0	-	1
	11	-	-	-	-
	10	0	0	-	1

Tabell 3: $q_1^+ = q_1 + Bq_0'$

q_0^+		q_1q_0			
		00	01	11	10
A, B	00	0	1	-	0
	01	0	1	-	0
	11	-	-	-	-
	10	1	1	-	0

Tabell 4: $q_0^+ = q_0 + Aq_1'$

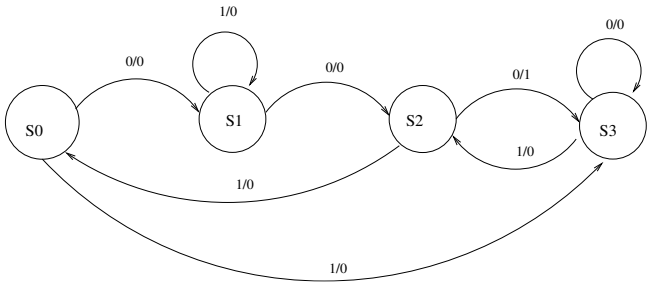
LA		q_0	
		0	1
q_1	0	0	1
	1	0	0 (-)

Tabell 5: $LA = q_1'q_0$ alt. $LA = q_0$

LB		q_0	
		0	1
q_1	0	0	0
	1	1	0 (-)

Tabell 6: $LB = q_1q_0'$ alt. $LB = q_1$

4a



Figur 2: Tillståndsdiagram

4b

x	q_1	q_0	q_1+	q_0+
0	0	0	0	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	1
1	1	0	0	0
1	1	1	1	0

q_1^+		q_1q_0			
		00	01	11	10
x	0	0	1	1	1
	1	1	0	1	0

Tabell 7: $q_1^+ = x'(q_1 + q_0) + q_1q_0 + xq_1'q_0'$

q_0^+		q_1q_0			
		00	01	11	10
x	0	1	0	1	1
	1	1	1	0	0

Tabell 8: $q_0^+ = q_1'q_0' + xq_1' + x'q_1$

4c Sekvensnätet är på Mealyform

5a Om processorn exekverar med en klockfrekvens på $1kHz$, dvs $1000Hz$, så är en klockperiod 1/1000. Om varje instruktion tar 10 klockcykler så blir uttrycket: $1sekund = 10per\,instruktion * x * 1/1000$ och löser man ut x får man att $x = 1000/10 = 100$, dvs processorn hinner exekvera 100 instruktioner på 1 sekund.

5b Om man har 1 bit kan man lagra talen 0 och 1, har man 2 bitar kan man lagra talen 0, 1, 2, och 3, har man 3 bitar kan man lagra talen 0, 1, 2, ... 7, har man 4 bitar kan man lagra talen 0, 1, ... 15, osv. Man kan också sätta upp ekvationen: $2^x = 65534$, vilket löses som: $x = \log(65534)/\log(2)$ och avrunda detta uppåt. Det viktiga är att få fram uttrycket.

5c Instruktioner som påverkar status-flaggor, t ex Z-flaggan är instruktioner som använder ALU:n. I exemplet nedan använder sig t ex XOR, ADD, och MUL av ALU:n. Dessa instruktioner kommer alltså påverka statusflaggor.

5d Instruktioner som påverkas av statusflaggor är vanligen hopp-instruktioner där statusflaggan används för att bestämma ett villkor. I exemplet nedan är BEZ - IF ZERO BRANCH TO 10000 – ett exempel där ett hopp kommer ske om ZERO flaggan är satt.

5e Om man antar att minnet är byte-adresserat, vilket är vanligast, är det ett värde som ligger lagrat på adresserna: 10010, 10011, 10100 och 10101. Om ett tal lagras enligt *Big.endian* är det som är lagrat på lägst adress det som svarar mot de mest signifikanta bitarna. Vi räknar ut vad som finns på varje adress. På adress: 10010 finns bitarna: 00001111, på adress 10011 finns det hexadecimalatalet 10, vilket motsvaras av 00010000, på adress 10100 ligger talet 0, dvs 00000000, och på adress 10101 ligger talet 5, vilket är 00000101. Vi sätter samman dessa tal, och får: 00001111000010000000000000000101

6a Den första strukturella hazarden uppkommer vid instruktion 1 och 2:

LOAD R1, X	Läs in det som finns på adress X till register R1
ADD R2, R4	R2←R2+R4

och problemet är att när LOAD R1, X vill läsa data i minnet är det samtidigt dags att läsa in instruktionen: ADD R2, R4.

6b Den första data hazarden som uppkommer är vid instruktion 3 och 4:

ADD R4, R3	R4←R4+R3
SUB R2, R4	R2←R2-R4

Det som är problematiskt är att instruktionen SUB R2, R4 vill använda resultatet, nämligen R4, från den tidigare instruktionen, som inte är färdigt när SUB R2, R4 vill ha resultatet.

6c Den första kontroll hazarden uppkommer vid instruktion 10 som är: BEZ TAR Branch om zero. Det som är problematiskt är att det inte är säkert att nästa instruktion som ska exekvera är: MOVE #10, R1 R1←10

6d Vid första instruktionen LOAD R1, X har vi en LOAD, men här är det inget problem eftersom R1 inte används i nästa instruktion. Men, för instruktion 5: LOAD R1, X så är det så att nästa instruktion använder R1. Det kommer alltså att bli problem. För att undvika detta problem så kan man ändra ordningen på instruktionerna. Från detta:

LOAD R1, X	Läs in det som finns på adress X till register R1
ADD R2, R1	R2←R2+R1
MUL R3, R4	R3←R3*R4

till detta:

LOAD R1, X	Läs in det som finns på adress X till register R1
MUL R3, R4	R3←R3*R4
ADD R2, R1	R2←R2+R1

Genom denna ändring så används inte R1 förrän lite senare.

6e Den första uppkomsten med delayed branching är vid instruktionen: BEZ TAR och här kan exekveringen ändras från detta:

MUL R3, R4	R3←R3*R4
SUB #1, R2	R2←R2-1
ADD R1, R2	R1←R1+R2
BEZ TAR	Branch om zero
MOVE #10, R1	R1←10

till detta:

SUB #1, R2	R2←R2-1
ADD R1, R2	R1←R1+R2
BEZ TAR	Branch om zero
MUL R3, R4	R3←R3*R4
MOVE #10, R1	R1←10

Det som ändrats är att instruktionen MUL R3, R4 har flyttats så att den är placerad efter BEZ TAR, men trots detta kommer instruktionen MUL R3, R4 alltid exekveras.