UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430/INF4430 Digital systemkonstruksjon**
Eksamensdag:	**30. november 2005**
Tid for eksamen:	**9 - 12**
Oppgavesettet er på 4 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Ingen**

Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1 – 14 (flervalgsoppgaver) som skal besvares med skjemaet som er vedlagt etter oppgaveteksten og oppgave 15 som besvares på vanlige ark. Oppgave 1 - 14 har til sammen vekt på 40%, mens oppgave 15 har vekt på 60%.

Generelt for oppgave 1- 14:
Hver oppgave består at et tema/spørsmål i venstre kolonne og en del utsagn hver angitt med en stor bokstav. Oppgavene besvares ved å merke tydelige kryss (X) i rett kolonne for riktig svaralternativ (dvs. at et utsagn er sant) i skjemaet i vedlegget. Det er alltid *minst* en riktig avmerking for hver oppgave, men det er ofte *flere* riktige avmerkninger. *For å få best karakter skal man sette flere kryss i en oppgave hvis det er flere riktige svar.* Det gis 2 poeng for hver avkrysning der det skal være avkrysning. Det gis -1 poeng for hver avkrysning der det ikke skal være avkrysning. Mangel på kryss der det skal være kryss gir også -1 poeng. Du kan benytte høyre kolonne i oppgaveteksten til kladd. Skjema påført ditt kandidatnummer i vedlegget er din besvarelse.

Oppgave 1

Lagring av konfigurasjon i FPGA med SRAM teknologi	A	Krever liten plass	
	B	Krever innlesning av konfigurasjon ved oppstart	
	C	Kan programmeres et uendelig antall ganger	
	D	Er en ekstra pålitelig teknologi	
	E	Anvendes ofte i FPGAer	

Oppgave 2

Lagring av konfigurasjon i FPGA med Anti-fuse teknologi	A	Krever høy programmeringsspenning	
	B	Kan slettes med UV-lys	
	C	Krever egen programmeringsenhet	
	D	Krever liten plass	
	E	Gjør det kun mulig å programmere FPGA en gang	

Oppgave 3

Tidsforsinkelse i moderne FPGAer	A	Forbindelseslinjer har større tidsforsinkelse enn LUTer	
	B	Forbindelseslinjer har omtrent lik tidsforsinkelse som LUTer	
	C	Forbindelseslinjer har mindre tidsforsinkelse enn LUTer	
	D	Både forbindelseslinjer og LUTer har ubetydelig tidsforsinkelse	

Oppgave 4

Simulering	A	Simulering er viktig for å finne feil tidlig	
	B	Simulering gjøres enklest etter at FPGAen er programmert	
	C	Simulering er en type verifisering	
	D	Simulering med tidsforsinkelser er raskere enn uten	

Oppgave 5

Testbenker	A	Testbenker lager input-stimuli til krets som skal simuleres	
	B	Testbenker kan benytte hele VHDL-språket	
	C	Testbenker syntetiseres ofte	
	D	Testbenker kan sjekke utganger på krets som skal simuleres	

Oppgave 6

Syntese	A	Synteseprosessen lager en nettliste på portnivå	
	B	Resultat fra synteseprosessen er teknologiuavhengig	
	C	Syntheseverktøyet benyttes til å simulere kretsen man utvikler	

Oppgave 7

Myke og harde kjerner	A	FPGA kan inneholde myke kjerner	
	B	Harde kjerner kan fjernes fra FPGAen hvis de ikke benyttes	
	C	En hard kjerne tar mindre fysisk plass enn en tilsvarende myk kjerne	

Oppgave 8

Prosessorkjerne	A	Er en prosessor som kan inngå som en del av en FPGA	
	B	Prosessorer i Xilinx FPGA er av merke Intel	
	C	Er en prosessor som kan utføre/eksekvere et program	
	D	MicroBlaze er en hard prosessorkjerne	

Oppgave 9

Block RAM (BRAM)	A	Dette er blokker av RAM i FPGA som kan lagre program	
	B	BRAM krever kortere tid for lesing og skriving enn ekstern RAM	
	C	BRAM som ikke brukes kan fjernes fra FPGAen	
	D	Dette er blokker av RAM som kan lagre data/variable	

Oppgave 10

Hvorfor er det gunstig å implementere komplette system med logikk, minne og prosessor på en krets?	A	Kompakt løsning siden antall kretser blir mindre	
	B	Det er ofte umulig å få tak i løse komponenter	
	C	Implementering på en FPGA gir rom for fleksibel tilpasning av enhetene	
	D	Mindre tidsforsinkelse mellom enhetene	
	E	Mindre effektforbruk	

Oppgave 11

Embedded Development Kit (EDK)	A	Gjør det mulig å designe med prosessor på FPGA	
	B	Det er lettere å bruke ISE og en uavhengig C-kompilator for design med prosessor	
	C	On-chip Peripheral Bus (OPB) kan inngå i design	
	D	EDK håndterer ikke bruk av BRAM	
	E	EDK håndterer bruk av virtuelle komponter/IP'er	

Oppgave 12

Ulemper ved lavnivådesignspråk som VHDL i forhold til bruk av høynivåspråk for maskinvaredesign.	A	Vanskelig å definere detaljer i koden	
	B	Mer regnekrevende å simulere	
	C	Språk er mindre egnet til design av både maskinvare og programvare (HW/SW co-design)	
	D	Tidkrevende å designe maskinvare med	

Oppgave 13

Hvorfor kan et C/C++ språk som er minimalt utvidet være gunstig for maskinvaredesign?	A	Koden kan implementeres på høyt abstraksjonsnivå	
	B	Simulering blir grundig, selv om den tar lang tid	
	C	Flere forskjellige implementeringer kan effektivt evalueres	
	D	Kode er uavhengig av målplattform	

Oppgave 14

Design med ASIC og FPGA	A	En trenger å ta mindre hensyn til måten å skrive kode på ved FPGA design	
	B	Det er begrenset hvor mange adskilte klokkerlinjer (klokkedomener) en kan ha i en FPGA i forhold til i en ASIC	
	C	FPGA er ikke egnet til asynkron logikk	
	D	En kan flytte design både fra FPGA til ASIC og motsatt	

Oppgave 15

I denne oppgaven skal det konstrueres en tilstandsmaskin som styrer en kaffeautomat. En kopp koster kr. 10,- og det serveres kun svart kaffe. Automaten tar kun 5 og 10 kronersmynter. Automaten innholder en sensor som finner ut at *en* mynt er puttet på (`Coin`) og hvilke myntsort dette er. Alle signalene fra sensoren er aktivt høye i en klokkeperiode. I klokkeperioden som følger like etter at `Coin` har vært aktiv, indikerer signalene `Five` og `Ten` hvilken myntsort som er lagt på. Puttes det på for mye penger eller feil myntsort, gis alle pålagte penger i retur ved at signalet `CoinBack` går aktivt i en klokkeperiode. Når korrekt sum er lagt på, serveres kaffen ved at signalet `ServeCoffee` går aktivt i en klokkeperiode. Spesifiser eventuelt egne forutsetninger du gjør utover oppgaveteksten.

I tilstandsmaskinen skal du benytte følgende entitet:

```
Entity CoffeeMachine is
  Port
  (
    CLK         : in std_logic;   --Klokke
    RESET       : in std_logic;   --Asynkron reset
    Coin        : in std_logic;   --Mynt er puttet på
    Five        : in std_logic;   --Fem kroner
    Ten         : in Std_logic;   --Ti kroner
    CoinBack    : out std_logic;  --Gir tilbake alle penger
    ServeCoffee: out std_logic    --Serverer kaffe
  );
end CoffeeMachine;
```

a) Vekt 20%
Lag et ASM flytdiagram som beskriver tilstandsmaskinen.

b) Vekt 25%
Implementer tilstandsmaskinen fra a) ved bruk av VHDL.

c) Vekt 10%
Hvordan vil du i hovedtrekk gå fram for å simulere tilstandsmaskinen i b). Det er ikke noe krav om å skrive VHDL-kode.

c) Vekt 5%
Er implementasjonen din i b) en Mealy eller Moore maskin? Begrunn svaret.

INF3430/INF4430 Oppgavesvar for kandidat nr:_____

Oppgave	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430/INF4430 Digital systemkonstruksjon**
Eksamensdag:	**6. desember 2006**
Tid for eksamen:	**9 - 12**
Oppgavesettet er på 7 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Ingen**

Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1 – 14 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 15 som besvares på vanlige ark. Oppgave 1 - 14 har til sammen vekt på 40%, mens oppgave 15 har vekt på 60%.

Generelt for oppgave 1- 14:
Hver oppgave består av et tema i venstre kolonne og en del utsagn hver angitt med en stor bokstav. Oppgavene besvares ved å merke tydelige kryss (X) i rett kolonne for riktig svaralternativ (dvs. at et utsagn er sant) i skjemaet i vedlegget. Det er alltid *minst en* riktig avmerking for hver oppgave, men det er ofte *flere* riktige avmerkninger. *For å få best karakter skal man sette flere kryss i en oppgave hvis det er flere riktige utsagn.* Det gis 2 poeng for hver avkrysning der det skal være avkrysning. Det gis -1 poeng for hver avkrysning der det ikke skal være avkrysning. Mangel på kryss der det skal være kryss gir også -1 poeng. Du kan benytte høyre kolonne i oppgaveteksten til kladd. Skjema påført ditt kandidatnummer i vedlegget er din besvarelse.

Oppgave 1

Kretsteknologier	A	FPGA har normalt mindre logikk enn en CPLD	
	B	FPGA er mer praktisk å programmere enn (S)PLD	
	C	Celler i CPLD har mye til felles med PAL	
	D	CPLD har eksistert lenger enn SPLD	

Oppgave 2

Lagringsteknologi	A	Antifuse-teknologien baserer seg på å opprette forbindelser når en krets programmeres	
	B	EPROM er basert på å lagre ladning på en *floating gate* i en transistor	
	C	SRAM er velegnet til permanent lagring	
	D	Flash teknologien er en videreutvikling av (E)EPROM	
	E	Reprogrammeringstiden for SRAM er lenger enn for Flash/EPROM	

Oppgave 3

Programmerings-teknologier for programmerbar logikk	A	En krets basert på Flash krever ekstern rekonfigureringsfil ved oppstart	
	B	Antifuse bruker lite effekt (i et system i drift)	
	C	FPGA med antifuse-teknologi egner seg godt til prototyping	
	D	En krets basert på Flash er umiddelbart aktiv etter strømtilkobling	
	E	SRAM-teknologi er vel så motstandsdyktig mot stråling som antifuse	

Oppgave 4

Størrelse på FPGA logikkblokker	A	En finkornet (fine grained) FPGA-blokk kan kun realisere enkle funksjoner	
	B	Fordelen med finkornede blokker er at rutingressursene som kreves blir begrenset	
	C	Brukeren konfigurerer en gitt FPGA til å enten være grovkornet eller finkornet	
	D	Utfordringene med grovkornede (coarse grained) blokker er å utnytte dem fullt ut	

Oppgave 5

Klokkestyring	A	Klokkesignal brukes normalt *ikke* i et synkront design med flip-floper	
	B	Klokketre skal begrense at klokkeflanker ankommer til forskjellig tid rundt i en krets	
	C	”Clock managers” kan generere klokker med forskjellig frekvens	
	D	En ulempe med ”Clock managers” er at problemet med jitter øker	

Oppgave 6

En 3-input Xilinx LUT (look-up table) med innhold FE (hex) realiserer en	A	AND funksjon	
	B	OR funksjon	
	C	NAND funksjon	
	D	NOR funksjon	

Oppgave 7

(A)synkront design	A	I et synkront design klokkes normalt alle flip-floper med samme klokkesignal	
	B	Problemet med asynkron logikk er at spesifikasjon av timing blir vanskelig og uforutsigbar	
	C	Innføring av ekstra flip-floper for synkronisering bør unngås i design	
	D	Det er ingen ulemper med å kombinere synkron og asynkron styring (set/reset) av en flip-flop med hensyn på forbruk av ressurser i en FPGA	

Oppgave 8

Verifikasjon	A	Hendelsebasert (event driven) simulering er normalt uten timinginformasjon	
	B	I statisk timinganalyse modelleres normalt alle porter med lik tidsforsinkelse	
	C	Formell verifikasjon kan finne andre feil enn de som finnes ved simulering	
	D	Design beskrevet i høynivåspråk gir raskere simulering enn for tilsvarende beskrivelse i lavnivåspråk	

Oppgave 9

Syntese	A	Mengden logikk og forbindelseslinjer mellom flip-floper i et design påvirker hva som blir maksimal klokkefrekvens	
	B	Endring av hvilke flip-floper som benyttes i en FPGA kan påvirke maksimal klokkehastighet til et design	
	C	Den viktigste grunnen til å justere på plassering av et design i en FPGA er å minske størrelsen på designet	

Oppgave 10

Myke og harde prosessorkjerner	A	Samme type prosessor finnes både som myk og hard kjerne til Xilinx FPGA-er	
	B	En myk kjerne er raskere (høyere klokkefrekvens) enn en hard kjerne	
	C	En myk kjerne er ikke så plasseffektiv som en hard kjerne	
	D	EDK kan benyttes til design med prosessorkjerner	
	E	MicroBlaze er eksempel på en hard kjerne	

Oppgave 11

Virtuelle komponenter/ Intellectual Properties	A	Intellectual Properties (IP) er betegnelsen på ferdigutviklede blokker	
	B	IP-er er tilgjengelig både fra FPGA produsenter og andre leverandører	
	C	On-chip Peripheral Bus (OPB) er anvendelig for tilkobling av IP-er i Xilinx FPGA-er	
	D	En IP basert på kildekode gir normalt ikke så effektiv implementering som en forhåndsrutet IP	

Oppgave 12

Designspråk	A	VHDL anses for å være et lavnivå designspråk	
	B	Ordinære C/C++ språk egner seg godt til å uttrykke parallelle realiseringer i maskinvare	
	C	SystemC er basert på C/C++	
	D	SystemVerilog er basert på VHDL	

Oppgave 13

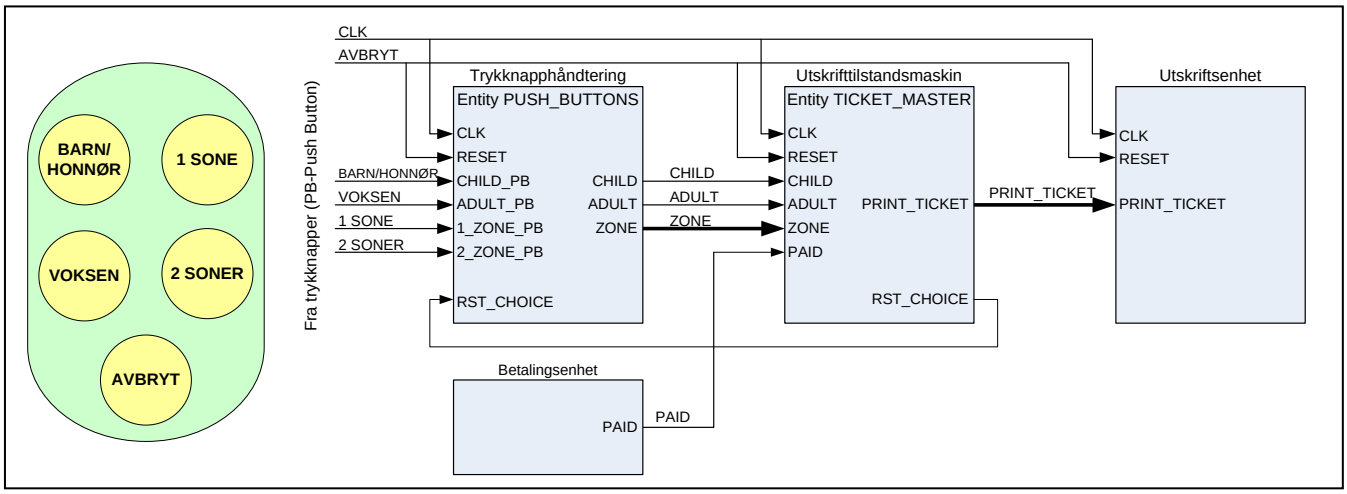
SystemC	A	Språket muliggjør design på forskjellig abstraksjonsnivå	
	B	Språket er lite egnet til verifisering	
	C	Syntese basert på SystemC kan gjøres direkte eller ved først å konvertere til VHDL	
	D	Språket egner seg godt til HW/SW codesign	

Oppgave 14

Høyhastighets serielinker	A	En av ulempene med seriekommunikasjon er at brukerprogrammet må sende/motta ett og ett bit	
	B	Et differensielt ledningspar er mindre følsomt for støy fra eksterne kilder enn en enkeltleder	
	C	Grunnen til at en overført firkantpuls ved høy datarate kan bli lik et sinussignal er at lavfrekvent frekvensinnhold er blitt filtrert bort	
	D	Et innkommende signal samples i senter av hvert bit	

Oppgave 15

Vi skal i denne oppgaven designe et system som skal kontrollere utskriften av billetter for Oslo Sporveier. Blokkskjema for systemet med betjeningspanel ser ut som følger:



Brukeren velger BARN/HONNØR *eller* VOKSEN, og 1 SONE *eller* 2 SONER (rekkefølgen er vilkårlig). Entiteten til PUSH_BUTTONS er som følger:

```
entity PUSH_BUTTONS is
port
(
   CLK             : in std_logic; -- Klokke
   RESET           : in std_logic; -- Asynkron reset
   CHILD_PB        : in std_logic; -- Velger barnebillett
   ADULT_PB        : in std_logic; -- Velger voksenbillett
   1_ZONE_PB       : in std_logic; -- Velger sone 1
   2_ZONE_PB       : in std_logic; -- Velger sone 2
   RST_CHOICE      : in std_logic; -- Reset fra tilstandsmaskin
   CHILD           : out std_logic; -- Barnebillett
   ADULT           : out std_logic; -- Voksenbillett
   ZONE            : out std_logic_vector(1 downto 0); -- Antall soner
);
end entity PUSH_BUTTONS;
```

Valg av sone skal enkodes i vektoren ZONE. ZONE skal lagres som en vektor i flip-floper og er tilkoblet de to sonetrykknappene ”1 SONE” og ”2 SONER” etter sannhetstabell 1 nedenfor. Signalet RESET er koblet til trykknappen ”AVBRYT” som er en angreknapp for brukeren. RESET skal kobles til *asynkron* reset på flip-flopene. Videre er RST_CHOICE en *synkron* reset som genereres av tilstandsmaskinen. I oppgaven er alle enkeltsignaler aktivt høye.

Sannhetstabell 1.

CLK	RESET	RST_CHOICE	1_ZONE_PB (1 SONE)	2_ZONE_PB (2 SONER)	ZONE
-[1]	1	-	-	-	00
↑	0	1	-	-	00
↑	0	0	1	0	01
↑	0	0	0	1	10
↑	0	0	1	1	00

1 ”-” er don’t care

Signalene CHILD og ADULT kan antas allerede lagret i flip-floper og disse er avledet fra trykknappene ”BARN/HONNØR” og ”VOKSEN” på billettautomaten på en slik måte at de ikke kan være aktive samtidig. Spesifiser eventuelt egne forutsetninger du gjør utover oppgaveteksten.

a) Vekt 10%

Implementer sannhetstabell 1 ved å benytte en prosess i VHDL.

Tilstandsmaskinen TICKET_MASTER skal ha følgende entitet:

```
entity TICKET_MASTER is
port
(
   CLK             : in std_logic; -- Klokke
   RESET           : in std_logic; -- Asynkron reset
   CHILD           : in std_logic; -- Velger barnebillett
   ADULT           : in std_logic; -- Velger voksenbillett
   ZONE            : in std_logic_vector(1 downto 0);  -- Antall soner
   PAID            : in std_logic; -- Betaling i orden, start utskrift
   PRINT_TICKET    : out std_logic_vector(2 downto 0); -- Printkommando
   RST_CHOICE      : out std_logic -- Nullstiller alle valg etter at
                                   -- utskrift er startet
);
end entity TICKET_MASTER;
```

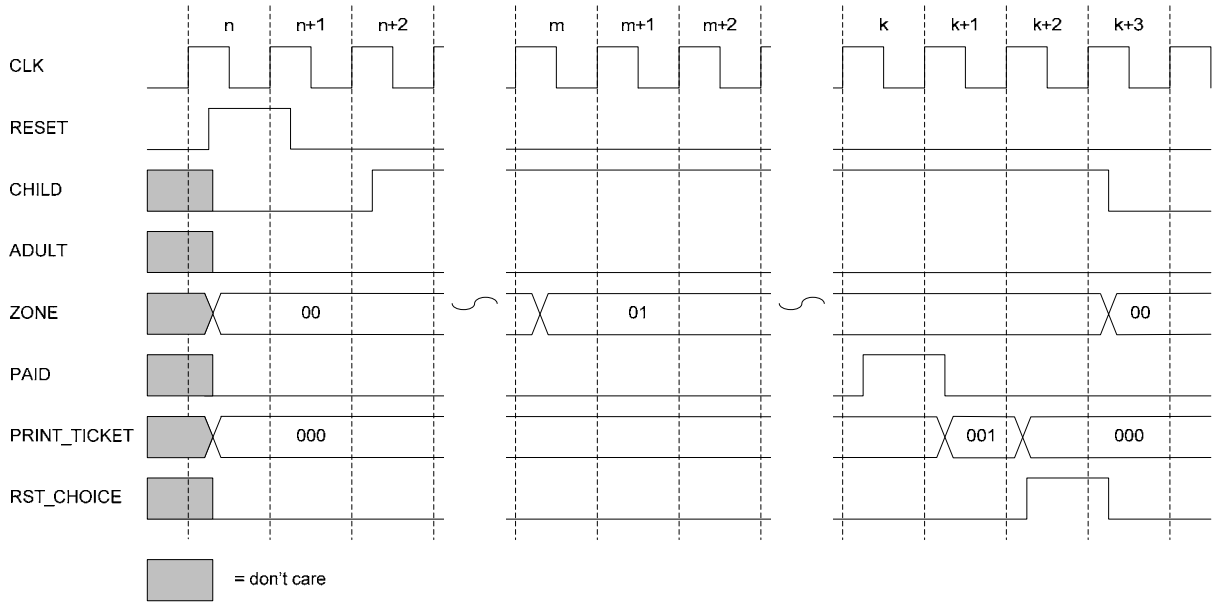
Man er nødt til å velge ”BARN/VOKSEN” *og* antall soner *før* man betaler. Etter at gyldig betaling er mottatt går signalet PAID aktivt i *en* klokkeperiode. (Automaten veksler, men dette er styrt av en annen tilstandsmaskin (Betalingsenhet). Samme tilstandsmaskin gir også pengene i retur dersom valg ikke er foretatt på automaten før penger puttes på.)

PRINT_TICKET signalet er en 3-bits vektor kodet etter sannhetstabell 2 nedenfor. PRINT_TICKET skal være aktiv (ulik ”000”) *en* klokkeperiode og følger *etter* at PAID har vært aktiv. PRINT_TICKET brukes som et startsignal for billettutskrift. Signalet RST_CHOICE er aktivt (”1”) i klokkeperioden *etter* PRINT_TICKET signalet har vært aktiv og brukes for å nullstille CHILD, ADULT og ZONE (som er lagret i flip-floper).

Sannhetstabell 2.

Inputs			**Outputs**
ADULT	**CHILD**	**ZONE**	**PRINT_TICKET**
0	0	0	000
0	1	01	001
0	1	10	010
1	0	01	101
1	0	10	110

Timingdiagrammet nedenfor illustrerer virkemåten til tilstandsmaskinen (basert på spesifikasjonen over):



Figur 1. Eksempel på timingdiagram over tilstandsmaskinen

b) Vekt 20%

Tegn et ASM diagram for å beskrive tilstandsmaskinen TICKET_MASTER.

c) Vekt 20%

Implementer tilstandsmaskinen du beskrev i b) ved bruk av VHDL.

d) Vekt 10%

Skisser blokkskjema (struktur) for hvordan tilstandsmaskinen bør testes.

Vedlegg 1

INF3430/INF4430 Oppgavesvar for kandidat nr:_____

Oppgave	**A**	**B**	**C**	**D**	**E**
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430/INF4430 Digital systemkonstruksjon**
Eksamensdag:	**6. desember 2007**
Tid for eksamen:	**9 - 12**
Oppgavesettet er på 8 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Ingen**

Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1 – 14 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 15 som besvares på vanlige ark. Oppgave 1 - 14 har til sammen vekt på 40%, mens oppgave 15 har vekt på 60%.

Generelt for oppgave 1- 14:
Hver oppgave består av et tema i venstre kolonne og en del utsagn hver angitt med en stor bokstav. Oppgavene besvares ved å merke tydelige kryss (X) i rett kolonne for riktig svaralternativ (dvs. at et utsagn er sant) i skjemaet i vedlegget. Det er alltid *minst en* riktig avmerking for hver oppgave, men det er ofte *flere* riktige avmerkninger. *For å få best karakter skal man sette flere kryss i en oppgave hvis det er flere riktige utsagn.* Det gis 1 poeng for hver avkrysning der det skal være avkrysning. Det gis -1 poeng for hver avkrysning der det ikke skal være avkrysning. Mangel på kryss der det skal være kryss gir også -1 poeng. Du kan benytte høyre kolonne i oppgaveteksten til kladd. Skjema påført ditt kandidatnummer i vedlegget er din besvarelse.

Oppgave 1

Kretsteknologier	A	En logikkblokk i en FPGA består normalt av en Look-Up Table (LUT) etterfulgt av en vippe (flip-flop)	
	B	En PLA består av OR-porter etterfulgt av AND-porter	
	C	I en PAL er tilkoblingene til AND-portene ikke programmerbare	
	D	I en ”full custom” ASIC har designeren full kontroll over hvert maskelag i kretsen	

Oppgave 2

Lagringsteknologi	A	I en CPLD lagres normalt konfigurasjonen i SRAM	
	B	En FPGA basert på antifuse-teknologi er ikke reprogrammerbar	
	C	En FPGA basert på antifuse-teknologi kan ikke slettes med UV-lys	
	D	En EPROM kan slette sitt innhold med en høy spenning	
	E	En SRAM kan kun programmeres et begrenset antall ganger	

Oppgave 3

Konfigurasjon av FPGA	A	En FPGA i master-modus styrer selv nedlastning av konfigurasjonen ved oppstart	
	B	”Daisy-chaining” gjør at flere FPGA-er kan ha et felles konfigurasjonsminne	
	C	En FPGA må alltid konfigureres parallelt hvis den er i slave-modus	
	D	JTAG-porten er egentlig tiltenkt testing men kan også brukes til konfigurasjon	

Oppgave 4

Optimalisert FPGA design	A	Selv om antall input til en funksjon er konstant, øker forbruket av logikk med kompleksiteten til funksjonen	
	B	Antall nivåer med logikk i en FPGA *mellom* klokkede vipper har betydning for maksimal klokkefrekvensen	
	C	Klokketre i en FPGA bør unngåes hvis en skal lage et effektivt synkront design	
	D	Dedikert mentelogikk kobler sammen logikk for hurtig menteforplantning	
	E	Bruk av dedikert mentelogikk gjør at det blir mindre tilgjengelig logikk i FPGA-en og bruken bør derfor begrenses	

Oppgave 5

En 3-input Xilinx LUT (look-up table) med innhold 7F (hex) realiserer en	A	AND funksjon	
	B	OR funksjon	
	C	NAND funksjon	
	D	NOR funksjon	

Oppgave 6

Prosessorkjerner	A	En hard kjerne er implementert fysisk i FPGA-en ved produksjon av kretsen	
	B	Kombinasjon av prosessor og logikk på en FPGA gir liten fleksibilitet i bestemmelsen av hva som blir programvare og hva som blir maskinvare	
	C	Separat buss mellom prosessor og minne gir lite gevinst og bør unngåes	
	D	Integrering av et helt system på en krets gir en mer kompakt løsning som også prismessig kan være gunstig	

Oppgave 7

Virtuelle komponenter/ Intellectual Property	A	En IP gitt som ikke-kryptert kildekode er normalt mer effektiv enn en IP gitt som forhåndsrutet IP	
	B	Intellectual Property er betegnelsen på ferdigutviklede blokker	
	C	MicroBlaze er eksempel på en IP	
	D	Det er enkelt å gjenbruke en IP fra en FPGA-produsent på kretser fra andre produsenter	

Oppgave 8

Sykelbasert simulering	A	Dette er et alternativ til hendelsesbasert simulering	
	B	En dropper å simulere hver hendelse i en krets men benytter boolske uttrykk på inngangene til registre	
	C	Metoden kan kombineres med hendelsesdrevet simulering for simulering av en krets	
	D	En ulempe, sammenlignet med alternative måter å simulere på, er at tiden for simulering øker betydelig	

Oppgave 9

Syntese	A	Syntese gjøres normalt etter ”place-and-route”	
	B	Syntese med informasjon om faktiske tidsforsinkelser i FPGA-en kan gi høyere maksimal klokkefrekvens	
	C	Plassering av registre (vipper) i forhold til logikk har normalt ingen betydning for ytelsen	
	D	Resyntese for optimalisering av kritisk signalvei kan være gunstig	

Oppgave 10

SystemC	A	Språket er definert av en spesifikk verktøyleverandør som selger designverktøy	
	B	Språket er basert på C/C++	
	C	Språket er bedre egnet til verifikasjon enn syntese	
	D	Språket kan spesifisere kode på flere abstraksjonsnivåer enn VHDL	
	E	SystemC brukes i dag like ofte som VHDL for FPGA design	

Oppgave 11

Kodestil for FPGA og ASIC	A	Samlebåndsprosessering (pipelining) kan være med på å øke maksimal klokkefrekvens i et design	
	B	Samlebåndsprosessering (pipelining) vil ofte medføre at en bruker færre vipper i et design	
	C	Tilbakekoblingssløyfer der vipper inngår må ikke brukes i en FPGA	
	D	Asynkront design er mulig i en ASIC, men anbefales ikke i en FPGA	

Oppgave 12

Valg mellom ASIC og FPGA	A	FPGA er bedre enn ASIC ved komplekse design	
	B	Det er bedre plass i en ASIC enn i en FPGA når kretsene har omtrent samme fysiske størrelse	
	C	Prototyping av ASIC på FPGA bør unngåes på grunn av forskjell i kodestil	
	D	ASIC har lang utviklingstid men de første kretsene er billige å produsere	

Oppgave 13

Høyhastighets serielinker	A	Grunnen til at en overført firkantpuls ved høy datarate kan bli lik et sinussignal er at høyfrekvent frekvensinnhold har blitt kraftig dempet	
	B	Konfigurasjon av parametere i transceiver muliggjør design med forskjellige kommunikasjonsstandarder	
	C	Pre-emphasis motvirker dempning i overført signal	
	D	”Comma”-tegn brukes for å dele opp lange bitstrenger	

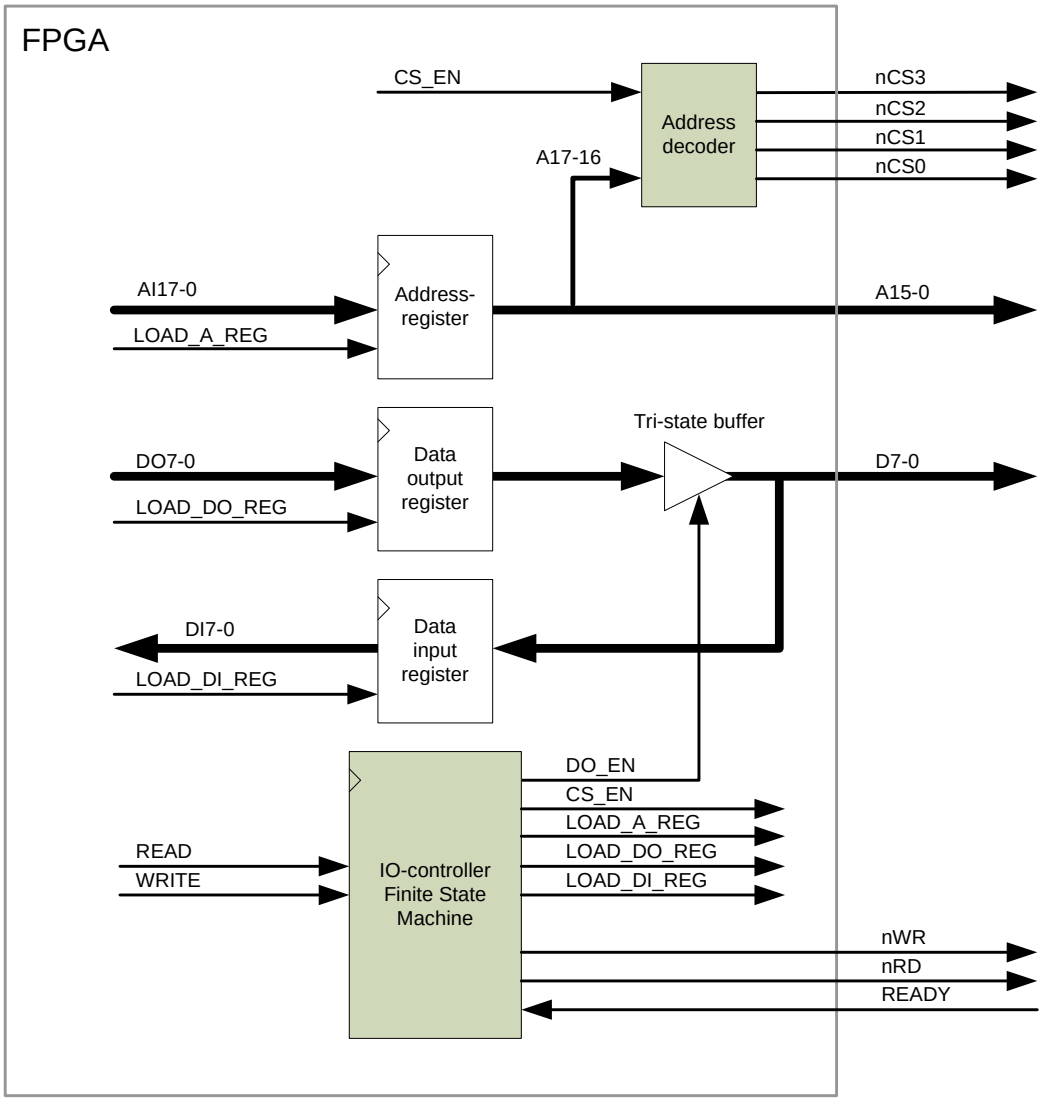
Oppgave 14

Rekonfigurering av aktiv FPGA	A	Virtuell maskinvare er en betegnelse som brukes om denne teknikken	
	B	Teknikken muliggjør å kunne utføre en større oppgave enn det kretsen tilsynelatende har logikk til	
	C	Effektforbruket kan ofte øke ved denne metoden	
	D	Lang rekonfigureringstid er en av hovedutfordringene	
	E	Det vil være ønskelig med denne metoden å rekon-figurere hele kretsen og ikke kun en begrenset del av den	

Oppgave 15

Vi skal i denne oppgaven konstruere deler av et grensesnitt som skal styre eksternt minne og/eller input/output kretser som er koblet til en FPGA. Vi vil referere til dette som I/O-grensesnittet. I/O-grensesnittet skal være del av en mikrokontroller i FPGAen.

Figur 1 viser en oversikt over I/O-grensesnittet. De gråskraverte boksene i figuren viser de delene av I/O-grensesnittet vi skal konsentrere oss om i de påfølgende oppgavene.



Figur 1. I/O-grensesnittet

Tabell 1. Signaler i I/O-grensesnittet

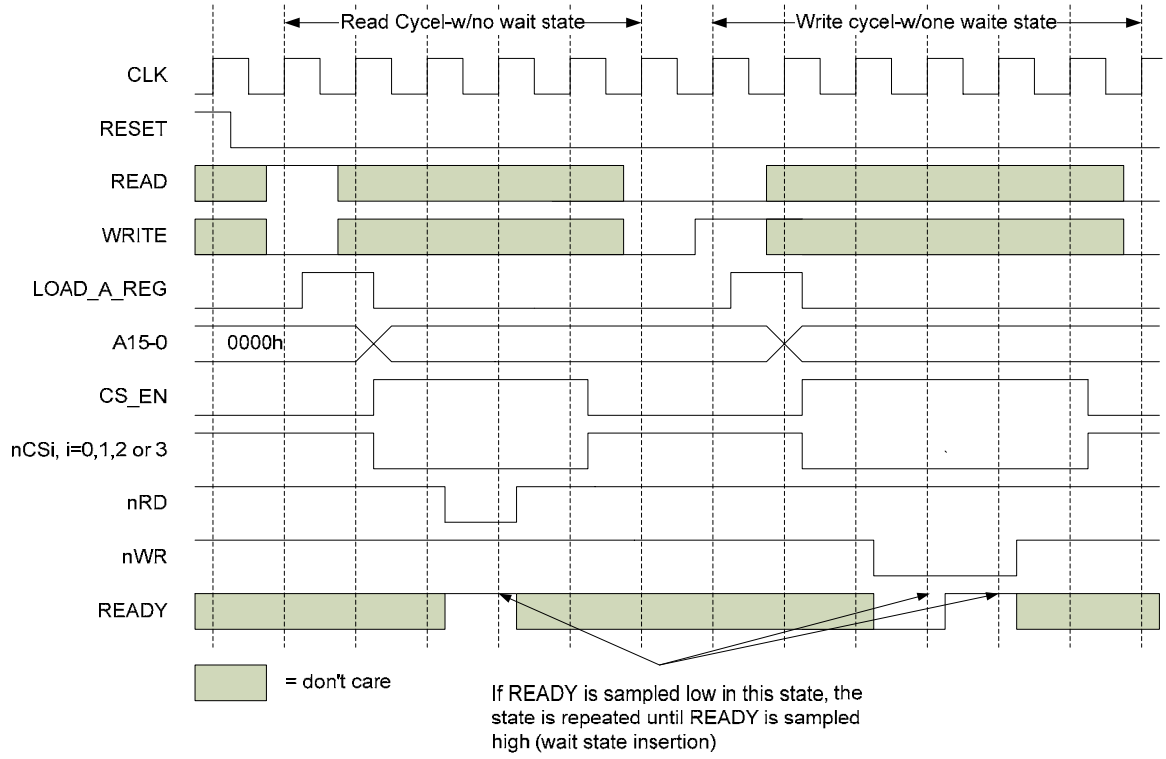
Signalnavn	**Beskrivelse**	**Retning**
Klokke og reset er ikke vist i figur 1		
CLK	50MHz systemklokke	Input til alle registre
RESET	Asynkron reset. Aktivt høyt	Input til alle registre
Eksterne signaler:		
A15-0	Adresse signaler	Output
D7-0	Data signaler	Input/Output/Tri-state
nCSi, i=0,1,2,3	Chip select signaler. Benyttes for å adressere eksternt minne eller I/O. Aktivt lave	Output fra adressedekoder
nWR	Write strobe. Aktivt lavt	Output fra tilstandsmaskin
nRD	Read strobe. Aktivt lavt.	Output fra tilstandsmaskin
READY	Viser om en I/O krets har data klare eller er klar til å ta i mot data. Aktivt høyt. Benyttes for å forlenge en les eller skriv I/O operasjon ved å sette inn ventetilstander (Wait states).	Input til tilstandsmaskin.
Interne signaler:		
READ	Starter en leseoperasjon fra I/O grensesnittet. Aktivt høyt	Input til tilstandsmaskin
WRITE	Starter en skriveoperasjon til I/O grensesnittet. Aktivt høyt	Input til tilstandsmaskin
AI17-0	Interne adressesignaler.	Input til adresseregisteret og adressedekoderen
DO7-0	Data output signaler	Input til data output registeret
DI7-0	Data input signaler	Output fra data input registeret
CS_EN	Enabler nCSi, i=0,1,2,3	Output fra tilstandsmaskinen
LOAD_A_REG	Lagrer adressene AI i adresseregistret. Aktivt høyt	Output fra tilstandsmaskinen
LOAD_DO_REG	Lagrer output data i data out registeret. Aktivt høyt	Output fra tilstandsmaskinen
DO_EN	Styrer output tri-state buffer	Output fra tilstandsmaskinen
LOAD_DI_REG	Lagrer input data i data input register. Aktivt høyt.	Output fra tilstandsmaskinen

Adressedekoderen "Address decoder" skal virke i henhold til sannhetstabellen under.

Sannhetstabell 1. Adressedekoderen

Inputs		Outputs			
CS_EN	A17-16	nCS0	nCS1	nCS2	nCS3
0	X	1	1	1	1
1	00	0	1	1	1
1	01	1	0	1	1
1	10	1	1	0	1
1	11	1	1	1	0

En les eller skriv I/O-operasjon er bygd opp av flere tilstander og starter med at READ- eller WRITE-signalet går aktivt. READ og WRITE kommer fra en tilstandmaskin som eksekverer programmer og er ikke aktive samtidig. Etter at READ/WRITE har vært aktiv skal de interne adressesignalene, AI17-0, lagres i adresseregisteret styrt av signalet LOAD_A_REG. A15-0 føres ut på pinner, mens A17-16 sammen med CS_EN er input til adressedekoderen som gir output i henhold til sannhetstabell 1. Resten av les eller skriv operasjonen skal følge timingdiagram 1 under. Legg merke til at en les eller skriv operasjon forlenges dersom READY-signalet er lavt når nRD eller nWR er aktivt. Benytt signalnavn som angitt over når du løser de etterfølgende oppgavene.



Timingdiagram 1

15a). Vekt 10%
Implementer sannhetstabell 1 ved å benytte en process i VHDL. Du trenger ikke å ta med entiteten.

Vi skal nå designe en tilstandsmaskin ”IO-controller” for å lage kontrollsignalene til I/O grensesnittet. Vi skal begrense oss til kontrollsignalene: LOAD_A_REG, CS_EN, nRD og nWR i I/O-grensesnittet.

15b). Vekt 20%
Tegn et ASM flytdiagram som beskriver tilstandsmaskinen gitt av tekst og timingdiagram over.

15c). Vekt 20%
Implementer tilstandsmaskinen beskrevet i ASM flytdiagrammet i 15b) i VHDL. Du trenger ikke å ta med entiteten.

15d). Vekt 10%
Skissèr en testbenk for å verifisere tilstandsmaskinen (du skal ikke lage en komplett testbenk).

Vedlegg 1

INF3430/INF4430 Oppgavesvar for kandidat nr:_____

Oppgave	**A**	**B**	**C**	**D**	**E**
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i: **INF3430/INF4430 Digital systemkonstruksjon**
Eksamensdag: **4. desember 2008**
Tid for eksamen: **9 - 12**
Oppgavesettet er på 9 sider
Vedlegg: **1**
Tillatte hjelpemidler: **Ingen**

Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1 – 14 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 15 som besvares på vanlige ark. Oppgave 1 - 14 har til sammen vekt på 40%, mens oppgave 15 har vekt på 60%.

Generelt for oppgave 1- 14:
Hver oppgave består at et tema i venstre kolonne og en del utsagn hver angitt med en stor bokstav. Oppgavene besvares ved å merke tydelige kryss (X) i rett kolonne for riktig svaralternativ (dvs. at et utsagn er sant) i skjemaet i vedlegget. Det er alltid *minst en* riktig avmerking for hver oppgave, men det er ofte *flere* riktige avmerkninger. *For å få best karakter skal man sette flere kryss i en oppgave hvis det er flere riktige utsagn.* Det gis 1 poeng for hver avkrysning der det skal være avkrysning. Det gis -1 poeng for hver avkrysning der det ikke skal være avkrysning. Mangel på kryss der det skal være kryss gir også -1 poeng. Du kan benytte høyre kolonne i oppgaveteksten til kladd. Skjema påført ditt kandidatnummer i vedlegget er din besvarelse.

Oppgave 1

Design med FPGA og ASIC	A	FPGA egner seg bedre enn ASIC i produkter med krav om lavt effektforbruk.	
	B	I en FPGA skal det alltid brukes register i en tilbakekoblingssløyfe.	
	C	Det er ingen begrensning av antall klokkedomener i en FPGA.	
	D	En ASIC krets kan inneholde analog og digital elektronikk i samme krets.	

Oppgave 2

Konfigurasjon av FPGA	A	JTAG porten er eneste metode for å få konfigurert en Xilinx FPGA.	
	B	Alle registere i en FPGA får en kjent verdi ved konfigurering.	
	C	En antifuse FPGA kan reprogrammeres 1 gang.	
	D	Konfigurasjonsfiler er alltid så små at det er raskt å bytte til en ny konfigurasjon.	

Oppgave 3

Verktøy og metodikk	A	Formell verifikasjon kan brukes for å sjekke at VHDL koden og ferdig nettliste er like.	
	B	Statisk timing analyse gjør at RTL simulering av designet i en VHDL testbenk er unødvendig.	
	C	En BFM (Bus Functional Model) kan erstatte prosessor bus interface i en testbenk	
	D	Retiming kan utføres under syntese	

Oppgave 4

FPGA design	A	Det er vanligvis kortere delay i wires mellom to LUT'er enn gjennom en LUT.	
	B	I en Xilinx FPGA har set inngangen til en flipflop/register lavere prioritet enn reset inngangen.	
	C	FPGA egner seg dårlig for pipelining pga. få registere.	
	D	En Xilinx Block RAM har to uavhengige porter som begge kan leses fra og skrives til.	

Oppgave 5

En 3-input Xilinx LUT med innhold 80 (hex) realiserer en:	A	AND funksjon	
	B	OR funksjon	
	C	NAND funksjon	
	D	NOR funksjon	

Oppgave 6

DCM og klokkenett for Xilinx	A	Klokkesignalene ut av Xilinx DCM modulen er ikke i fase med hverandre.	
	B	Klokkenett kan bare brukes for klokke signaler.	
	C	DCM kan forsinke en generert klokke slik at den er i fase med inngangsklokken.	
	D	En Xilinx BUFG modul brukes for hvert klokkenett.	

Oppgave 7

DSP konstruksjon	A	Xilinx har harde DSP moduler som har MAC (Multiply and Accumulate) funksjon.	
	B	Med verktøy fra Xilinx og Mathworks/Matlab kan det genereres FPGA moduler uten at konstruktøren trenger å skrive VHDL kode.	
	C	Flere DSP operasjoner kan gjøres i parallell i en DSP prosessor enn i en FPGA.	
	D	Det er vanlig at FPGA har harde DSP moduler for Analog-til-Digital og Digital-til-Analog konvertering.	

Oppgave 8

Virtuelle componenter og IP for Xilinx FPGA	A	Gigabit Transceivere finnes som myk kjerne.	
	B	ROM kan lages av harde Block RAM (BRAM) moduler	
	C	En myk prosessorkjerne har vanligvis lavere maksimal klokkehastighet enn en hard prosessorkjerne.	
	D	RAM kan lages av LUT'er.	

Oppgave 9

Timing constraints for Xilinx	A	FFS og PADS er eksempler på forhåndsdefinerte timing grupper.	
	B	From-To constraint har lavere prioritet enn Period constraint.	
	C	Timing krav spesifiseres i en User Constraint File (UCF).	
	D	Ved å "constraine" inngangsklokken til Xilinx DCM modulen vil alle utgangsklokker være timing "constrainet".	

Oppgave 10

En Xilinx FPGA kan inneholde harde kjerner for:	A	Multiplikasjon	
	B	MicroBlaze prosessor	
	C	MAC	
	D	Divisjon	

Oppgave 11

Gigabit Transceivere	A	En Transceiver modul har FIFO i senderetningen og FIFO i mottaksretningen.	
	B	Såkalte "comma" karakterer brukes i forbindelse med utjevning (equalization) i mottageren.	
	C	8B/10B signal koding brukes for å unngå flere enn 8 påfølgende like bit.	
	D	Differensielle signaler brukes for å redusere støy problemer.	

Oppgave 12

Kan variabler i VHDL deklareres i:	A	Process	
	B	Procedure	
	C	Architecture	
	D	Function	

Oppgave 13

Når vil `indata` satt lik ”11110000” komme ut på utgangen outdata i VHDL koden vist under med påtrykk (stimuli) som vist under?	A	350 ns	
	B	450 ns	
	C	550 ns	
	D	650 ns	

```
entity oppgave_delay is
  port (
    -- System Clock and Reset
    rst_n        : in  std_logic;
    mclk         : in  std_logic;
    indata       : in  std_logic_vector(7 downto 0);
    outdata      : out std_logic_vector(7 downto 0));
end oppgave_delay;

architecture rtl of oppgave_delay is
  signal data1, data3, data5 : std_logic_vector(7 downto 0);
begin
  process (rst_n, mclk) is
    variable data2, data4 : std_logic_vector(7 downto 0);
  begin
    if (rst_n = '0') then
      data1    <= (others => '0');
      data2    := (others => '0');
      data3    <= (others => '0');
      data4    := (others => '0');
      data5    <= (others => '0');
    elsif rising_edge(mclk) then
      data1    <= indata;
      data2    := data1;
      data3    <= data2;
      data4    := data3;
      data5    <= data4;
    end if;
  end process;

  outdata  <= data5;
end rtl;
```

Påtrykk(stimuli):

```
  P_CLK: process
  begin
    mclk <= '0';
    wait for 50 ns;
    mclk <= '1';
    wait for 50 ns;
  end process P_CLK;

  rst_n  <= '0', '1' after 100 ns;
  indata <= "00000000", "11110000" after 200 ns;
```

Oppgave 14

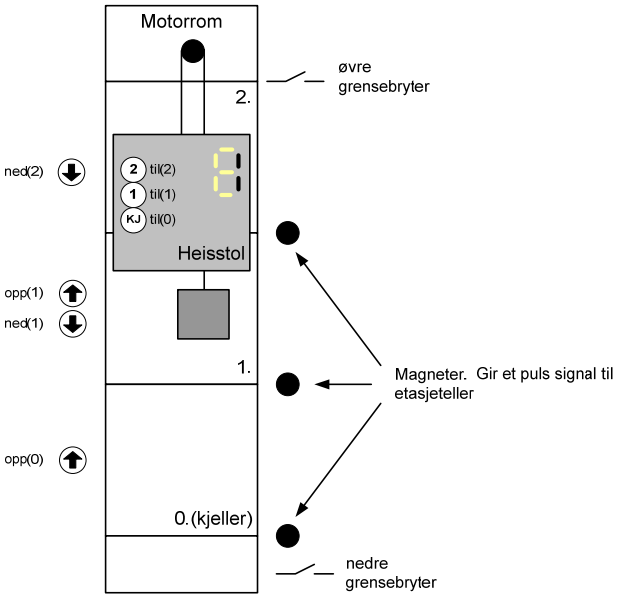
I VHDL koden vist under settes `indata` til '1' og vil gi outdata(7 downto 0) lik:	A	"10001011"	
	B	"01000111"	
	C	"10001011"	
	D	"10000111"	

```
entity oppgave_variabler_og_signaler is
  port (
    indata   : in  std_logic;
    outdata1 : out std_logic_vector(7 downto 0)
  );
end oppgave_variabler_og_signaler;

architecture rtl of oppgave_variabler_og_signaler is
  signal sig1, sig2 : std_logic;
begin

  process (indata, sig1, sig2) is
    variable var1, var2 : std_logic;
  begin
    var1:= indata;
    var2:= indata;
    sig1<= var1;
    sig2<= var2;
    outdata1(1 downto 0)<= var2 & var1;
    outdata1(3 downto 2)<= sig2 & sig1;
    var1:= not var1;
    var2:= not var2;
    sig1<= not var1;
    sig2<= not indata;
    outdata1(5 downto 4)<= var2 & var1;
    outdata1(7 downto 6)<= sig2 & sig1;
  end process;

end rtl
```

Oppgave 15 (Vekt 60%)

Vi skal i denne oppgaven lage en tilstandsmaskin som skal være en forenklet del av styringen av en treetasjers heis, fra 0.etasje (kjeller) til 2.etasje. I en virkelig heis vil det bl.a. være en rekke sikkerhetsfunksjoner som ikke vil bli berørt i oppgaven.
En heis består av forskjellige enheter. De to mest grunnleggende er:

- Heisstol - den bevegelige delen av heissystemet som frakter personer/ting
- Heissjakt - rommet hvor heisstolen beveger seg
- Motorrom - Som regel øverst i sjakten. Elektrisk motor med wire som er koblet til tak på heisstol og til motvekt. (Kan være hydraulisk system også)



Figur 1. Treetasjers heis

Heisen har utvendige trykknapper i hver etasje. Funksjonen til disse er å bestille heis til seg og samtidig fortelle hvilke retning man ønsker å ta heisen. I nederste og øverste etasje er det kun en trykknapp fordi det finnes bare en alternativ retning derfra.
Heisen har også innvendige trykknapper for å fortelle hvilke etasje man vil til. Vi skal avgrense heisstyringen til kun å ta hensyn til utvendige trykknappsignalene.
Når en av trykknappene blir trykket på blir dette lagret i flip-flop. Utgangen fra flip-flop'en er koblet til en lysdiode som gir et såkalt *kvitteringslys* som indikasjon på at knappen er trykket på. Lysdioden skal slukke når funksjonen til trykknappen som indikeres er ferdig utført.

Heisstyringen skal være det man i heisterminologien kaller *kollektiv*. Med *kollektiv* menes at alle trykknapper for en retning (opp eller ned) skal betjenes ferdig (kvitteringslys slukkes) før man snur og betjener trykknapper for motsatt retning.
F.eks. hvis heisstolen er på vei nedover og er mellom 0. og 1. etasje og det er trykket på `opp(1)`, `ned(1)`og `ned(2)` skal heisstolen stoppe i 1.etasje og slukke opp(1) LED, kjøre videre til 2.etasje og slukke ned(2) LED, snu og slukke ned(1) LED på vei nedover. Et annet eksempel er hvis det er trykket `opp(0)` og `opp(1)` og heisstolen er på vei nedover og befinner seg mellom 1. og 2. etasje skal den ikke stoppe for `opp(1)` før etter at den har vært i 0. etasje og er på vei oppover.

En teller, `et` (etasjeteller), holder rede på hvilke etasje heisstolen befinner seg i. `et` skifter verdi hver gang heisstolen er i bevegelse og samtidig med at den er på nivå med en etasje. Avhengig av hvilke knapper som er trykket, skal heisstolen stoppe i det øyeblikket etasjetelleren skifter verdi[1]. Telleren benytter bit 1 i `motor`- signalet (se tabell under) for å bestemme hvilken retning den skal telle.
Signalet `et_puls` gir en puls med varighet en periode av `clk` hver gang en etasje passeres. `et_puls` skifter verdi på samme tidspunkt som `et` eventuelt skifter verdi.

Ved oppstart, etter at `reset` har vært aktiv, så skal heisen kjøre ned til nedre grensebryter, `nedre`, og deretter opp til 0.etasje. `et` nullstilles på vei opp til 0.etasje og når heisen er i 0.etasje flagget ved en puls på `et_puls` uten at teller skifter verdi (fordi `et` nå er i reset) er heisen klar til bruk.

Vi antar at alle inngangssignaler er aktivt høye, prellfrie og synkrone med klokken `clk`.

Tabell 1. Oversikt over signaler i heisstyringen

Signalnavn	**Beskrivelse (Funksjon)**	**Retning**
`clk`	Systemklokke	Inngang til tilstandsmaskin
`reset`	Asynkron reset	Inngang til tilstandsmaskin
`et(1 downto 0)`	Etasjeteller (Teller opp/ned fra 0-2, 2-0)	Inngang til tilstandsmaskin
`et_puls`	Puls på en clk periode hver gang en magnet passeres. Synkron med transisjon på teller	Inngang til tilstandsmaskin
`ovre`	*Øvre grensebryter*	*Inngang* ***(skal ikke benyttes i oppgaven)***
`nedre`	Nedre grensebryter	Inngang til tilstandsmaskin
Signaler fra trykknapper inne i heisstolen		
`til(2 downto 0)`	*Bestilling av heis fra heisstol*	*Inngang* ***(skal ikke benyttes i oppgaven)***
Signaler fra trykknapper i de enkelte etasjer		
`opp(1 downto 0)`	Bestilling av heis oppover fra 0.-1.etasje	Inngang til tilstandsmaskin
`ned(2 downto 1)`	Bestilling av heis nedover fra 1.-2.etasje	Inngang til tilstandsmaskin
Signaler avledet av utvendige og innvendige trykknapper		
`over(1 downto 0)`	Øverste etasje det er trykket en knapp	Inngang til tilstandsmaskin
`under(1 downto 0)`	Nederste etasje det er trykket en knapp	Inngang til tilstandsmaskin
`trykket`	Viser at minst en trykknapp har blitt trykket	Inngang til tilstandsmaskinen
Utgangssignaler fra tilstandsmaskin		
`til_res(2 downto 0`	*Slukker LED i TIL(2 downto 0) knappene*	*Utgang fra tilstandsmaskin* ***(skal ikke benyttes i oppgaven)***
`opp_res(1 downto 0)`	Slukker LED i opp(1 downto 0) knappene	Utgang fra tilstandsmaskin
`ned_res(2 downto 1)`	Slukker LED i ned(2 downto 1) knappene	Utgang fra tilstandsmaskin
`et_res`	Reset etasjeteller	Utgang fra tilstandsmaskin
`motor(1 downto 0)`	00-Stopp på vei nedover (STOPP_NED) 01-Kjører nedover (START_NED) 10-Kjører oppover (START_OPP) 11-Stopp på vei oppover (STOPP_OPP)	Utgang fra tilstandsmaskin til motorstyring

[1] Dette fungerer ok for heiser som beveger seg langsomt, men for raskere heiser blir dette svært ubehagelig, les bråstopp

En vesentlig informasjon for å starte heisen den ene eller andre veien er posisjonen til heisstolen i forhold hvilke etasjer det er trykket på knapper. Ved å implementere funksjonene gitt av sannhetstabellene 1-2 har man det man trenger av informasjon for å kunne foreta nødvendige valg av retning heisen skal starte i. Signalet `trykket` er nødvendig i tillegg til `over` og `under` signalene fordi verdien til disse ikke er entydige med at det er trykket på en knapp.

Sannhetstabell 1. Sannhetstabell som viser øverste etasje der det er trykket en knapp, `over`, og signalet `trykket`. `trykket` viser at det er trykket på minst en knapp.

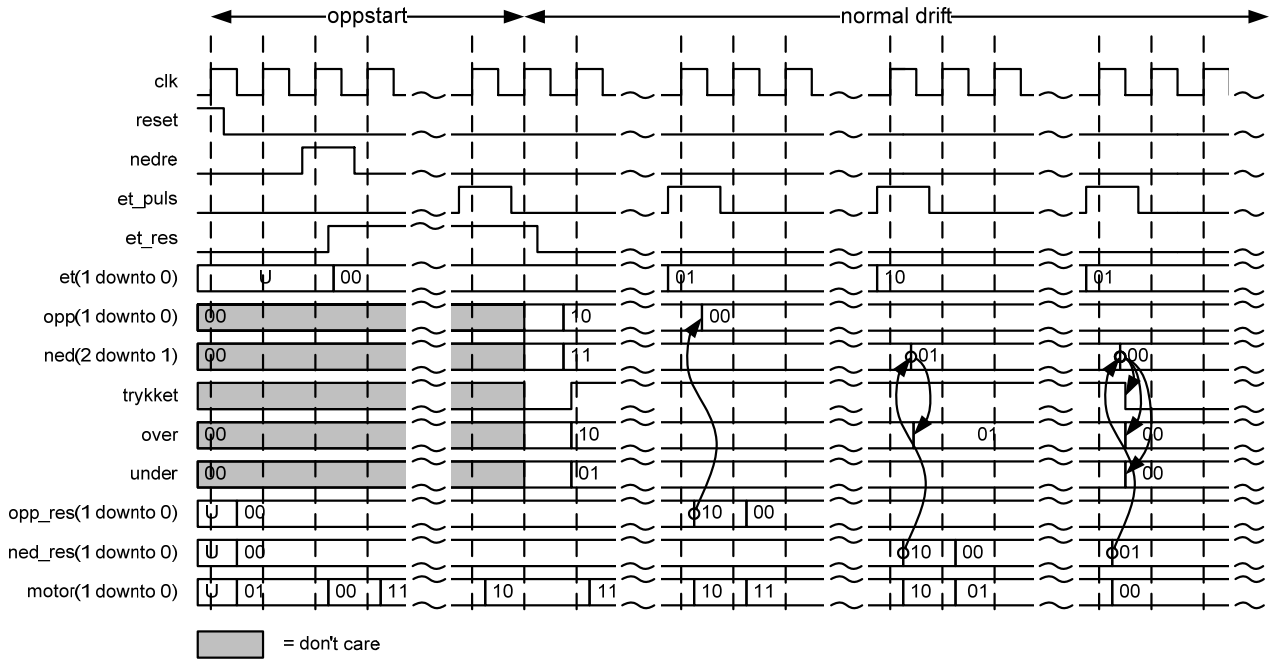
Innganger				Utganger	
ned(2)	**opp(1)**	**ned(1)**	**opp(0)**	**over**	**trykket**
0	0	0	0	00	0
0	0	0	1	00	1
0	X	1	X[2]	01	1
0	1	X	X	01	1
1	X	X	X	10	1

Sannhetstabell 2. Sannhetstabell som viser nederste etasje der det er trykket en knapp, `under`

Innganger				Utgang
ned(2)	**opp(1)**	**ned(1)**	**opp(0)**	**under**
0	0	0	0	00
X	X	X	1	00
X	X	1	0	01
X	1	X	0	01
1	0	0	0	10

a) Vekt 10%. Implementerer sannhetstabell 1 ved å benytte en "process" i VHDL.
Hint. Prioritetsenkoder.

[2] X er don't care, dvs. kan være 0 eller 1



Figur 2. Eksempel timing

b).Vekt 10%

Lag et ASM-tilstandsdiagram for å beskrive oppstartsforløpet til heisstyringen (etter at `reset` har vært aktiv). Se oppstartområdet i figur 1 over.

c).Vekt 15% .

Utvid ASM-tilstandsdiagrammet i b) til å beskrive tilstandsmaskinen for heisstyringen etter oppstartsforløpet. Se normal drift området i figur 1 over.

d) Vekt 15%.
Implementer tilstandsmaskinen beskrevet av ASM-flytdiagrammet fra b) i en 2-"process" tilstandsmaskin i VHDL (en "process" for nestetilstandslogikk og utgangslogikk, og en "process" for tilstandsregisteret). Dere trenger bare å implementere "architecture"-delen av tilstandsmaskinen.

e) Vekt 10%.
Tilstandsmaskinen beskrevet i timingdiagrammet over har den svakheten at den lager en stopp som bare varer en klokkeperiode.

Hvordan kan man lage en pause etter at heisen har stoppet og til den starter igjen? Lag en kort forklaring med ord.

INF3430/4430. Oppgavesvar for kandidat nr: _____

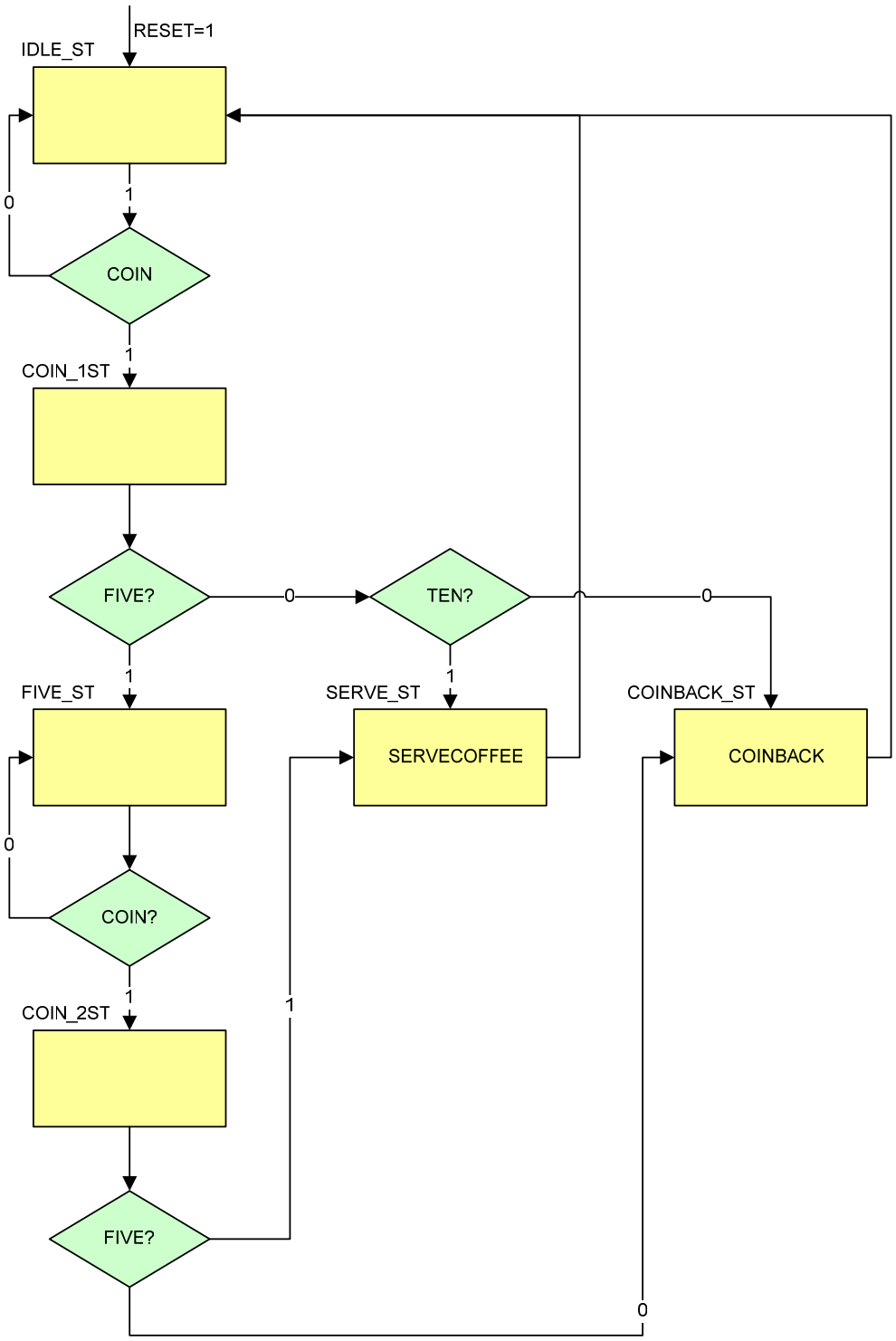
Oppgave	A	B	C	D
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				

INF3430/INF4430 Eksamensfasit H2005, Oppgave 1-14

Oppgave	A	B	C	D	E
1		X	X		X
2	X		X	X	X
3	X				
4	X		X		
5	X	X		X	
6	X				
7	X		X		
8	X		X		
9	X	X		X	
10	X		X	X	X
11	X		X		X
12		X	X	X	
13	X		X	X	
14		X	X	X	

Oppgave 15a).

ASM-flytskjema for kaffemaskinen:



```
1   library IEEE;
2   use IEEE.std_logic_1164.all;
3   use IEEE.std_logic_unsigned.all;
4
5   Entity COFFEEMACHINE is
6     Port
7     (
8       CLK         : in std_logic;   --Klokke
9       RESET       : in std_logic;   --Asynkron reset
10      COIN        : in std_logic;   --Mynt er puttet på
11      FIVE        : in std_logic;   --Fem kroner
12      TEN         : in Std_logic;   --Ti kroner
13      COINBACK    : out std_logic;  --Gir tilbake alle penger
14      SERVECOFFEE : out std_logic   --Serverer kaffe
15    );
16  end Entity COFFEEMACHINE;
17
18
19  architecture RTL_COFFEEMACHINE of Crchitecture RTL_COFFEEMACHINE of COF
    FEEMACHINE is
20
21  --Definere tilstander ved å benytte enumerert datatype
22  type COFFEEMACHINE_STATES is (IDLE_ST,COIN_1ST,FIVE_ST,COIN_2ST,SERVE_S
    T,COINBACK_ST);
23  signal CURRENT_ST, NEXT_ST : COFFEEMACHINE_STATES;
24
25  begin
26
27  --Tilstandsregister
28  STATE_REG:
29  process(RESET,CLK)
30  begin
31    if RESET = '1' then
32      CURRENT_ST <= IDLE_ST;
33    elsif rising_edge(CLK) then
34      CURRENT_ST <= NEXT_ST;
35    end if;
36  end process;
37
38  --Nestetilstandslogikk og utgangssignaler i
39  --samme kombinatoriske process
40  STATE_COMB:
41  process(COIN,FIVE,TEN,CURRENT_ST)
42  begin
43    COINBACK    <= '0';
44    SERVECOFFEE <= '0';
45    case CURRENT_ST is
46      when IDLE_ST =>
47        if COIN = '1' then
48          NEXT_ST <= COIN_1ST;
49        else
50          NEXT_ST <= IDLE_ST;
51        end if;
52      when COIN_1ST =>
53        if FIVE = '1' then
54          NEXT_ST <= FIVE_ST;
55        elsif TEN = '1' then
56          NEXT_ST <= SERVE_ST;
57        else
58          NEXT_ST <= COINBACK_ST;
59        end if;
```

```
60          when FIVE_ST =>
61            if COIN = '1' then
62              NEXT_ST <= COIN_2ST;
63            else
64              NEXT_ST <= FIVE_ST;
65            end if;
66          when COIN_2ST =>
67            if FIVE = '1' then
68              NEXT_ST <= SERVE_ST;
69            else
70              NEXT_ST <= COINBACK_ST;
71            end if;
72          when SERVE_ST =>
73            SERVECOFFEE <= '1';
74            NEXT_ST <= IDLE_ST;
75          when COINBACK_ST =>
76            COINBACK <= '1';
77            NEXT_ST <= IDLE_ST;
78        end case;
79      end process STATE_COMB;
80      end architecture RTL_COFFEEMACHINE;
81
```

Oppgave 15c).

For å simulere kretsen i oppgave 15b) må man påtrykke inngangene stimuli og sjekke utgangene.

I VHDL gjør man dette ved å lage en testbenk. I sin enkleste er den bygd opp på følgende måte

1. En (vanligvis) tom entitet for selve testbenken. Dvs. testbenken har vanligvis ikke noe interface mot verden utenfor men er ”selfcontained”.
2. Komponentdeklarasjon for UUT (Unit Under Test)
3. Deklarsjon av input stimuli signaler
4. Definisjon av klokke
5. Instantiering av UUT
6. Stimuli process der man lager en sekvens av input signaler
7. Sjekker output i ”Waveform-viewer”

I mer avanserte testbenker kan man istedenfor stimuliprosessen påtrykke inputstimuli ved å benytte simuleringsmodeller av omkringliggende kretser og instantiere disse i testbenken. Selve testbenken kan bli vesentlig enklere på denne måten.

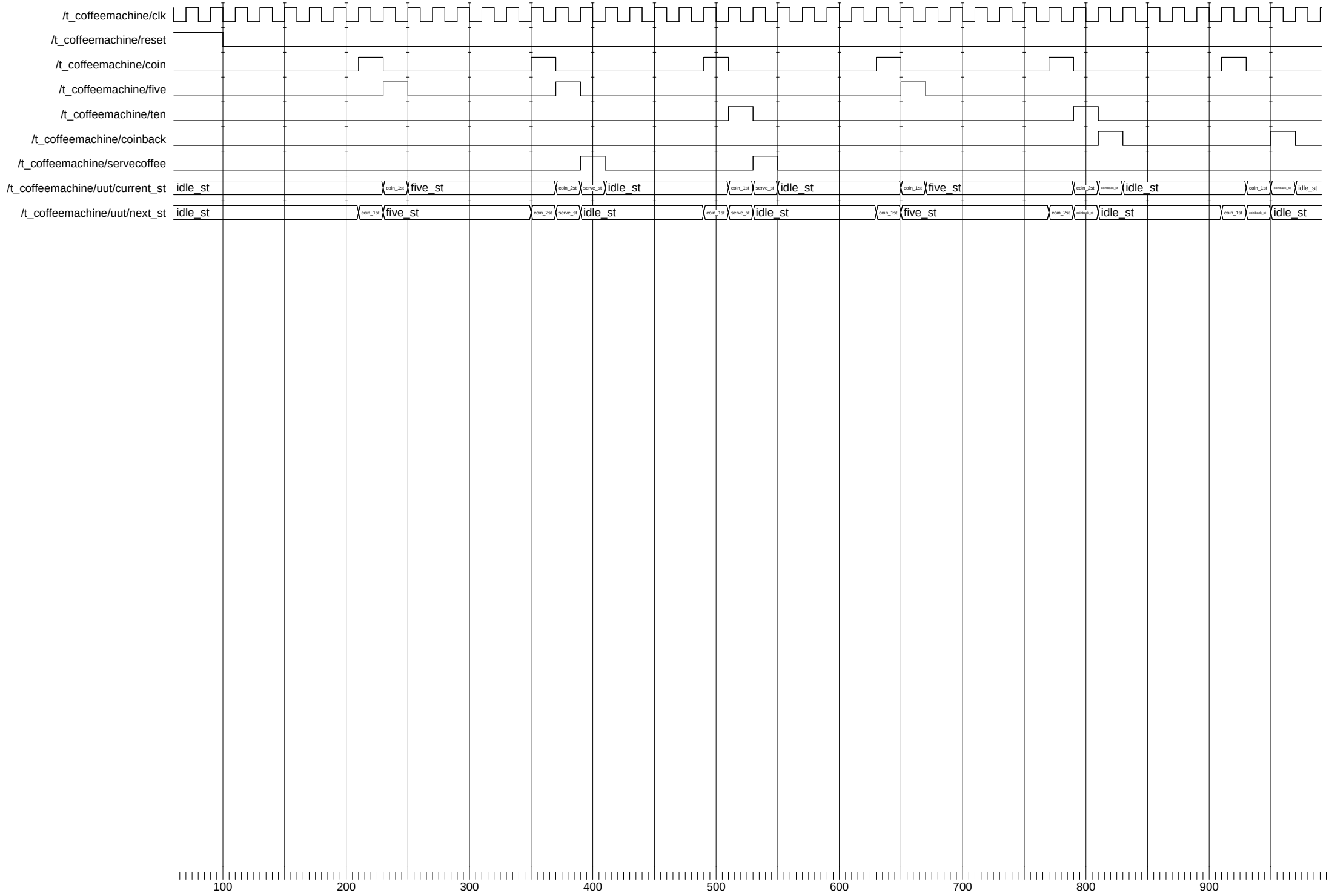
Et annet alternativ er å hente input stimuli fra fil.

En mer avansert måte å sjekke korrekt funksjon er å lage en fasit over forventede verdier på utgangene og sammenligne disse med utgangene av UUT. Fasiten kan man lagre i en egen fil eller inni testbenken.
På denne måten kan testbenken være selvtestende og man kan slippe å studere timingdiagrammer. Man kan bare rapportere om resultatet er OK eller ikke.

Eksemplet nedenfor er et eksempel på en testbenk av den enkleste varianten som inneholder punktene 1-6 over:

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

Entity T_COFFEEMACHINE is
end T_COFFEEMACHINE;

architecture TEST_COFFEEMACHINE of T_COFFEEMACHINE is

Component COFFEEMACHINE is
  Port
  (
    CLK         : in std_logic;   --Klokke
    RESET       : in std_logic;   --Asynkron reset
    COIN        : in std_logic;   --Mynt er puttet på
    FIVE        : in std_logic;   --Fem kroner
    TEN         : in Std_logic;   --Ti kroner
    COINBACK    : out std_logic;  --Gir tilbake alle penger
    SERVECOFFEE : out std_logic   --Serverer kaffe
  );
end component COFFEEMACHINE;

--Inngangssignaler
signal CLK          : std_logic := '0';
signal RESET        : std_logic := '0';
signal COIN         : std_logic := '0';
signal FIVE         : std_logic := '0';
signal TEN          : std_logic := '0';

--Utgangssignaler
signal COINBACK     : std_logic;
signal SERVECOFFEE  : std_logic;

constant CLK_Period : time:= 20 ns; --50MHz klokke

begin

--Genererer klokke
KLOKKE:
CLK <= not CLK after CLK_Period/2;

--Instantierer Unit Under Test
UUT: COFFEEMACHINE
port map
(
  CLK          => CLK,
  RESET        => RESET,
  COIN         => COIN,
  FIVE         => FIVE,
  TEN          => TEN,
  COINBACK     => COINBACK,
  SERVECOFFEE  => SERVECOFFEE
);

--Genererer input stimuli
STIMULI:
process
begin
  RESET <= '1','0' after 100 ns;
  wait for 10*CLK_Period;
  wait until rising_edge(CLK);
```

```
62        loop
63          --
64          COIN <= '1';
65          wait for CLK_Period;
66          COIN <= '0';
67          FIVE <= '1';
68          wait for CLK_Period;
69          FIVE <= '0';
70          wait for CLK_Period*5;
71          COIN <= '1';
72          wait for CLK_Period;
73          COIN <= '0';
74          FIVE <= '1';
75          wait for CLK_Period;
76          FIVE <= '0';
77          wait for CLK_Period*5;
78
79          --Ingen penger tilbake, kaffe serveres
80          COIN <= '1';
81          wait for CLK_Period;
82          COIN <= '0';
83          TEN <= '1';
84          wait for CLK_Period;
85          TEN <= '0';
86          wait for CLK_Period*5;
87          --Kaffe ventet
88
89          COIN <= '1';
90          wait for CLK_Period;
91          COIN <= '0';
92          FIVE <= '1';
93          wait for CLK_Period;
94          FIVE <= '0';
95          wait for CLK_Period*5;
96          COIN <= '1';
97          wait for CLK_Period;
98          COIN <= '0';
99          TEN <= '1';
100         wait for CLK_Period;
101         TEN <= '0';
102         wait for CLK_Period*5;
103         --Pengene tilbake, ingen kaffe
104
105         COIN <= '1';
106         wait for CLK_Period;
107         COIN <= '0';
108         wait for CLK_Period*5;
109         --Ingen gyldige penger, penger tilbake, ingen kaffe
110       end loop;
111     end process STIMULI;
112
113     end architecture TEST_COFFEEMACHINE;
114
```



Entity:t_coffeemachine Architecture:test_coffeemachine Date: Thu Dec 01 22:47:47 Romance Standard Time 2005 Row: 1 Page: 1

Oppgave 15d)

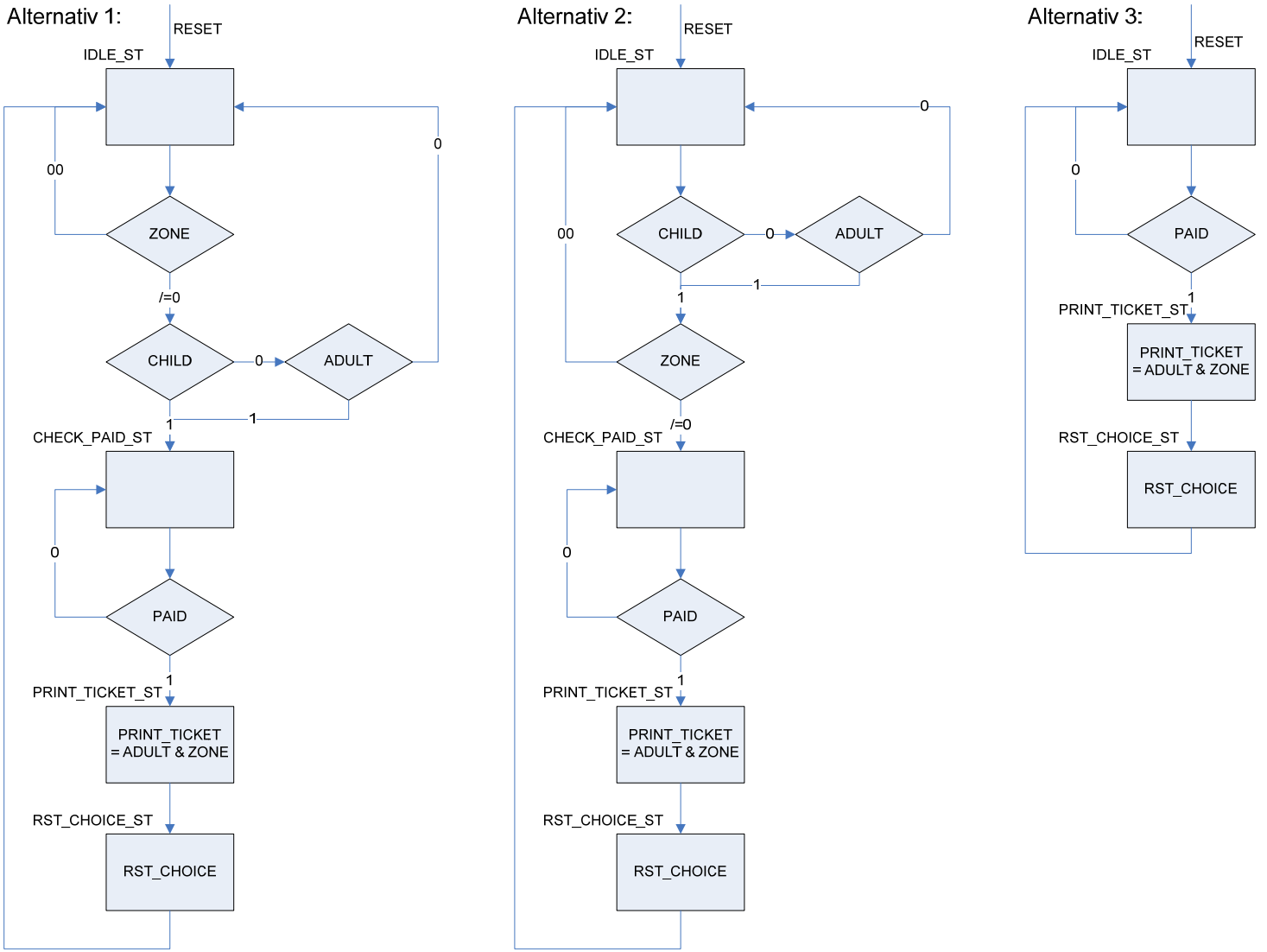
Tilstandsmaskinen implementert i oppgave 15b) er et eksempel på en Moore-maskin. I en Moore-maskin avhenger utgangene bare av nåværende tilstand i motsetning til en Mealy-maskin der utgangene avhenger av både nåværende tilstand og inngangene.

INF3430/INF4430 Fasit eksamen 2006 Oppgave 1-14

Oppgave	A	B	C	D	E
1		O	O		
2	O	O		O	
3		O		O	
4	O			O	
5		O	O		
6		O			
7	O	O			
8		O	O	O	
9	O	O			
10			O	O	
11	O	O	O	O	
12	O		O		
13	O		O	O	
14		O		O	

```
--Oppgave 15a:
--=============


ZONE_REG:
process(RESET,CLK)
begin
  if RESET = '1' then
    ZONE <= (others => '0');
  elsif rising_edge(CLK) then
    if RST_CHOICE = '1' then
      ZONE <= (others => '0');
    elsif 2_ZONE_PB = '0' and 1_ZONE_PB = '1' then
      ZONE <= "01";
    elsif 2_ZONE_PB = '1' and 1_ZONE_PB = '0' then
      ZONE <= "10";
    elsif 2_ZONE_PB = '1' and 1_ZONE_PB = '1' then
      ZONE <= (others => '0');
    end if;
  end if;
end process;

```

Oppgave 15b).

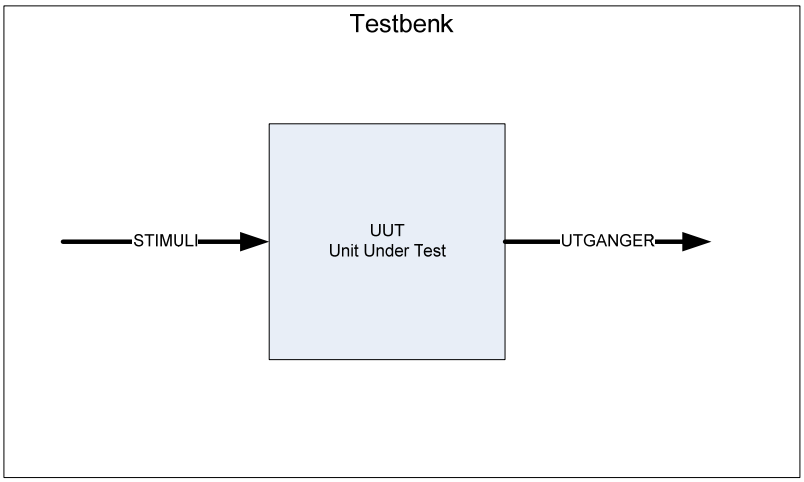

Alternativ 1:
RESET
IDLE_ST
00
ZONE
/=0
CHILD
0
ADULT
0
1
1
CHECK_PAID_ST
0
PAID
1
PRINT_TICKET_ST
PRINT_TICKET
= ADULT & ZONE
RST_CHOICE_ST
RST_CHOICE
Alternativ 2:
RESET
IDLE_ST
0
00
CHILD
0
ADULT
1
1
ZONE
/=0
CHECK_PAID_ST
0
PAID
1
PRINT_TICKET_ST
PRINT_TICKET
= ADULT & ZONE
RST_CHOICE_ST
RST_CHOICE
Alternativ 3:
RESET
IDLE_ST
0
PAID
1
PRINT_TICKET_ST
PRINT_TICKET
= ADULT & ZONE
RST_CHOICE_ST
RST_CHOICE

```
39      --Oppgave 15c:
40      --=============
41
42      --Alternativ 1 og 2
43      --==================
44      architecture RTL_TICKET_MASTER of TICKET_MASTER is
45
46      type TICKET_MASTER_STATE is (IDLE_ST,CHECK_PAID_ST,PRINT_TICKET_ST,RST_CHOICE_ST
        )
47      signal CURRENT_ST,NEXT_ST : TICKET_MASTER_STATE;
48
49      begin
50
51      --Next state and output logic
52      NEXT_STATE_COMB:
53      process (CHILD,ADULT,ZONE,PAID,CURRENT_ST)
54      begin
55
56        RST_CHOICE    <= '0';
57        PRINT_TICKET  <= (others => '0');
58        NEXT_ST       <= IDLE_ST;
59
60        case CURRENT_ST is
61
62          when IDLE_ST =>
63            --Alt 1:
64            --==============
65            --if ZONE /= "00" then
66            --  if (CHILD = '1' or ADULT = '1') then
67            --    NEXT_ST <= CHECK_PAID_ST;  --vi kan benytte if uten else fordi
68            --  end if;                      --vi benytter defaultverdier i starten
69            --end if;                       --av processen
70
71            --Alt 2:
72            --==============
73            if CHILD = '1' then
74              if ZONE /= "00" then
75                NEXT_ST <= CHECK_PAID_ST;
76              end if;
77            elsif ADULT = '1' then
78              if ZONE /= "00" then
79                NEXT_ST <= CHECK_PAID_ST;
80              end if;
81            end if;
82
83          when CHECK_PAID_ST =>
84            if PAID = '1' then
85              NEXT_ST <= PRINT_TICKET_ST;
86            else
87              NEXT_ST <= CHECK_PAID_ST;
88            end if;
89
90          when PRINT_TICKET_ST =>
91            NEXT_ST <= RST_CHOICE_ST;
92            PRINT_TICKET <= ADULT & ZONE;  --
93
94          when RST_CHOICE_ST =>
95            NEXT_ST <= IDLE_ST;
96            RST_CHOICE <= '1';
97
98        end case;
99
100     end process NEXT_STATE_COMB;
101
102     CURRENT_STATE_REG:
103     process (RESET,CLK)
104     begin
105       if RESET = '1' then
106         CURRENT_ST <= IDLE_ST;
107       elsif rising_edge (CLK) then
108         CURRENT_ST <= NEXT_ST;
```

```
109       end if;
110     end process CURRENT_STATE_REG;
111
112     end architecture RTL_TICKET_MASTER;
113
114
115     --Alternativ 3:
116     --==============
117     architecture RTL_TICKET_MASTER of TICKET_MASTER is
118
119     type TICKET_MASTER_STATE is (IDLE_ST,PRINT_TICKET_ST,RST_CHOICE_ST)
120     signal CURRENT_ST,NEXT_ST : TICKET_MASTER_STATE;
121
122     begin
123
124     --Next state and output logic
125     NEXT_STATE_COMB:
126     process (PAID,CURRENT_ST)
127     begin
128
129       RST_CHOICE    <= '0';
130       PRINT_TICKET  <= (others => '0');
131
132       case CURRENT_ST is
133
134         when IDLE_ST =>
135           if PAID = '1' then            --Antar at CHILD/ADULT og ZONE er aktivert
136             NEXT_ST <= PRINT_TICKET_ST; --før PAID kan gå aktivt
137           else                          --Mefører at disse ikke er nødvendige på
138             NEXT_ST <= IDLE_ST;         --sensitivitetslisten
139           end if;
140
141         when PRINT_TICKET_ST =>
142           NEXT_ST <= RST_CHOICE_ST;
143           PRINT_TICKET <= ADULT & ZONE; --Legg merke til enkel
144                                         --sammenheng i sannhetstabell 2
145         when RST_CHOICE_ST =>
146           NEXT_ST <= IDLE_ST;
147           RST_CHOICE <= '1';
148
149       end case;
150
151     end process NEXT_STATE_COMB;
152
153     CURRENT_STATE_REG:
154     process (RESET,CLK)
155     begin
156       if RESET = '1' then
157         CURRENT_ST <= IDLE_ST;
158       elsif rising_edge (CLK) then
159         CURRENT_ST <= NEXT_ST;
160       end if;
161     end process CURRENT_STATE_REG;
162
163     end architecture RTL_TICKET_MASTER;
```

Oppgave 15d).

For å simulere kretsen i oppgave 15c) må man påtrykke inngangene stimuli og sjekke utgangene.



I VHDL gjør man dette ved å lage en testbenk. I sin enkleste er den bygd opp på følgende måte

1. En (vanligvis) tom entitet for selve testbenken. Dvs. testbenken har vanligvis ikke noe interface mot verden utenfor men er ”selfcontained”.
2. Komponentdeklarasjon for UUT (Unit Under Test)
3. Deklarsjon av input stimuli signaler
4. Definisjon av klokke
5. Instantiering av UUT
6. Stimuli process der man lager en sekvens av input signaler
7. Sjekker output i ”Waveform-viewer”

I mer avanserte testbenker kan man istedenfor stimuliprosessen påtrykke inputstimuli ved å benytte simuleringsmodeller av omkringliggende kretser og instantiere disse i testbenken. Selve testbenken kan bli vesentlig enklere på denne måten.

Et annet alternativ er å hente input stimuli fra fil.

En mer avansert måte å sjekke korrekt funksjon er å lage en fasit over forventede verdier på utgangene og sammenligne disse med utgangene av UUT. Fasiten kan man lagre i en egen fil eller inni testbenken.
På denne måten kan testbenken være selvtestende og man kan slippe å studere timingdiagrammer. Man kan bare rapportere om resultatet er OK eller ikke.

INF3430/INF4430 Fasit eksamen Høst 2007 Oppgave 1 – 14

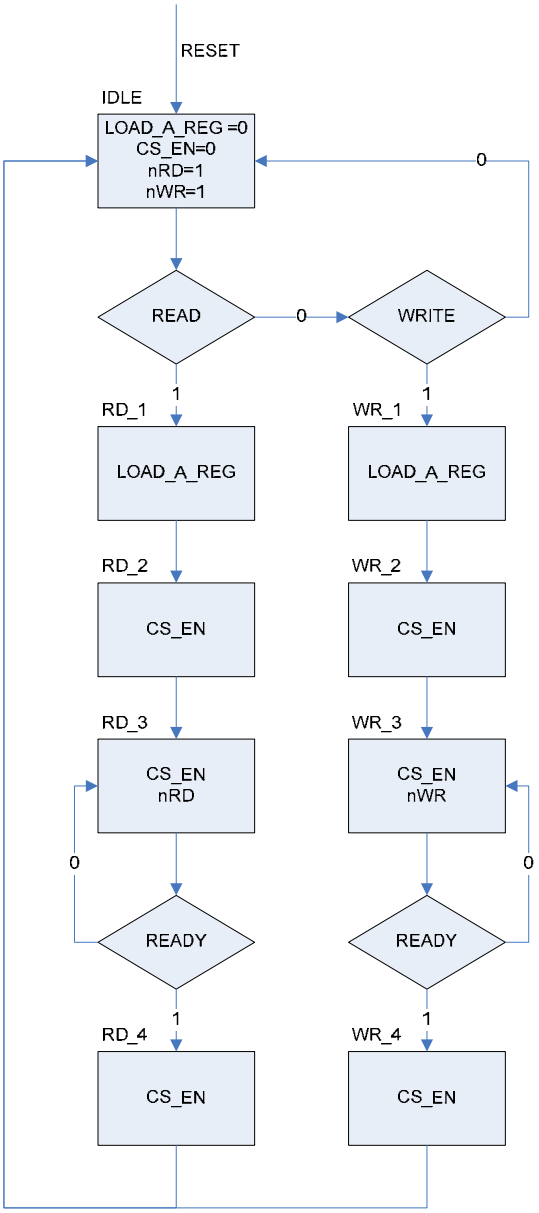
Oppgave	A	B	C	D	E
1	O		O	O	
2		O	O		
3	O	O		O	
4		O		O	
5			O		
6	O			O	
7		O	O		
8	O	O	O		
9		O		O	
10		O	O	O	
11	O			O	
12		O			
13	O	O	O		
14	O	O		O	

Oppgave 15.

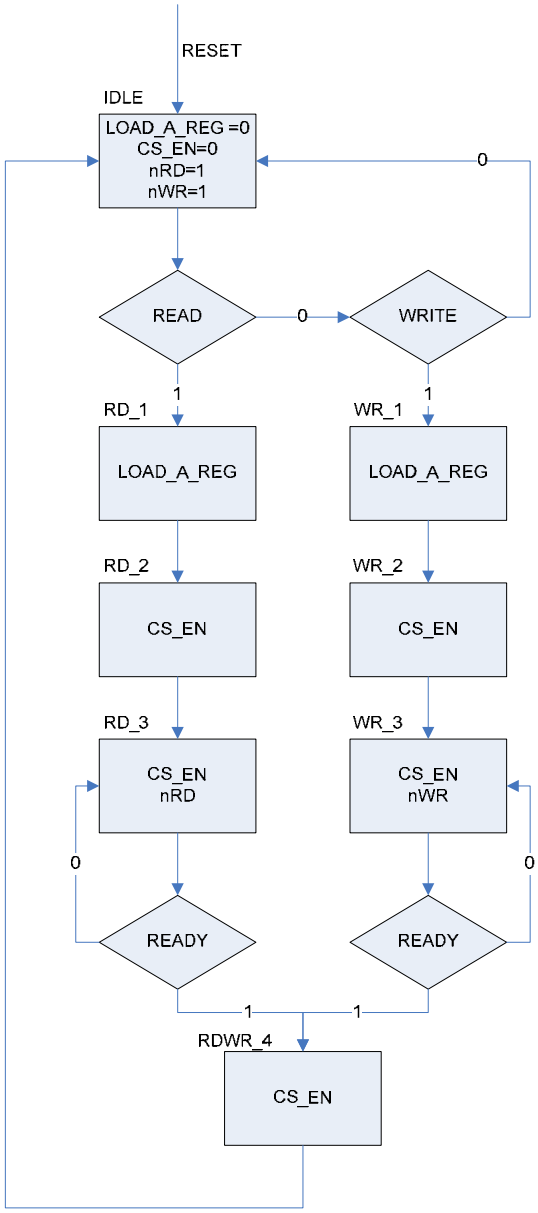
a).

```
--Oppgave 15a).
--==============
AIN <= A(17 downto 16);
--Alt.A
ADDRESS_DECODER:
process(CS_EN,AIN)
begin
  nCS <= "1111";
  if CS_EN = '1' then
    case AIN is
      when "00"   => nCS <= "1110";
      when "01"   => nCS <= "1101";
      when "10"   => nCS <= "1011";
      when others => nCS <= "0111";
    end case;
  end if;
end process;
end;

--Alt.B
ADDRESS_DECODER:
process(CS_EN,AIN)
begin
  nCS <= "1111";
  if CS_EN = '1' then
    if AIN = "00" then
      nCS(0) <= '0';
    elsif AIN = "01" then
      nCS(1) <= '0';
    elsif AIN = "10" then
      nCS(2) <= '0';
    elsif AIN = "11" then
      nCS(3) <= '0';
    end if;
  end if;
end process;
```

b).



Alternativ A



Alternativ B

c).

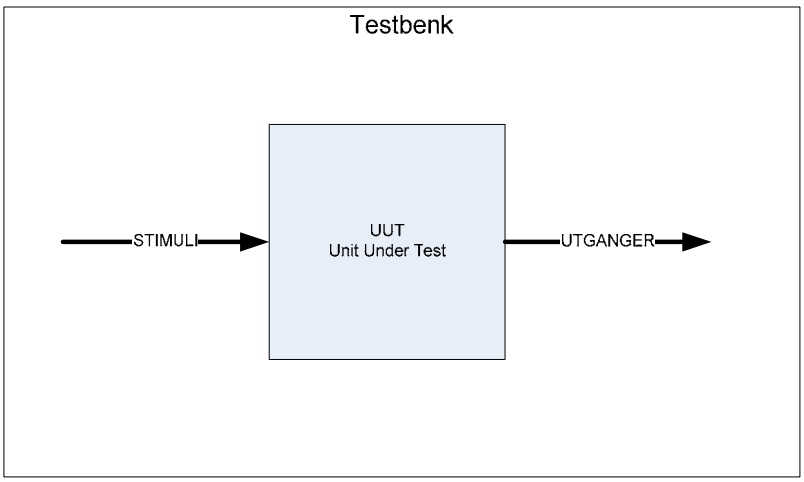
```
  type IO_CONTROLLER_TYPE is (IDLE, RD_1,RD_2,RD_3,RD_4,
                              WR_1,WR_2,WR_3,WR_4);
--eller
  --type IO_CONTROLLER_TYPE is (IDLE, RD_1,RD_2,RD_3,,
  --                            WR_1,WR_2,WR_3,RDWR_4);
  signal IO_CTRL_ST,IO_CTRL_NEXT_ST   : IO_CONTROLLER_TYPE;

  --Assumes that all signals declared in the entity
begin

IO_CONTROLLER_COMB:
process(READ,WRITE,READY,IO_CTRL_ST)
begin
  nRD     <= '1';
  nWR     <= '1';
  CS_EN   <= '0';
  LOAD_A_REG <= '0';
  IO_CTRL_NEXT_ST <= IDLE;

  case IO_CTRL_ST is
    when  IDLE =>
      if READ = '1' then
        IO_CTRL_NEXT_ST <= RD_1;
      end if;
      if WRITE = '1' then
        IO_CTRL_NEXT_ST <= WR_1;
      end if;
    --Read control
    when RD_1 =>
      LOAD_A_REG <= '1';
      IO_CTRL_NEXT_ST <= RD_2;
    when RD_2 =>
      CS_EN <= '1';
      IO_CTRL_NEXT_ST <= RD_3;
    when RD_3 =>
      CS_EN <= '1';
      nRD <= '0';
      if READY = '0' then
        IO_CTRL_NEXT_ST <= RD_3;
      --Alt A
      else
        IO_CTRL_NEXT_ST <= RD_4;
```

```
      --Alt B
        --IO:CTRL_NEXT_ST <= RDWR_4;
      end if;
    --Alt A
    when RD_4 =>
      CS_EN <= '1';
      IO_CTRL_NEXT_ST <= IDLE;
    --Alt B
    --when RDWR_4 =>
    -- CS_EN <= '1';
    -- IO_CTRL_NEXT_ST <= IDLE;
    --Write control
    when WR_1 =>
      LOAD_A_REG <= '1';
      IO_CTRL_NEXT_ST <= WR_2;
    when WR_2 =>
      CS_EN <= '1';
      IO_CTRL_NEXT_ST <= WR_3;
    when WR_3 =>
      CS_EN <= '1';
      nWR <= '0';
      if READY = '0' then
        IO_CTRL_NEXT_ST <= WR_3;
      else
        --Alt A
        IO_CTRL_NEXT_ST <= WR_4;
        --Alt B
        --IO_CTRL_NEXT_ST <= RDWR_4;
      end if;
    when WR_4 =>
      CS_EN <= '1';
      IO_CTRL_NEXT_ST <= IDLE_ST;

    when others => --Unecessary when using default values
      IO_CTRL_NEXT_ST <= IDLE;
  end case;
end process IO_CONTROLLER_COMB;
```

```
IO_CONTROLLER_STATE_REG:
process (RESET,CLK)
begin
  if RESET = '1' then
    IO_CTRL_ST <= IDLE;
  elsif rising_edge(CLK) then
    IO_CTRL_ST <= IO_CTRL_NEXT_ST;
  end if;
end process IO_CONTROLLER_REG;
```

d).

For å simulere kretsen i oppgave 15c) må man påtrykke inngangene stimuli og sjekke utgangene.



I VHDL gjør man dette ved å lage en testbenk. I sin enkleste er den bygd opp på følgende måte

1. En (vanligvis) tom entitet for selve testbenken. Dvs. testbenken har vanligvis ikke noe interface mot verden utenfor men er ”selfcontained”.
2. Komponentdeklarasjon for UUT (Unit Under Test)
3. Deklarsjon av input stimuli signaler
4. Definisjon av klokke
5. Instantiering av UUT
6. Stimuli process der man lager en sekvens av input signaler
7. Sjekker output i ”Waveform-viewer”

I mer avanserte testbenker kan man istedenfor stimuliprosessen påtrykke inputstimuli ved å benytte simuleringsmodeller av omkringliggende kretser og instantiere disse i testbenken. Selve testbenken kan bli vesentlig enklere på denne måten.

Et annet alternativ er å hente input stimuli fra fil.

En mer avansert måte å sjekke korrekt funksjon er å lage en fasit over forventede verdier på utgangene og sammenligne disse med utgangene av UUT. Fasiten kan man lagre i en egen fil eller inni testbenken.
På denne måten kan testbenken være selvtestende og man kan slippe å studere timingdiagrammer. Man kan bare rapportere om resultatet er OK eller ikke.

Fasit INF3430/4430 Eksamen H-2008

Oppgave 1-14

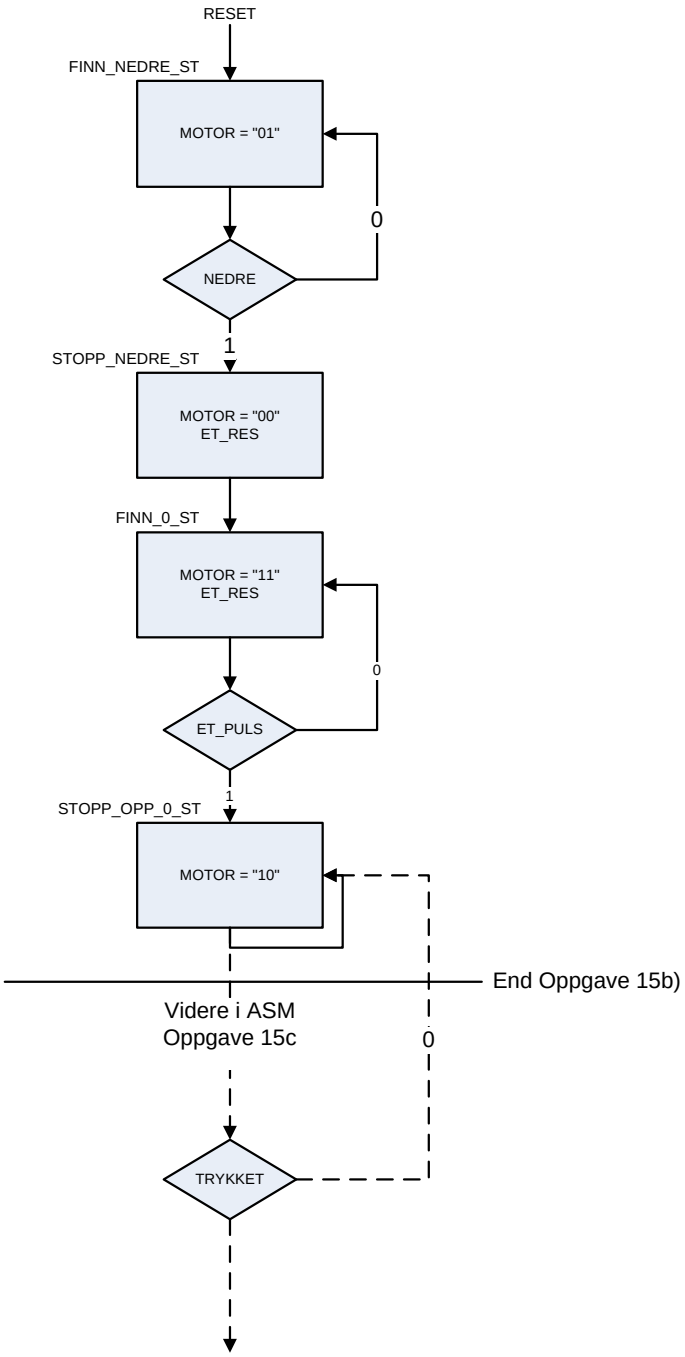
Oppgave	A	B	C	D
1		X		X
2		X		
3	X		X	X
4		X		X
5	X			
6			X	X
7	X	X		
8		X	X	X
9	X		X	X
10	X		X	
11	X			X
12	X	X		X
13		X		
14		X		

Oppgave 15

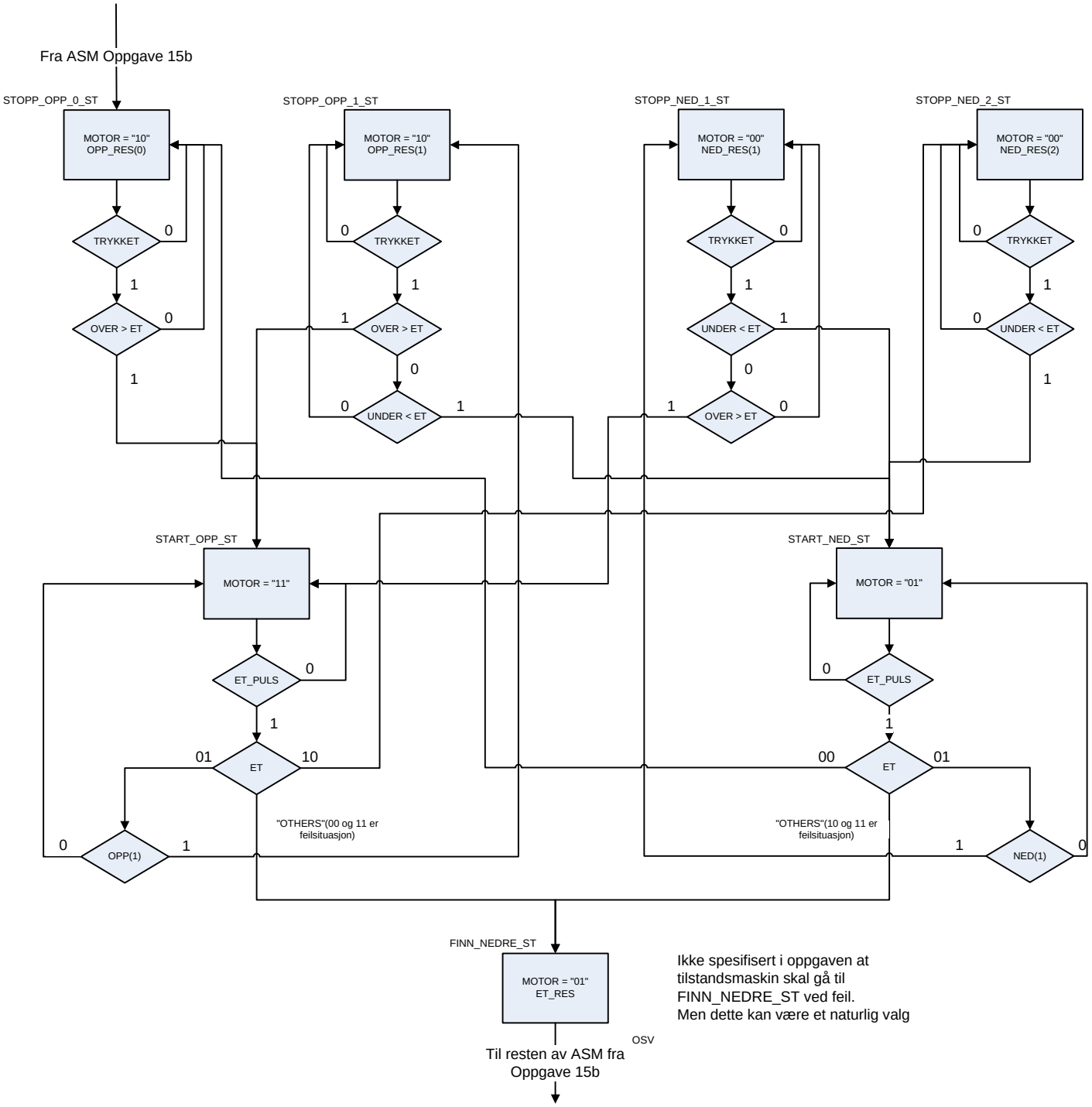
a)

```
-- Start Oppgave 15a)
OVER_COMB:
process(opp,ned)
begin
  if unsigned(opp) /= 0 or unsigned(ned) /= 0 then
    trykket <= '1';
  else
    trykket <= '0';
  end if;

  over <= "00";
  if ned(2) = '1' then
    over <= "10";
  elsif opp(1) = '1' or ned(1) = '1' then
    over <= "01";
  elsif opp(0) = '1' then
    over <= "00";
  end if;
end process;
--End oppgave 15a)
```

b)


RESET
FINN_NEDRE_ST
MOTOR = "01"
0
NEDRE
1
STOPP_NEDRE_ST
MOTOR = "00"
ET_RES
FINN_0_ST
MOTOR = "11"
ET_RES
0
ET_PULS
1
STOPP_OPP_0_ST
MOTOR = "10"
End Oppgave 15b)
Videre i ASM
Oppgave 15c
0
TRYKKET

c) Implementert som Moore maskin (Kan også implementeres som Mealy maskin)



d)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

architecture oppgave15_moore_rtl of oppgave15 is

  constant STOPP_NED : std_logic_vector(1 downto 0) := "00";
  constant START_NED : std_logic_vector(1 downto 0) := "01";
  constant STOPP_OPP : std_logic_vector(1 downto 0) := "10";
  constant START_OPP : std_logic_vector(1 downto 0) := "11";

  type heis_st_type is (FINN_NEDRE_ST,FINN_0_ST,STOPP_NEDRE_ST,
                        STOPP_OPP_0_ST);
  signal heis_st,heis_next_st : heis_st_type;

begin

  --Start Oppgave 15d)
  HEIS_ST_COMB:
  process(et,et_puls,nedre,heis_st)
  --Må legge til opp,ned,trykket på sensitivitestlisten
  --for komplett styring
  begin
    et_res  <= '0';
    opp_res <= "00";
    ned_res <= "00";
    heis_next_st <= FINN_NEDRE_ST;

    case heis_st is
      when  FINN_NEDRE_ST =>
        motor <= START_NED;
        opp_res <= "11";
        ned_res <= "11";
        heis_next_st <= FINN_NEDRE_ST;
        if nedre = '1' then
          heis_next_st <= STOPP_NEDRE_ST;
        end if;

      when STOPP_NEDRE_ST =>
        motor <= STOPP_NED;
        et_res <= '1';
        heis_next_st <= FINN_0_ST;

      when FINN_0_ST =>
        motor <= START_OPP;
        et_res <= '1';
        heis_next_st <= FINN_0_ST;
        if et_puls = '1' then
          heis_next_st <= STOPP_OPP_0_ST;
        end if;

      when STOPP_OPP_0_ST =>
        motor <= STOPP_OPP;
        opp_res(0) <= '1';
        heis_next_st <= STOPP_OPP_0_ST;
--    Ikke nødvendig for oppgave 15b
--        if trykket = '1' then
```

```
--           if over > et then
--             heis_next_st <= START_OPP_ST;
--           end if;
--         end if;

      when others =>
        motor <= STOPP_NED;
        heis_next_st <= FINN_NEDRE_ST;

    end case;

  end process;

  HEIS_ST_REG:
  process(reset,clk)
  begin
    if reset = '1' then
      heis_st <= FINN_NEDRE_ST;
    elsif rising_edge(CLK) then
      heis_st <= heis_next_st;
    end if;
  end process;

end architecture oppgave15d_moore_rtl;
```

e)

Man kan benytte f.eks. signalet `et_puls` til å sette en teller til en verdi som kan lage en passende pause . Telleren teller ned til 0. Når telleren er 0 er signalet `timeout` aktivt. `timeout` benyttes som en betingelse sammen med `trykket` og `over/under` for å lage en forsinket start.