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

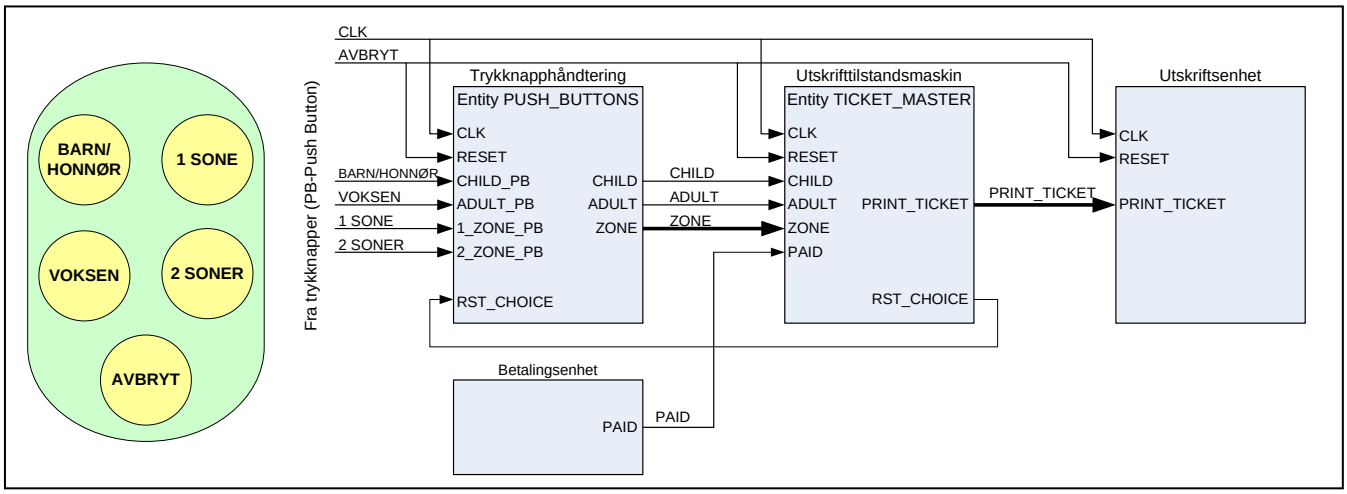
SystemC	A	Språket muliggjør design på forskjellig abstraksjonsnivå	
	B	Språket er lite egnet til verifisering	
	C	Syntese basert på SystemC kan gjøres direkte eller ved først å konvertere til VHDL	
	D	Språket egner seg godt til HW/SW codesign	

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;
```

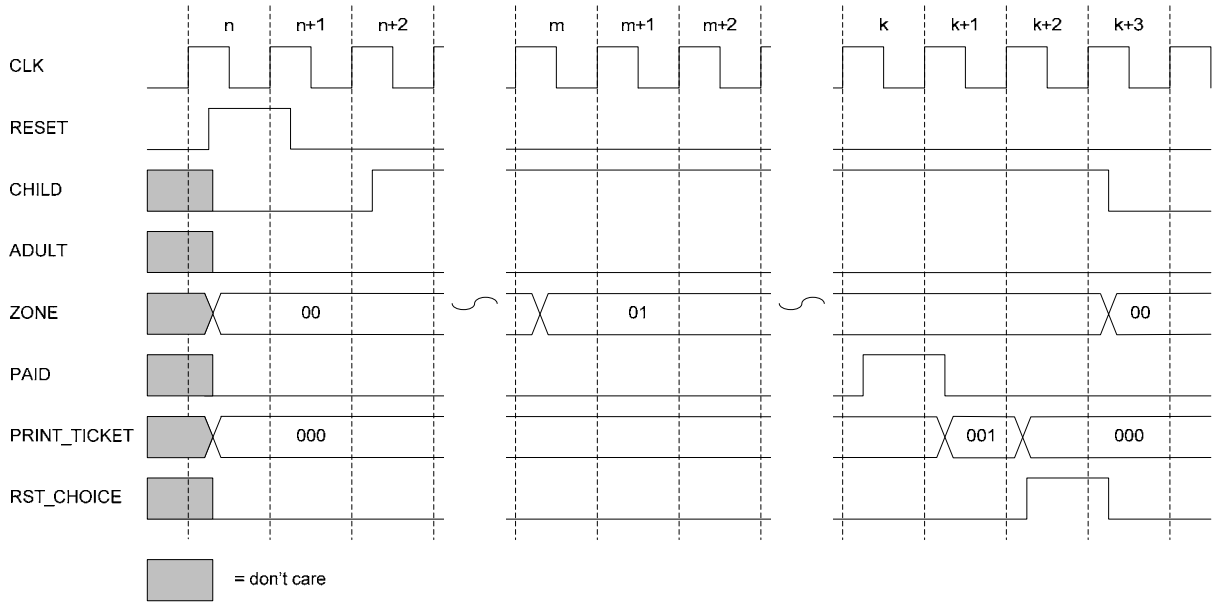
Man er nødt til å velge ”BARN/VOKSEN” *og* antall soner *før* man betaler. Etter at gyldig betaling er mottatt går signalet PAID aktivt i *en* klokkeperiode. (Automaten veksler, men dette er styrt av en annen tilstandsmaskin (Betalingsenhet). Samme tilstandsmaskin gir også pengene i retur dersom valg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

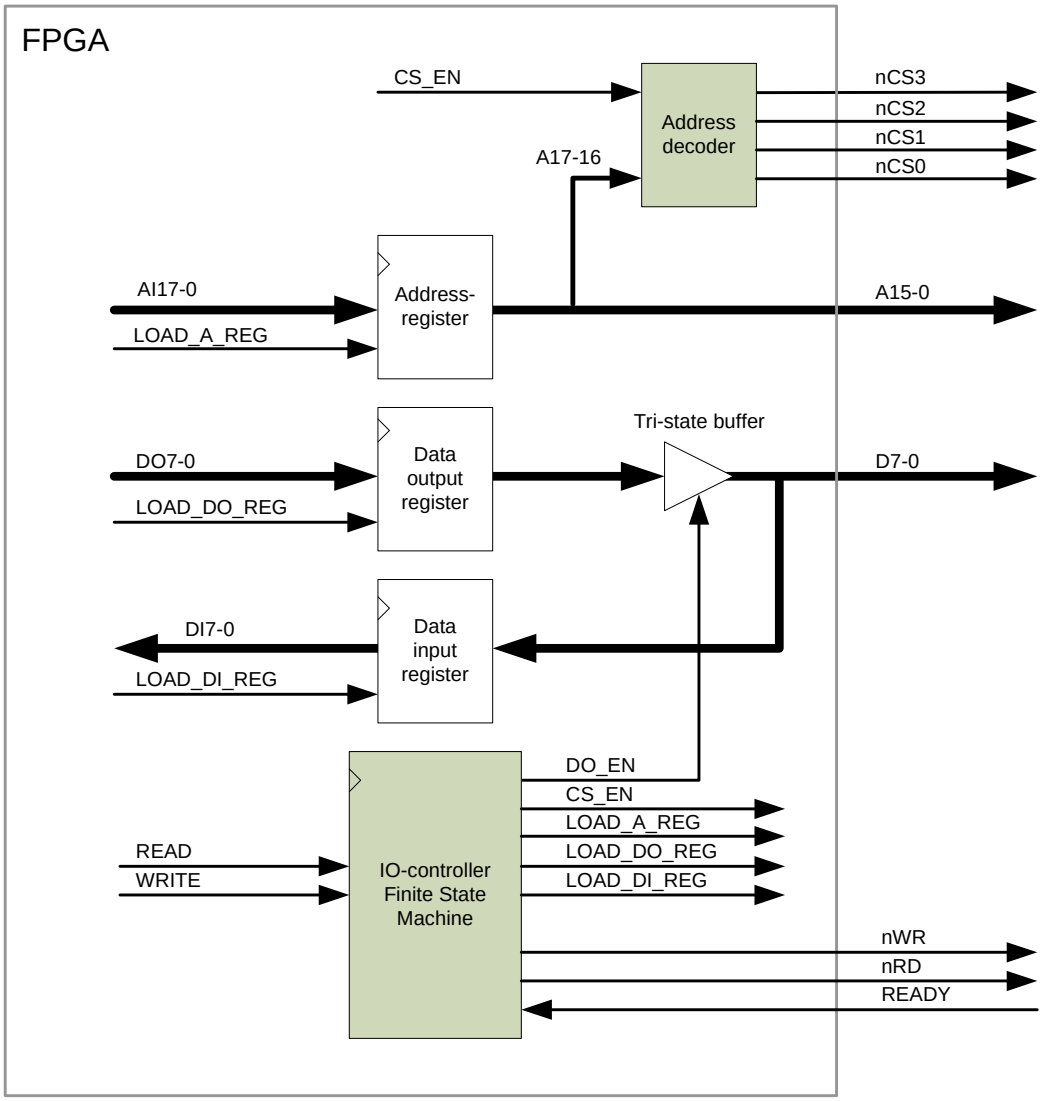
Oppgave 14

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

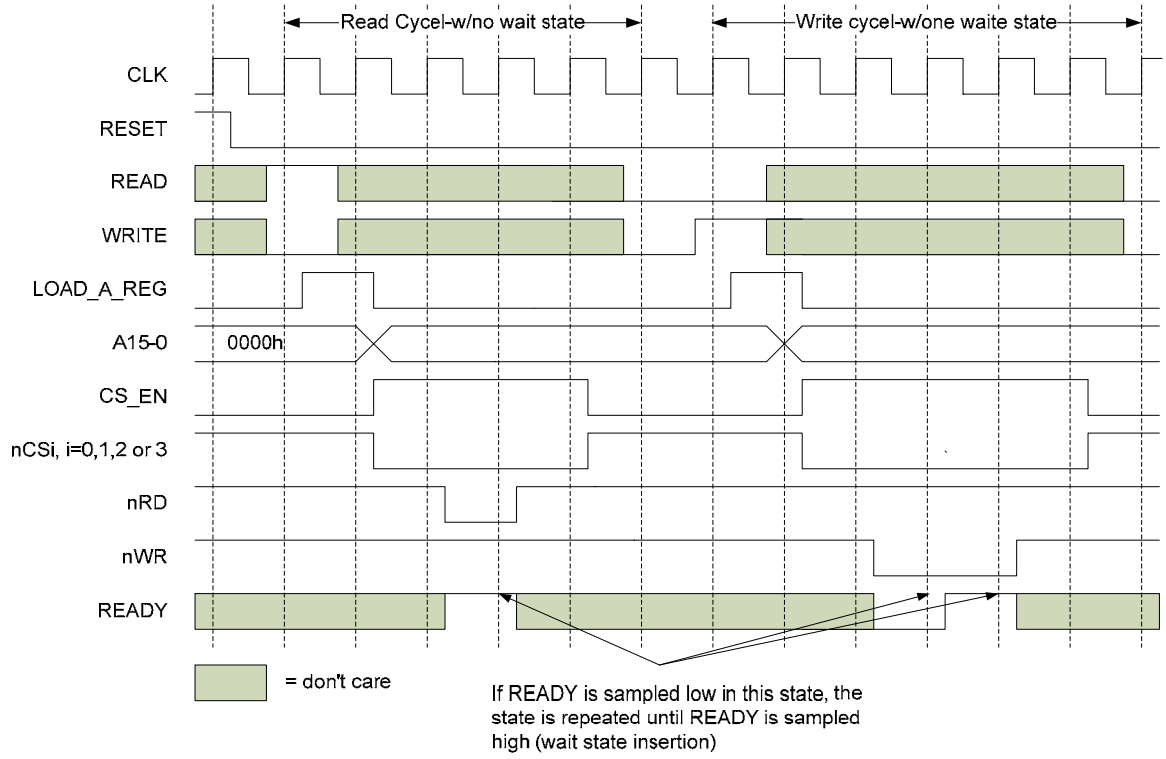
Signalnavn	**Beskrivelse**	**Retning**
Klokke og reset er ikke vist i figur 1		
CLK	50MHz systemklokke	Input til alle registre
RESET	Asynkron reset. Aktivt høyt	Input til alle registre
Eksterne signaler:		
A15-0	Adresse signaler	Output
D7-0	Data signaler	Input/Output/Tri-state
nCSi, i=0,1,2,3	Chip select 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

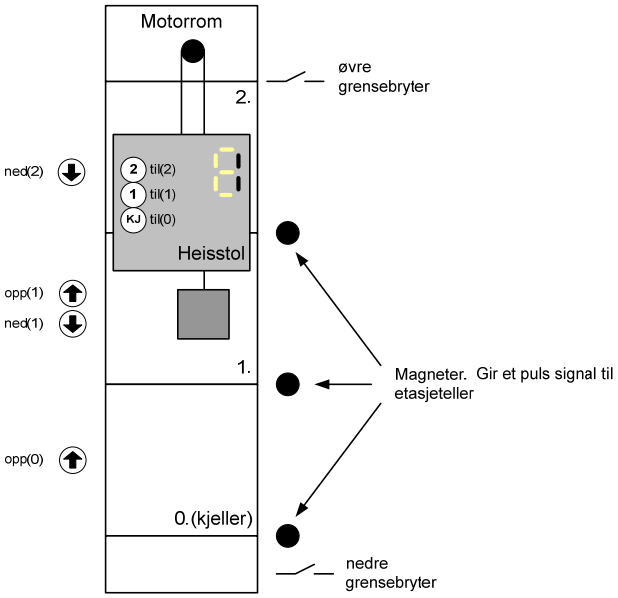
I VHDL koden vist under settes `indata` til '1' og vil gi outdata(7 downto 0) lik:	A	"10001011"	
	B	"01000111"	
	C	"10001011"	
	D	"10000111"	

```
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.

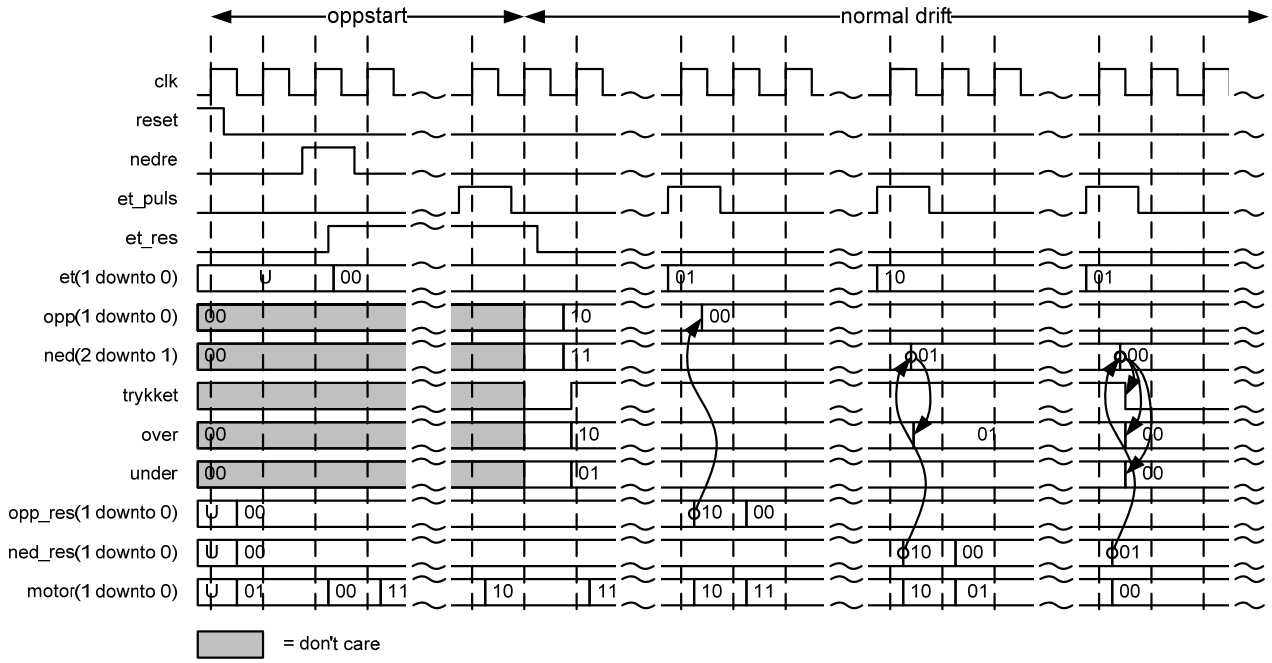
Innganger				Utganger	
ned(2)	**opp(1)**	**ned(1)**	**opp(0)**	**over**	**trykket**
0	0	0	0	00	0
0	0	0	1	00	1
0	X	1	X[2]	01	1
0	1	X	X	01	1
1	X	X	X	10	1

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				

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430/INF4430 Digital systemkonstruksjon**
Eksamensdag:	**3. desember 2009**
Tid for eksamen:	**9 - 12**
Oppgavesettet er på 9 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Mark Zwolinski** **Digital System Design with VHDL, 2nd edition** **Prentice Hall, 2004.**

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

Oppgaveteksten består av oppgave 1 – 6 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 7-12 som besvares på vanlige ark. Oppgave 1 - 6 har til sammen vekt på 25%, mens oppgave 7 -11 har til sammen vekt på 25% og oppgaven 12 har til sammen vekt på 50%.

Generelt for oppgave 1-6:
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

VHDL og simulering	A	Det er raskere og enklere å simulere med variabler enn med signaler i en process.	
	B	Initialverdien etter deklarasjon av et signal av typen std_logic vil være ’0’.	
	C	Alle variabler som er deklarert i en process vil ikke bli satt tilbake til sin initialverdi neste gang processen utføres.	
	D	For hver deltacycle går tiden et picosekund.	
	E	Etter en tid i metastabil tilstand vil alle flip-flop’er alltid gå til verdien ’0’.	

Oppgave 2

En 3-input Xilinx LUT med innhold ”01” (hex) realiserer en:	A	AND funksjon	
	B	NAND funksjon	
	C	NOR funksjon	
	D	XOR funksjon	
	E	OR funksjon	

Oppgave 3

Konfigurasjon og rekonfigurasjon av Xilinx FPGA	A	En SRAM FPGA er det ikke mulig å finne ut hvordan fungerer (dvs. ved ”reverse-engineering”) ut i fra konfigurasjonsfilen.	
	B	Ved delvis rekonfigurasjon vil de delene av kretsen som ikke blir rekonfigurert også være inaktive under rekonfigureringen.	
	C	Et problem ved dynamisk rekonfigurering er vanligvis for lang rekonfigureringstid.	
	D	En mikroprocessor kan konfigurere en FPGA som er i slave mode.	
	E	Block RAM’er har ikke en kjent initialverdi etter konfigurering.	

Oppgave 4

Design	A	Intellectual Property (IP) er i FPGA teknologi en betegnelse på myke ferdigutviklede moduler.	
	B	En BUFG modul kan bare brukes til klokkesignaler.	
	C	Retiming brukes av synteseverktøyet for å lettere oppnå timingkrav og det gir aldri økt forbruk av registere (flip-flop’er).	
	D	I Spartan 3 teknologi er vanligvis forsinkelsen gjennom en LUT lengre enn typisk forsinkelse mellom LUT’er.	
	E	En hard IP kjerne tar mindre plass enn en tilsvarende myk IP kjerne.	

Oppgave 5

Design	A	Det er samme begrensning med antall klokkenett i ASIC teknologi som i FPGA teknologi.	
	B	Et problem med DCM moduler er at ”jitter” øker.	
	C	En hard IP kjerne som ikke brukes frigjør plass til annen logikk.	
	D	En Flash FPGA er umiddelbart aktiv etter strømtilkobling.	
	E	Det må vanligvis brukes registere (flip-flop’er) i en tilbakekobling i FPGA.	

Oppgave 6

Høyhastighets-linker	A	Høyhastighetslinker har i tillegg til de differensielle datalinjene også differensielle klokkelinjer.	
	B	For PCIe gen. 1 er faktisk datarate 2.0 Gbit/s med 8B/10B koding som gjør at linjens baudrate blir 2.5 Gbit/s.	
	C	Det er bare når commategn sendes at sender utfører pre-emphasis.	
	D	Commategn brukes for å dele opp lange bitstrenger med mange påfølgende ’1’ og mange påfølgende ’0’.	
	E	De differensielle linjene fra en sender kan gå til opptil 4 mottagere.	

Oppgave 7

I VHDL koden oppgitt under får signalet ”b” verdier. Hvilke verdier får signalet ”b” i tidsrommet fra til 0 ns til 90 ns?

```
library ieee;
use ieee.std_logic_1164.all;

entity signal_values is
end signal_values;

architecture beh of signal_values is
  signal b : std_logic_vector(3 downto 0);
begin

  P_0 : process
  begin
    b <= x"1", x"3" after 20 ns, x"5" after 40 ns, x"7" after 60 ns;
    wait for 30 ns;
    b <= x"9";
    b <= x"A", x"C" after 20 ns, x"E" after 40 ns;
    wait for 50 ns;
  end process P_0;

end beh;
```

Oppgave 8

Under er det oppgitt funksjonen ”something”. Gi en kort forklaring med ord om hva funksjonen gjør?

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

entity somefunction is
  port (
    -- System Clock and Reset
    rst_n   : in  std_logic;
    mclk    : in  std_logic;
    din     : in  std_logic_vector(31 downto 0);
    result  : out std_logic_vector(5 downto 0)
  );
end somefunction;

architecture rtl of somefunction is

  function something(a : std_logic_vector)
      return integer is
    constant ALEN  : integer := a'length;
    variable value : integer;
  begin
    value:= ALEN;
    for i in ALEN-1 downto 0 loop
      if a(i)='1' then
        value := i;
        exit;
      end if;
    end loop;
    return value;
  end;

begin

  process (rst_n, mclk) is
    variable tmpres : integer;
  begin
    if (rst_n = '0') then
      tmpres := 0;
      result <= (others => '0');
    elsif rising_edge(mclk) then
      tmpres := something(din);
      result <= std_logic_vector(to_unsigned(tmpres,6));
    end if;
  end process;

end rtl;
```

Oppgave 9

Skriv om funksjonen ”something” oppgitt i oppgave 8 til en procedure ”something” med samme funksjonalitet som i oppgave 8.

Oppgave 10

Skriv om arkitekturen ”rtl1” av entiteten ”to_concurrent” til en ny architecture ”rtl2” som **bare** har concurrent statements.

```
library ieee;
use ieee.std_logic_1164.all;

entity to_concurrent is
  port (
    a       : in  std_logic;
    b       : in  std_logic;
    c       : in  std_logic;
    d       : in  std_logic;
    ena     : in  std_logic;
    abc_out : out std_logic;
    d_out   : out std_logic
  );
end to_concurrent;

architecture rtl1 of to_concurrent is
begin

  process (a,b,c,d) is
    variable x : std_logic;
  begin
    x := a and b;
    abc_out <= x or c or d;
  end process;

  process (ena, d) is
  begin
    if ena='1' then
      d_out <= d;
    else
      d_out <= 'Z';
    end if;
  end process;

end rtl1;
```

Oppgave 11

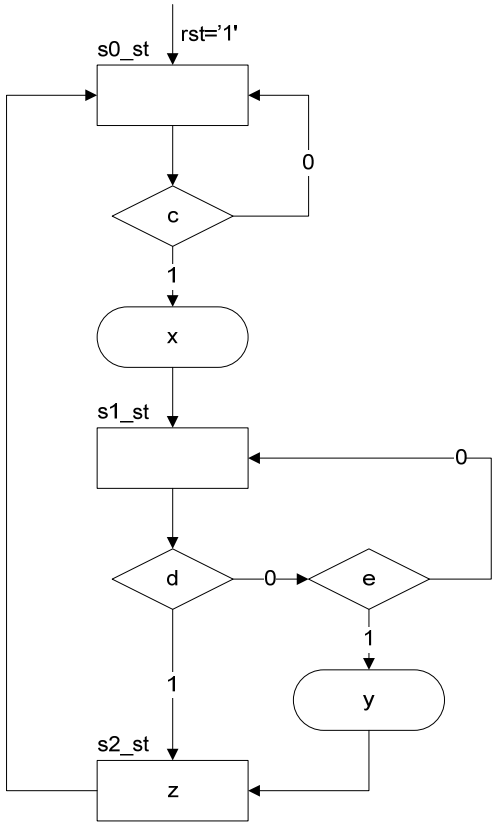
I VHDL koden under er det oppgitt en package ”mypack”, en entity ”dager” og en uferdig architecture ”rtl”.

Lag ferdig processen i ”rtl” så den setter ut antall dager i måneden i ”antdager” med kun de oppgitte inngangsdata. Når det er skuddår er det 29 dager i februar, ellers 28 dager. Ellers er det 31 dager i januar, mars, mai, juli, august, oktober, desember og 30 dager i de resterende.

```
package mypack is
  type month_type is (JAN, FEB, MAR, APR, MAY, JUN,
                      JUL, AUG, SEP, OCT, NOV, DEC);
end mypack;

library ieee;
use ieee.std_logic_1164.all;
use work.mypack.all;

entity dager is
  port (
    mnd      : in  month_type;
    skuddaar : in  boolean;
    antdager : out std_logic_vector(4 downto 0)
  );
end dager;

architecture rtl of dager is
begin

  P_DAYS: process(mnd, skuddaar)
  begin
    -- Begynn å skriv VHDL kode her:
    :
    :
    :
    -- Slutt på skriv VHDL kode.
  end process;
end rtl;
```

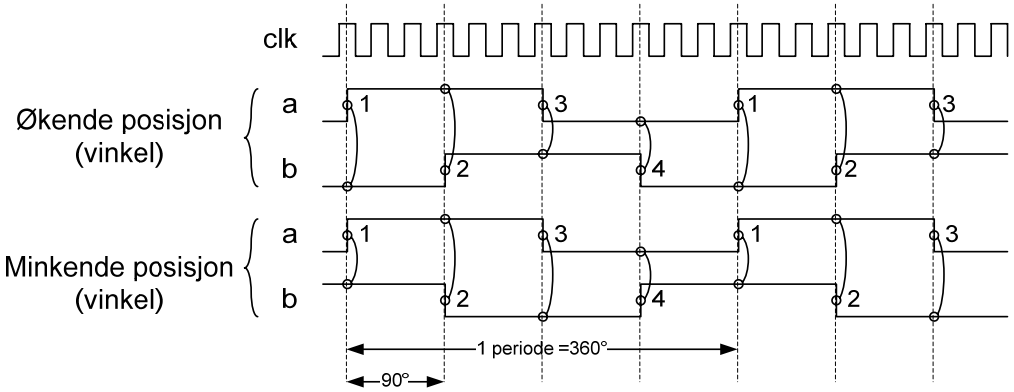
Oppgave 12. Vekt 50%

a) Vekt 15%
Implementer ASM-flytdiagrammet under VHDL.
Hva slags type tilstandsmaskin beskriver ASM-flytdiagrammet under?
Hva er forskjellen mellom en Mealy og Moore tilstandsmaskin?



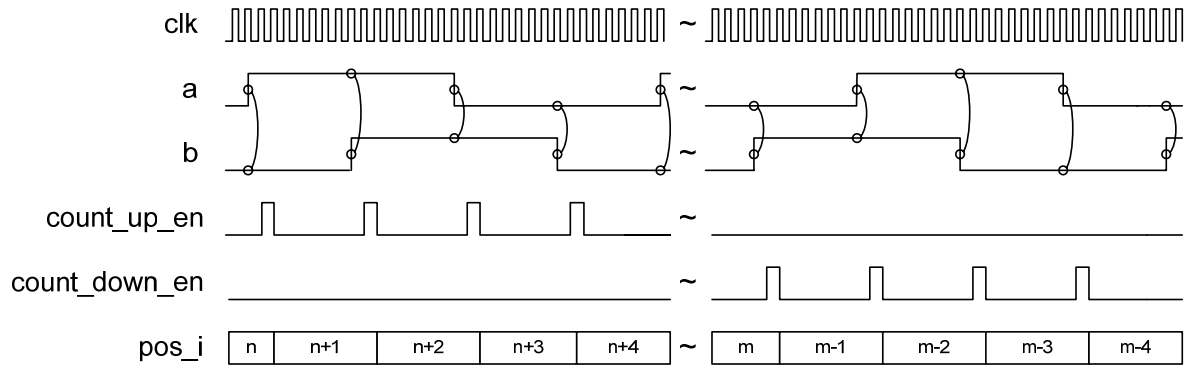
I de etterfølgende deloppgavene b-c skal vi beskrive en posisjonsmåler basert på en en såkalt ”shaft encoder”. En ”shaft encoder” er velegnet til å koble til en aksling (derav navnet) og kan benyttes til å måle en relativ vinkel. Den relative vinkelen kan f.eks. benyttes til å representere posisjonen til en robotarm. Forutsetningen for å vise posisjon er at posisjonsmåleren må resettes til en kjent verdi(0) i en valgt posisjon, som typisk kan være en endeposisjon. I det etterfølgende vil vinkel og posisjon bety det samme.

Internt består ”shaft encoderen” av to par med lysdioder/fototransistorer. De er arrangert på en slik måte at de gir ut to signaler ”a” og ”b” som er 90° faseforskjøvet i forhold til hverandre. Når asklingen roterer den ene veien er ”a” 90° foran ”b”, og ved rotasjon motsatt vei er ”a” 90° etter ”b”. Ved ro og ved oppstart kan ”a” og ”b” være ’0’ eller ’1’. Dvs. ”a” og ”b” kan innta hvilken som helst av de 4 mulige kombinasjonene: (’0’,’0’), (’0’, ’1’), (’1’, ’0’) eller (’1’, ’1’). Encoderen har videre den egenskapen at den gir ut 360 perioder av signalene ”a” og ”b” på en hel omdreining. Det betyr 1° pr. periode. Vi kan imidlertid måle posisjonen med en nøyaktighet på 0.25° dersom vi utnytter hver eneste av de 4 flankene til ”a” og ”b” innenfor en periode. Og det er det denne oppgaven går ut på. Se figuren under:



Vi antar at signalene ”a” og ”b” er synkrone med klokken ”clk” og at de er prellfrie. Videre så er frekvensen til ”clk” mye større enn frekvensen til ”a” og ”b”.

Posisjonsmåleren skal implementeres som en tilstandsmaskin som skal gi utgangssignalene ”count_up_en” og ”count_down_en”. Et av disse signalene skal gå aktivt etter hver flanke av ”a” eller ”b”. Disse skal styre en posisjonsteller, ”pos_i”. ”pos_i” skal telle oppover når ”count_up_en” er aktiv ’1’ og nedover når ”count_down_en” er aktiv ’1’. Telleren ”pos_i” skal alltid befinne seg i intervalet [0,1439(59Fh)]. Når signalet ”rst” går aktivt til ’1’ skal ”pos_i” resettes til 0.



b) Vekt 20%.
Lag et ASM flyskjema som en Moore type tilstandsmaskin for å styre posisjonstelleren ”pos_i” som beskrevet over i tekst og i timingdiagram.

c) Vekt 15%
Implementer tilstandsmaskinen beskrevet i deloppgave b) i VHDL. I tillegg til selve tilstandsmaskinen skal koden også inneholde en process for selve posisjonstelleren, ”pos_i”, og biblioteksreferanser til benyttede standardbiblioteker. Man kan anta at posisjonsmåleren har følgende entitet:

```
entity oppgave12c is
  port
  (
    clk     : in  std_logic;
    rst     : in  std_logic;
    a       : in  std_logic;
    b       : in  std_logic;
    pos     : out std_logic_vector(15 downto 0)
  );
end oppgave12c;
```

NB! Selv om du ikke har fått til oppgave 12b så kan du fortsatt implementere "pos"/"pos_i"signalene (se entiteten).

INF3430 / INF4430. Oppgavesvar for kandidat nr: _____

Oppgave	A	B	C	D	E
1					
2					
3					
4					
5					
6					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430 Digital systemkonstruksjon**
Eksamensdag:	**8. desember 2010**
Tid for eksamen:	**9-13**
Oppgavesettet er på 10 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Mark Zwolinski** **Digital System Design with VHDL, 2nd edition** **Prentice Hall, 2004.**

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

Oppgaveteksten består av oppgave 1–7 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 8-13 som besvares på vanlige ark. Oppgave 1-7 har til sammen vekt på 25%, mens oppgave 8-12 har til sammen vekt på 25% og oppgaven 13 har til sammen vekt på 50%.

Generelt for oppgave 1-7:
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

En 4-input Xilinx LUT med innhold ”7FFF” (hex) realiserer en:	A	AND funksjon	
	B	NAND funksjon	
	C	NOR funksjon	
	D	XOR funksjon	
	E	OR funksjon	

Oppgave 2

Signalverdier i VHDL av typen std_logic:	A	Initialverdien etter deklarasjon til et signal av typen std_logic er ’0’.	
	B	To signaler av typen std_logic med verdiene ’0’ og ’0’ som driver samme signal får verdien ’0’.	
	C	To signaler av typen std_logic med verdiene ’0’ og ’1’ som driver samme signal får verdien ’X’.	
	D	To signaler av typen std_logic med verdiene ’Z’ og ’1’ som driver samme signal får verdien ’Z’.	
	E	To signaler av typen std_logic med verdiene ’Z’ og ’1’ som driver samme signal får verdien ’X’.	

Oppgave 3

Konfigurasjon og lagringsteknologi	A	JTAG porten kan brukes både til konfigurasjon og til debugging.	
	B	En FPGA i master-modus styrer selv nedlasting av konfigurasjon ved oppstart.	
	C	En FPGA må alltid konfigureres serielt hvis den er i slave-modus.	
	D	SRAM FPGA’er har høyere sikkerhet enn Antifuse FPGA’er.	
	E	Antifuse FPGA’er kan bare konfigureres en gang.	

Oppgave 4

Design 1	A	Med en Digital Clock Manager modul kan man øke klokkesignalet til det dobbelt og det genererte signalet kan være i fase med inngangsklokken.	
	B	En hard prosessorkjerne tar vanligvis mindre plass enn en myk prosessorkjerne.	
	C	Xilinx Block RAM’er kan fjernes fra FPGA’en hvis de ikke brukes for å spare plass.	
	D	I et klokkedomene klokkes alle registre (flipflop’er) med samme klokkesignal.	
	E	FPGA er egnet for asynkron logikk.	

Oppgave 5

Design 2	A	Bruk av dedikert mentekjede (carry chain) gjør at det blir mindre tilgjengelig logikk i FPGA'en og bruken bør derfor begrenses.	
	B	I serielle linker har 8B/10B encoding mindre overhead enn 64B/66B enkoding.	
	C	Serielle linker er vanligvis enveissignaler (uni-direksjonale).	
	D	DSP-modulene "MAC" står for "Multiply-And-Compare	
	E	DSP-modulene til Xilinx er myke moduler som kan fjernes for å spare plass i FPGA'en.	

Oppgave 6

VHDL-koden "oppgave_delay" oppgitt under forsinker et input signal (etter at reset er blitt inaktivt) som oppgitt i testbenken "tb_oppgave_delay".

Når får signalet dout i testbenken verdien "11001100"?	A	250 ns.	
	B	300 ns.	
	C	350 ns.	
	D	400 ns.	
	E	450 ns.	

Oppgave 7

VHDL-koden "oppgave_delay" gitt under forsinker et input signal (etter at reset er blitt inaktivt) som oppgitt i testbenken "tb_oppgave_delay".

Når får signalet dout i testbenken verdien "00110011"?	A	400 ns.	
	B	450 ns.	
	C	500 ns.	
	D	550 ns.	
	E	600 ns.	

```
library ieee;
use ieee.std_logic_1164.all;

entity oppgave_delay is
  port (
    rst        : in  std_logic;
    mclk       : in  std_logic;
    din        : in  std_logic_vector(3 downto 0);
    dout       : out std_logic_vector(7 downto 0)
  );
end oppgave_delay;

architecture rtl of oppgave_delay is
  signal d2, d3 : std_logic_vector(3 downto 0);
  signal d5     : std_logic_vector(7 downto 0);
begin
```

```
  process (rst, mclk) is
    variable d1 : std_logic_vector(3 downto 0);
  begin
    if (rst = '1') then
      d1    := (others => '0');
      d2    <= (others => '0');
    elsif rising_edge(mclk) then
      d1    := din;
      d2    <= d1;
    end if;
  end process;

  process (rst, mclk) is
    variable d4 : std_logic_vector(3 downto 0);
  begin
    if (rst = '1') then
      d3    <= (others => '0');
      d4    := (others => '0');
      d5    <= (others => '0');
    elsif rising_edge(mclk) then
      d3    <= d2;
      d4    := d3;
      d5(7 downto 4) <= d4;
      d5(3 downto 0) <= d4;
    end if;
  end process;

  dout  <= d5;

end rtl;

library ieee;
use ieee.std_logic_1164.all;

entity tb_oppgave_delay is
  -- empty;
end tb_oppgave_delay;

architecture beh of tb_oppgave_delay is

  component oppgave_delay
    port (
      rst   : in  std_logic;
      mclk  : in  std_logic;
      din   : in  std_logic_vector(3 downto 0);
      dout  : out std_logic_vector(7 downto 0));
  end component;

  signal rst  : std_logic;
  signal mclk : std_logic:= '0';
  signal din  : std_logic_vector(3 downto 0);
  signal dout : std_logic_vector(7 downto 0);

begin

  P_oppgave_delay: oppgave_delay
    port map (
      rst  => rst,
      mclk => mclk,
      din  => din,
      dout => dout);
```

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

--Alternativ til processen over
--mclk <= not mclk after 50 ns;

  rst <= '1', '0' after 100 ns;
  din <= "1100", "0011" after 300 ns;

end beh;
```

Oppgave 8

I VHDL-koden oppgitt under får signalet ”a” verdier. Hvilke verdier får signalet ”a” i tidsrommet fra til 0 ns. til 100 ns.?

```
library ieee;
use ieee.std_logic_1164.all;

entity signal_values is
end signal_values;

architecture beh of signal_values is

  signal a : std_logic_vector(3 downto 0);

begin

  process
  begin
    a <= x"1", x"3" after 10 ns, x"5" after 20 ns, x"7" after 40 ns;
    wait for 30 ns;
    a <= x"9", x"A" after 10 ns, x"C" after 20 ns, x"E" after 40 ns;
    wait for 60 ns;
    a <= x"F";
  end process;

end beh;
```

Oppgave 9

I VHDL-koden til oppgave_counter er det oppgitt en uferdig process i architecture rtl_1.
Denne prosessen skal skrives ferdig i synthetiserbar VHDL kode.

Prosessen skal telle opp antall bit med verdien ’1’ i input vektoren ”data” og la output vektoren ”cnt” få denne verdien.

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

entity oppgave_counter is
  port (
    rst   : in  std_logic;
    mclk  : in  std_logic;
    data  : in  std_logic_vector(31 downto 0);
    cnt   : out std_logic_vector(7 downto 0)
  );
end oppgave_counter;

architecture rtl_1 of oppgave_counter is
begin

  process (rst, mclk) is
  -- Start å skrive VHDL kode her:
   :
   :
   :
  -- Slutt å skrive VHDL kode her.
  end process;

end rtl_1;
```

Oppgave 10

I VHDL-koden til oppgave_counter under er det oppgitt en uferdig procedure som brukes i arkitekturen rtl_2. Denne procedure skal skrives ferdig i synthetiserbar VHDL kode.

Procedure skal som i oppgave 9, telle opp antall bit med verdien ’1’ i input vektoren og la output vektoren få denne verdien.

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

entity oppgave_counter is
  port (
    rst   : in  std_logic;
    mclk  : in  std_logic;
    data  : in  std_logic_vector(31 downto 0);
    cnt   : out std_logic_vector(7 downto 0)
  );
end oppgave_counter;

architecture rtl_2 of oppgave_counter is

  procedure pcount
  -- Start å skrive VHDL kode her:
     -- Viktig del av oppgaven er input og output parametrene
     -- til procedure pcount
  :
  :
  -- Slutt å skrive VHDL kode her.
  end procedure;

begin

  process (rst, mclk) is
  begin
    if (rst = '1') then
      cnt <= (others => '0');
    elsif rising_edge(mclk) then
      pcount(data, cnt);
    end if;
  end process;

end rtl_2;
```

Oppgave 11

Skriv om architecture rtl_2 i oppgave 10 til en ny architecture rtl_3 hvor VHDL procedure “pcount” endres til VHDL function “fcount”, og vis hvordan function fcount brukes i rtl_3.

Oppgave 12

Under er det oppgitt arkitekturen rtl til ”something”. Gi en forklaring med ord om hva funksjonen gjør?

```
library ieee;
use ieee.std_logic_1164.all;

entity something is
  port(
    rst    : in  std_logic;
    mclk   : in  std_logic;
    a      : in  std_logic;
    res    : out std_logic
  );
end entity;

architecture rtl of something is

  signal a_s1, a_s2, a_d1, a_d2 : std_logic;

begin

  process (mclk, rst)
  begin
    if (rst = '1') then
      a_s1 <= '0';
      a_s2 <= '0';
    elsif rising_edge(mclk) then
      a_s1 <= a;
      a_s2 <= a_s1;
    end if;
  end process;

  process (mclk, rst)
  begin
    if (rst = '1') then
      a_d1 <= '0';
      a_d2 <= '0';
      res  <= '0';
    elsif rising_edge(mclk) then
      a_d1 <= a_s2;
      if a_d1=a_s2 then
        a_d2<= a_d1;
      end if;
      res  <= (not a_d1) and a_d2;
    end if;
  end process;

end architecture rtl;
```

Oppgave 13

Vi skal i denne oppgaven lage en en tilstandsmakin som implementerer en kodelås, noe tilsvarende det vi f.eks. finner i safer på hotellrom i dag.

Vi antar at kodelåstilstandsmaskinen har følgende entitet:

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity code_lock is
  port
  (
    reset     : in std_logic;  --asynkron reset
    clk       : in std_logic;  --klokke
    code_in   : in std_logic_vector(3 downto 0); --siffer(0-9) fra numerisk tastatur på
                                                 --safedøren
    entered   : in std_logic;  --aktivt en klokkeperiode når et siffer er trykket
    door      : in std_logic;  --aktivt når safedøren er lukket. Brukes for å unngå at
                               --låsemekanismen blir aktivert når døren er åpen
    open_door : out std_logic; --aktivt en klokkeperiode når safedør skal åpnes
    close_door: out std_logic; --aktivt en klokkeperiode når safedør skal låses
    error     : out std_logic; --aktivt når det er tastet inn feil 4-sifret code
    alarm     : out std_logic  --aktivt dersom feil kode er tastet inn for mange ganger
  );
end codelock;
```

Vi antar videre at alle inputsignaler er synkronisert med klokken *clk*, at alle inputsignaler er prellfrie og alle signaler (input og output) er aktiv høye.

Virkemåten er som følger:

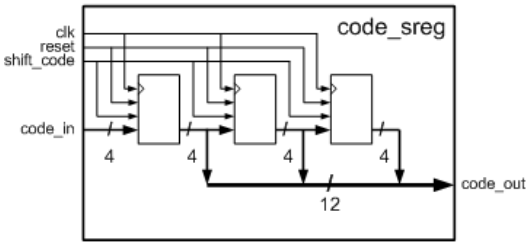
- Ved oppstart (reset) er døren åpen, alle utgangssignaler er inaktive.
- Tilstandsmaskinen er da klar til å lese inn en ny kode på 3 siffer (kun tallene 0-9).
- For hvert siffer som tastes inn går signalet *entered* aktivt en klokkeperiode. Hvert siffer må lagres.
- Etter at 3 siffer er lest inn og lagret går signalet *close_door* aktivt en klokkeperiode dersom *door* er aktivt. *close_door* aktivt aktiverer en mekanisme som låser safedøren.
- Etter dette er tilstandsmaskinen klar til å ta i mot 3 siffer som skal låse opp døren. Som før går *entered* aktivt en klokkeperiode for hvert siffer som tastes inn. Etter at de 3 sifrene er tastet inn sammenlignes de med koden som aktiverte låsing.

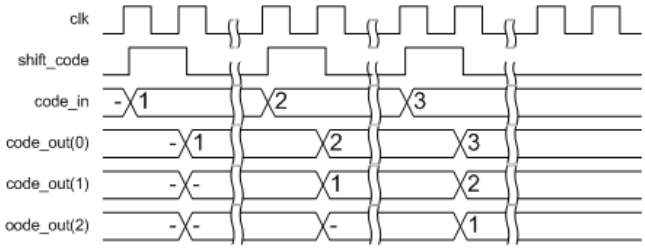
Det er to mulige utfall av dette:

- Dersom de 3 sifrene er lik koden som ble tastet inn går signalet *open_door* aktivt en klokkeperiode og aktiverer en mekanisme som låser opp døren. Etter dette går tilstandsmaskinen tilbake til oppstarttilstanden og er klar til å ta i mot en ny 3-siffers kode.
- Dersom de 3 inntastede sifrene ikke stemte overens med koden går signalet *error* aktivt og tilstandsmaskinen er klar til å motta 3 nye sifre. *error* skal være aktivt inntil riktig kode er tastet inn. Dersom ikke rett kombinasjon er tastet inn etter 10 forsøk går tilstandsmaskinen i en vranglåstilstand og både *alarm* og *error* er aktive i denne tilstanden.

Eneste måten å komme ut av vranglåstilstanden er å aktivere reset. Dette kan bare gjøres av servicepersonale (vaktmester) ved å låse opp safedøren manuelt med nøkkel og aktivere *reset* ved å trykke på en resetknapp inni i selve safen.

a) vekt 10%
Den 3-siffers 4-bits koden som tastes inn kan lagres i et skiftregister styrt av signalet *shift_code*. Når *shift_code* er aktivt shiftes mot høyre (se figuren under).





Implementer et slikt skiftregister med entitet *code_sreg* i VHDL og ved å benytte en process. Shiftregisteret skal kunne resettes asynkront med signalet *reset*.

Du skal ta utgangspunkt i typedefinisjonen *code_type* nedenfor for å lagre koden:

```
type code_type is array (0 to 2) of unsigned(3 downto 0);
```

Vi tenker oss *code_type* er definert i pakken *code_lock_pkg*.

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

entity code_sreg is
  port
  (
    reset      : in std_logic;
    clk        : in std_logic;
    shift_code : in std_logic;
    code_in    : in std_logic_vector(3 downto 0);
    code_out   : out code_type
  );
end ;

architecture rtl of code_sreg is

--Start å skrive VHDL kode her
..
..
--Slutt å skrive VHDL kode her

end architecture rtl;
```

b) vekt 15%
Lag et ASM flytskjema som beskriver kodelåstilstandsmaskinen. Det skal lages en Mealy maskin. Du må selv definere eventuelle interne signaler til hjelp til å realisere kodelåstilstandsmaskinen.

c) vekt 15%
Implementer tilstandsmaskinen i b) som en to-process tilstandsmaskin i VHDL.

d) vekt 10%
Lag en testbenk i VHDL som tester entiteten *code_lock*.
Merk at denne oppgaven kan løses uavhengig av de andre deloppgavene.

INF3430. Oppgavesvar for kandidat nr: _____

Oppgave	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF3430 Digital systemkonstruksjon**
Eksamensdag:	**7. desember 2011**
Tid for eksamen:	**9-13**
Oppgavesettet er på 10 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Alle trykte og skrevne hjelpemidler tillatt, samt kalkulator.**

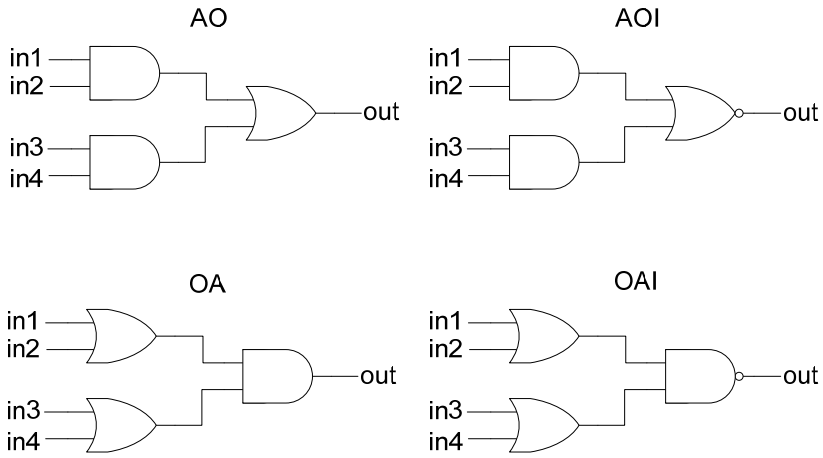
Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1–6 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 7-11 som besvares på vanlige ark. Oppgave 1-6 har til sammen vekt på 25%, mens oppgave 7-10 har til sammen vekt på 25% og oppgave 11 har til sammen vekt på 50%.

Generelt for oppgave 1-6:
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

Figuren under viser de kombinatoriske kretsene and-or (AO), and-or-invert (AOI), or-and (OA) og or-and-invert (OAI).



En 4-input Xilinx LUT med innhold **"0777"** (hex) realiserer en:	A	and-or (AO)	
	B	and-or-invert (AOI)	
	C	or-and (OA)	
	D	or-and-invert (OAI)	
	E	nor	

Oppgave 2

Design	A	Integrering av et helt system med prosessor på en krets gir en mer kompakt løsning som prismessig kan være gunstig.	
	B	En hard prosessorkjerne er vanligvis raskere (høyere klokkefrekvens) enn en myk prosessorkjerne.	
	C	Gigabit Transceivere finnes som myke kjerner.	
	D	ROM kan ikke lages av harde Block RAM (BRAM) moduler.	
	E	RAM kan lages av LUT'er.	

Oppgave 3

FPGA-teknologi	A	En SRAM FPGA er det ikke mulig å finne ut hvordan fungerer (dvs. ved "reverse-engineering") ut i fra konfigurasjonsfilen.	
	B	Block RAM'er har en kjent initialverdi etter konfigurering.	
	C	JTAG porten er eneste metode for å få konfigurert en Xilinx FPGA.	
	D	En antifuse FPGA kan reprogrammeres 1 gang.	
	E	En FPGA i master-modus styrer selv nedlastning av konfigurasjonen ved oppstart.	

Oppgave 4

Klokkenet, DCM og design	A	To klokkesignaler ut av Xilinx DCM modulen hvor en klokke er multiplisert 1.0 med original frekvens og den andre klokken er multiplisert 2.5 med original frekvens er i fase med hverandre.	
	B	DCM kan forsinke en generert klokke slik at den er i fase med inngangsklokken.	
	C	Antall nivåer med logikk i en FPGA *mellom* klokkede flipflop'er/registere har betydning for maksimal klokkefrekvensen.	
	D	Et differensielt ledningspar er mere følsomt for støy fra eksterne kilder enn en enkeltleder.	
	E	FPGA egner seg for pipelining pga. mange registere.	

Oppgave 5

Metastabilitet og timing constraints	A	Det er ikke enkelt å oppdage metastabilitet ved simulering.	
	B	Etter en tid i metastabil tilstand vil alle flip-flop'er alltid gå til verdien '0'.	
	C	Deaktivering av et eksternt reset signal må alltid synkroniseres for alle klokkedomener hvor reset brukes.	
	D	PERIOD timing constraint har høyere prioritet enn FROM-TO constraint.	
	E	OFFSET IN constraint kan bruke internt genererte klokker fra DCM.	

Oppgave 6

Design	A	En Xilinx Block RAM har to uavhengige porter som begge kan leses fra og skrives til samtidig.	
	B	Selv om antall input til en funksjon er konstant, øker forbruket av LUT'er alltid ved økning av kompleksiteten til funksjonen.	
	C	I Spartan 3 teknologi er vanligvis forsinkelsen gjennom en LUT lengre enn typisk forsinkelse mellom LUT'er.	
	D	Det er begrenset hvor mange adskilte klokkerlinjer fra BUFG'er som finnes i en FPGA i forhold til i en ASIC.	
	E	I en Xilinx FPGA har den asynkrone set inngangen til en flipflop/register høyere prioritet enn den asynkrone reset inngangen.	

Oppgave 7

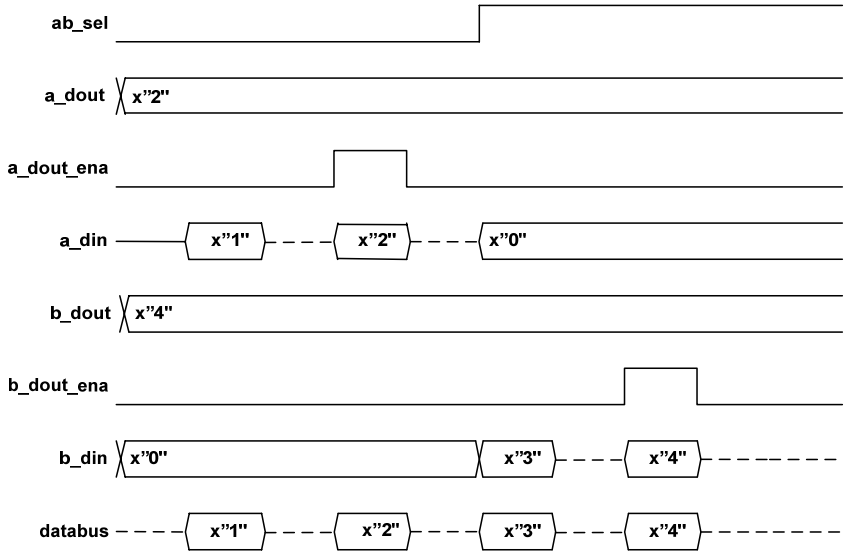
I VHDL koden oppgitt under blir telleren enablet når *ena*=’1’ og resetsignalet *rst*=’0’. Tegn opp et timingdiagram som viser verdiene signalene *mclk*, *cnt*, *cnt_eq3* og *cnt_eq7* i 12 klokkeperioder etter at *ena*=’1’ og *rst*=’0’. Bruk gjerne heltallsverdi i timingdiagrammet for signalet *cnt*.

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

entity counter is
  port (
    rst     : in  std_logic;
    mclk    : in  std_logic;
    ena     : in  std_logic;
    cnt_eq3 : out std_logic;
    cnt_eq7 : out std_logic;
    cnt     : out std_logic_vector(2 downto 0)
  );
end counter;

architecture rtl of counter is
    signal cnt_i : unsigned(2 downto 0);
begin
  process (rst, mclk) is
  begin
    if (rst = '1') then
      cnt_i   <= (others => '0');
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';
    elsif rising_edge(mclk) then
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';

      if ena='1' then
        cnt_i <= cnt_i + 1;
      end if;
      if cnt_i=3 then
        cnt_eq3 <= '1';
      end if;

      if cnt_i=7 then
        cnt_eq7 <= '1';
      end if;
    end if;
  end process;

  cnt <= std_logic_vector(cnt_i);
end rtl;
```

Oppgave 8

Modulen *busunit* gjengitt under multiplekser data inn og ut på den bidireksjonale databussen *databus* ved hjelp av three-state buffer og kontrollsignalet *ab_sel*. Et eksempel på bruk av modulen er vist i timingdiagrammet under. Skriv ferdig arkitekturen *rtl* i synthetiserbar VHDL kode så den får en funksjon som vist i eksemplet i timingdiagrammet under.

Timingdiagrammet viser at når *ab_sel*='0' settes verdien fra a *dout* ut på *databus* når *a_dout_ena*='1' og *databus* settes lik 'Z' når *a_dout_ena*='0'. Inngangen *a_din* settes til *databus* når *ab_sel*='0' og alle bit settes til '0' når *ab_sel*='1'. Tilsvarende velges *b_** signalene når *ab_sel*='1'. I figuren viser stiplete linje verdien 'Z'. I diagrammet er *databus* brukt som inngang for de eksterne inngangsverdiene x"1" og x"3" fra testbenk, og *databus* blir satt til 'Z' i testbenken ellers slik at *databus* kan få utgangsverdiene x"2" og x"4" fra *busunit*.




```
entity busunit is
  port (
    ab_sel     : in    std_logic;
    a_dout     : in    std_logic_vector(3 downto 0);
    a_dout_ena : in    std_logic;
    a_din      : out   std_logic_vector(3 downto 0);
    b_dout     : in    std_logic_vector(3 downto 0);
    b_dout_ena : in    std_logic;
    b_din      : out   std_logic_vector(3 downto 0);
    databus    : inout std_logic_vector(3 downto 0)
  );
end busunit;

architecture rtl of busunit is
begin
-- Start å skrive VHDL kode her:
  :
  :
-- Slutt å skrive VHDL kode her.
end rtl;
```

Oppgave 9

I denne oppgaven skal det lages et ASM-flytdiagram som utfører utregning av største felles divisor av to positive heltall *a_in* og *b_in*. Dette blir ofte omtalt som en "Greatest Common Divisor" (GCD) funksjon. For eksempel er gcd(1, 10) lik 1 og gcd(12,9) lik 3.

GCD algoritmen er en iterativ algoritme som kan uttrykkes som pseudokoden:

```
a = a_in;
b = b_in;
while (a /= b) do
  if (a > b) then
    a= a-b;
  else
    b= b-a;
  end if;
end do;
r= a;
```

GCD modulen skal ha entiteten:

```
entity gcd is
port(
  clk   : in  std_logic;
  rst   : in  std_logic;
  start : in  std_logic;
  a_in  : in  unsigned(7 downto 0);
  b_in  : in  unsigned(7 downto 0);
  ready : out std_logic;
  r     : out unsigned(7 downto 0)
);
```

Modulen begynner å beregne en GCD verdi når *start* ='1'. Statussignalet *ready* er lik '1' når modulen er klar til å starte en utregning, og '0' når utregning pågår.

Implementer pseudokoden til GCD modulen som en Moore type tilstandsmaskin (FSM) i et ASM-flytdiagram (det skal i denne deloppgaven IKKE implementeres i VHDL kode).

Oppgave 10

Under er det oppgitt entiteten og arkitekturen *rtl* til *something*. Gi en kort forklaring med ord hvilken funksjon modulen har og hva signalene *status1*, *status2* og *status3* viser?

```
entity something is
  port (
    clock    : in  std_logic;
    rst      : in  std_logic;
    din      : in  std_logic_vector(15 downto 0);
    ctrl1    : in  std_logic;
    ctrl2    : in  std_logic;
    dout     : out std_logic_vector(15 downto 0);
    status1  : out std_logic;
    status2  : out std_logic;
    status3  : out unsigned(3 downto 0)
  );
end something;
```

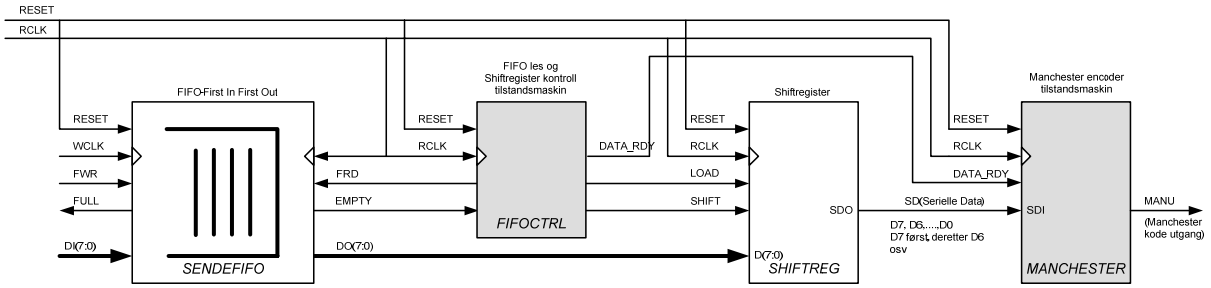
```
architecture rtl of something is

  type REGISTERS_TYPE is array(15 downto 0) of
                         std_logic_vector(15 downto 0);
  signal registers : REGISTERS_TYPE;
  signal status1_i : std_logic;
  signal status2_i : std_logic;
  signal addr_ctrl1: unsigned(3 downto 0);
  signal addr_ctrl2: unsigned(3 downto 0);

begin

  process (clock, rst) is
  begin
    if (rst = '1') then
      for i in 0 to 15 loop
        registers(i) <= (others => '0');
      end loop;
      addr_ctrl1<= (others => '0');
      addr_ctrl2<= (others => '0');
    elsif rising_edge(clock) then

      if (ctrl1 = '1' and status1_i = '0') then
        registers(to_integer(addr_ctrl1)) <= din;
        addr_ctrl1<= addr_ctrl1 + 1;
      end if;

      if (ctrl2 = '1' and status2_i = '0') then
        addr_ctrl2<= addr_ctrl2 + 1;
      end if;

    end if;
  end process;

  process (addr_ctrl1, addr_ctrl2)
  begin
    if (addr_ctrl2 = addr_ctrl1) then
      status2_i<= '1';
    else
      status2_i<= '0';
    end if;

    if (addr_ctrl1 + 1 = addr_ctrl2) then
      status1_i<= '1';
    else
      status1_i<= '0';
    end if;
  end process;

  status1<= status1_i;
  status2<= status2_i;
  status3<= addr_ctrl1 - addr_ctrl2;

  dout<= registers(to_integer(addr_ctrl2));

end architecture rtl;
```

Oppgave 11

Vi skal i denne oppgaven se på sendedelen, transmitdelen, av et tenkt datakommunikasjons-system. Vi skal implementere deler av det (merket med gråskraverte bokser i figuren under).

Systemet består av følgende byggeklosser:

- En sendeFIFO (First In First Out)
- En tilstandsmaskin, FIFOkontroller, for å lese data fra FIFO og skrive inn i et skiftregister.
- Et skiftregister
- En Manchester encoder tilstandsmaskin som gjør om data på serieform til Manchester kode.

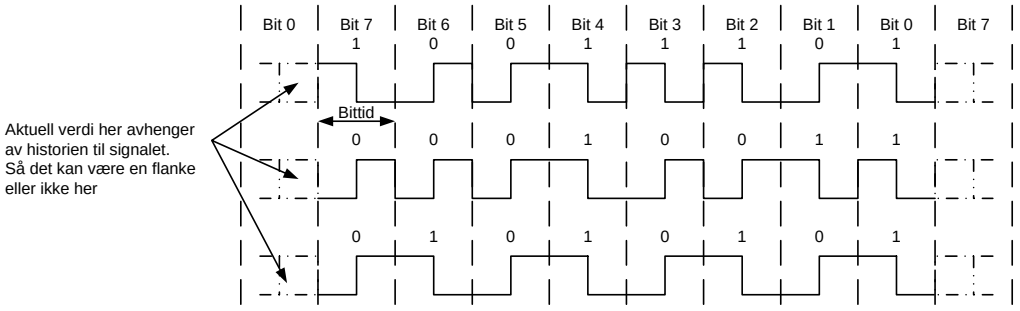


SendeFIFO:
Dette er en synkron FIFO, dvs. har egne klokkeinnganger.
Data skrives inn fra den ene siden og leses ut fra den andre. Den har separat skriveklokke (*WCLK*) og leseklokke (*RCLK*). Det er vanlig å køe opp data med en rask skriveklokke, mens data kan leses ut med en langsommere leseklokke, gjerne tilpasset hastigheten på kommunikasjons-linjen (bitraten).
Skriveporten består av signalene *WCLK*, *FWR*, *FULL* og *DI(7:0)*. Data skrives inn i FIFOen via *DI(7:0)* på positiv flanke av *WCLK* når *FWR* er aktivt. *FULL* er en status utgang som går aktivt høyt når FIFOen er full og benyttes til flytkontroll for unngå å overskrive data i FIFOen.
Data leses ut av FIFOen via *DO(7:0)* kontrollert av positiv flanke av *RCLK* når *FRD* er aktivt. *EMPTY* er et statussignal som går aktivt høyt når FIFOen er tom. *EMPTY* benyttes til flytkontroll for å unngå å lese ut ugyldige data når FIFOen er tom.
I oppgavene nedenfor antar vi at FIFO blir skrivet til av en eller annen FIFO skrivekontroller som noen andre har ansvar for å lage.

FIFOkontroller *FIFOCTRL*:
FIFOCTRL er en tilstandsmaskin som kontrollerer utlesningen fra FIFOen og skal virke på følgende måte:
EMPTY sjekkes og når det går inaktivt går *FRD* aktivt og en byte leses ut av FIFOen.
I perioden etter settes *LOAD* aktivt og *DO* klokkes inn i skiftregisteret via inngangene *D* til internt register *DQ* . Databit DQ(7)skal gå rett ut på den serielle datalinjen ut av skiftregisteret, *SDO*.
SHIFT signalet kontrollerer skifting av bit mot høyre (*DQ(6)->DQ(7), DQ(5)->DQ(6)* osv) og skal være aktivt annenhver klokkeperiode. Utgangssignalet *DATA_RDY* er aktivt så lenge det er gyldige data på *SDO*.

Skiftregisteret *SHIFTREG*:
Skiftregisteret er et et vanlig skiftregister med parallel synkron load, styrt av signalet *LOAD*.
Data skiftes ut på *SDO* når *SHIFT* går aktivt. Databit *DQ(7)* skal gå rett ut på *SDO* etter *LOAD*.

Manchester encoder tilstandsmaskinen *MANCHESTER*:
Funksjonen til Manchester encoderen er å kode de serielle dataene fra skiftregisteret som Manchester kode.
Manchester kode er en meget utbredt kodeteknikk i datakommunikasjon.
Det som er karakteristisk for Manchester koden er at vi alltid har en flanke midt i et bit.
En positiv flanke betyr at bitet er 0, mens en negativ flanke betyr 1.
Har vi flere 0'er eller 1'ere etter hverandre må det skje en transisjon i starten av bitet slik at vi kan få riktig flanke midt i bitet.
Nedenfor er noen eksempler på Manchester kodesekvenser (timingdiagram) for byter med verdiene 9Dh, 13h og 55h.



a) Vekt 5%
Tegn Manchester kodesekvensen(timingdiagram) til bytene AAh, 3Ch og 58h.

b) Vekt 10%
Lag et ASM flytskjema som beskriver Manchester encoder tilstandsmaskinen, *MANCHESTER*.
Den skal virke i henhold til nedre del av timingdiagrammet på neste side merket "*MANCHESTER* Deloppgave b) og c)". Legg merke til at de serielle data bit'ene *SDI* endrer seg annenhver klokkeperiode vist som *D7*, *D6*, osv i figuren.

Du må selv definere eventuelle interne signaler til hjelp for å realisere tilstandsmaskinen.
MANCHESTER skal ha følgende entitet og realiseres som en Mealy maskin:

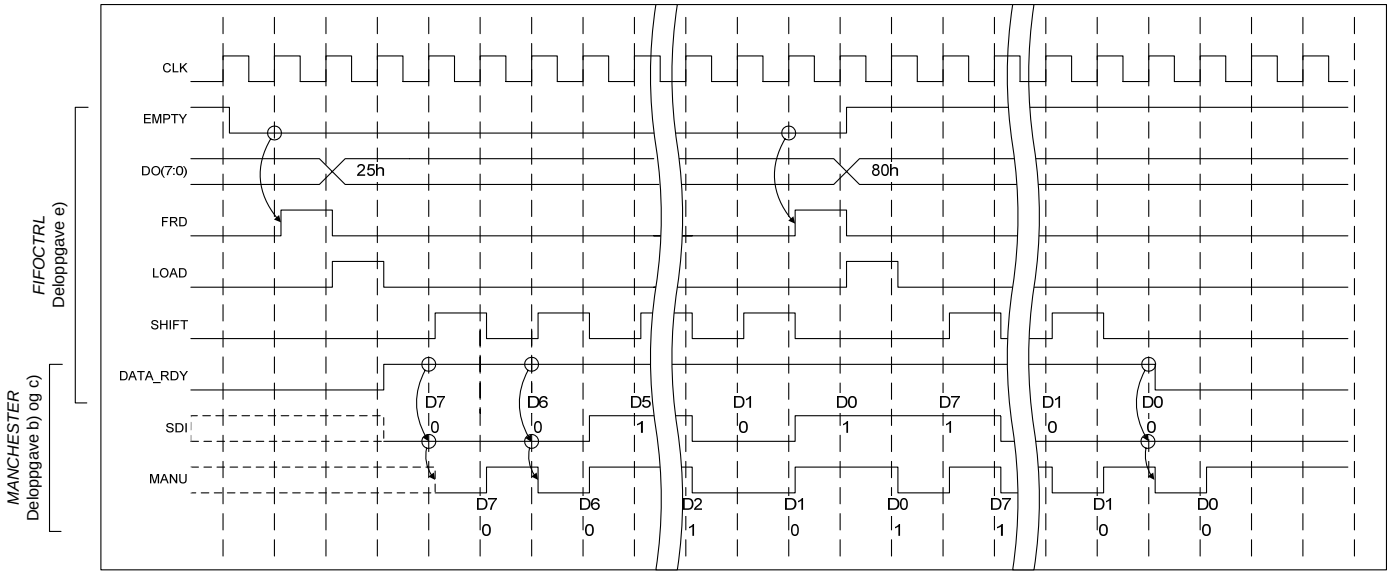
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MANCHESTER is
  port
  (
    RESET     : in std_logic;  --Asynkron reset
    CLK       : in std_logic;  --Klokke
    DATA_RDY  : in std_logic;  --Viser gyldige data inn
    SDI       : in std_logic;  --Serielle input data
    MANU      : out std_logic; --Manchester kodet sekvens ut
  );
end ;
```

c) Vekt 10%
Implementer tilstandsmaskinen i b) som en to-process tilstandsmaskin i VHDL.

d) Vekt 15%
Lag en testbenk i VHDL som tester entiteten *MANCHESTER.*
Merk at denne oppgaven kan løses uavhengig av de andre deloppgavene.

e) Vekt 10%
Lag et ASM flytskjema som beskriver *FIFOCTRL* tilstandsmaskinen. *FIFOCTRL* skal virke i henhold til øvre del av timingdiagrammet nederst på denne siden og merket "*FIFOCTRL* Deloppgave e)".
FIFOCTRL skal ha følgende entitet og skal realiseres som en Mealy maskin. Det skal IKKE lages VHDL kode.

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

entity FIFOCTRL is
  port
  (
    RESET     : in std_logic;  --Asynkron reset
    CLK       : in std_logic;  --Klokke
    EMPTY     : in std_logic;  --Fra FIFO og indikerer at FIFO er tom
    FRD       : out std_logic; --Lesestrobe til FIFO
    LOAD      : out std_logic; --LOAD signal til shiftregister
    SHIFT     : out std_logic; --SHIFT signal til shiftregister
    DATA_RDY  : out std_logic  --Indikerer gyldige serielle data til
                               --Manchester encoder

  );
end ;
```



Timingdiagram for *FIFOCTRL* og *MANCHESTER*

INF3430. Oppgavesvar for kandidat nr: _____

Oppgave	A	B	C	D	E
1					
2					
3					
4					
5					
6					

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	**INF4431 Digital systemkonstruksjon**
Eksamensdag:	**7. desember 2011**
Tid for eksamen:	**9-13**
Oppgavesettet er på 11 sider	
Vedlegg:	**1**
Tillatte hjelpemidler:	**Alle trykte og skrevne hjelpemidler tillatt, samt kalkulator.**

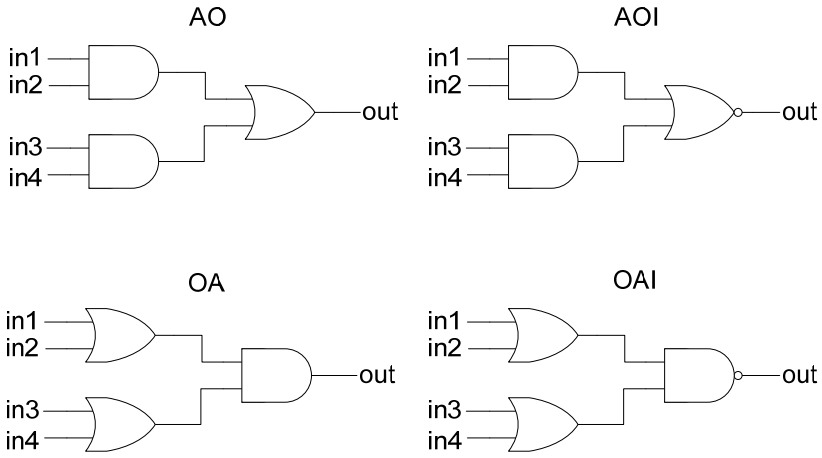
Kontroller at oppgavesettet er komplett
før du begynner å besvare spørsmålene.

Oppgaveteksten består av oppgave 1–5 (flervalgsoppgaver) som skal besvares på skjemaet som er vedlagt etter oppgaveteksten og oppgave 6-11 som besvares på vanlige ark. Oppgave 1-6 har til sammen vekt på 25%, mens oppgave 7-10 har til sammen vekt på 25% og oppgave 11 har til sammen vekt på 50.

Generelt for oppgave 1-5:
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

Figuren under viser de kombinatoriske kretsene and-or (AO), and-or-invert (AOI), or-and (OA) og or-and-invert (OAI).



En 4-input Xilinx LUT med innhold **”0777”** (hex) realiserer en:	A	and-or (AO)	
	B	and-or-invert (AOI)	
	C	or-and (OA)	
	D	or-and-invert (OAI)	
	E	nor	

Oppgave 2

Design	A	Integrering av et helt system med prosessor på en krets gir en mer kompakt løsning som prismessig kan være gunstig.	
	B	En hard prosessorkjerne er vanligvis raskere (høyere klokkefrekvens) enn en myk prosessorkjerne.	
	C	Gigabit Transceivere finnes som myke kjerner.	
	D	ROM kan ikke lages av harde Block RAM (BRAM) moduler.	
	E	RAM kan lages av LUT’er.	

Oppgave 3

FPGA-teknologi	A	En SRAM FPGA er det ikke mulig å finne ut hvordan fungerer (dvs. ved ”reverse-engineering”) ut i fra konfigurasjonsfilen.	
	B	Block RAM’er har en kjent initialverdi etter konfigurering.	
	C	JTAG porten er eneste metode for å få konfigurert en Xilinx FPGA.	
	D	En antifuse FPGA kan reprogrammeres 1 gang.	
	E	En FPGA i master-modus styrer selv nedlastning av konfigurasjonen ved oppstart.	

Oppgave 4

Klokkenet, DCM og design	A	To klokkesignaler ut av Xilinx DCM modulen hvor en klokke er multiplisert 1.0 med original frekvens og den andre klokken er multiplisert 2.5 med original frekvens er i fase med hverandre.	
	B	DCM kan forsinke en generert klokke slik at den er i fase med inngangsklokken.	
	C	Antall nivåer med logikk i en FPGA *mellom* klokkede flipflop'er/registere har betydning for maksimal klokkefrekvensen.	
	D	Et differensielt ledningspar er mere følsomt for støy fra eksterne kilder enn en enkeltleder.	
	E	FPGA egner seg for pipelining pga. mange registere.	

Oppgave 5

Metastabilitet og timing constraints	A	Det er ikke enkelt å oppdage metastabilitet ved simulering.	
	B	Etter en tid i metastabil tilstand vil alle flip-flop'er alltid gå til verdien '0'.	
	C	Deaktivering av et eksternt reset signal må alltid synkroniseres for alle klokkedomener hvor reset brukes.	
	D	PERIOD timing constraint har høyere prioritet enn FROM-TO constraint.	
	E	OFFSET IN constraint kan bruke internt genererte klokker fra DCM.	

Oppgave 6

I VHDL koden oppgitt under blir telleren enablet når *ena*=’1’ og resetsignalet *rst*=’0’. Tegn opp et timingdiagram som viser verdiene signalene *mclk*, *cnt*, *cnt_eq3* og *cnt_eq7* i 12 klokkeperioder etter at *ena*=’1’ og *rst*=’0’. Bruk gjerne heltallsverdi i timingdiagrammet for signalet *cnt*.

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

entity counter is
  port (
    rst     : in  std_logic;
    mclk    : in  std_logic;
    ena     : in  std_logic;
    cnt_eq3 : out std_logic;
    cnt_eq7 : out std_logic;
    cnt     : out std_logic_vector(2 downto 0)
  );
end counter;

architecture rtl of counter is
    signal cnt_i : unsigned(2 downto 0);
begin
  process (rst, mclk) is
  begin
    if (rst = '1') then
      cnt_i   <= (others => '0');
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';
    elsif rising_edge(mclk) then
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';

      if ena='1' then
        cnt_i <= cnt_i + 1;
      end if;
      if cnt_i=3 then
        cnt_eq3 <= '1';
      end if;

      if cnt_i=7 then
        cnt_eq7 <= '1';
      end if;
    end if;
  end process;

  cnt <= std_logic_vector(cnt_i);
end rtl;
```

Oppgave 7

Koden oppgitt i deloppgave 6 som er gjengitt under, skal i denne deloppgaven endres fra VHDL til SystemVerilog med samme funksjon og timing som den oppgitte VHDL koden. Den nye uferdige modulen *counter* er oppgitt under. Skriv ferdig modulen i SystemVerilog kode.

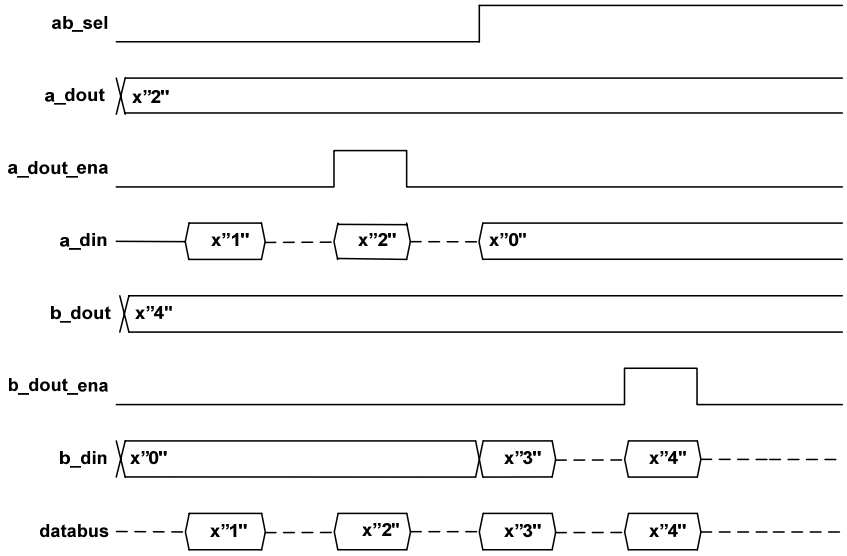
```
entity counter is
  port (
    rst     : in  std_logic;
    mclk    : in  std_logic;
    ena     : in  std_logic;
    cnt_eq3 : out std_logic;
    cnt_eq7 : out std_logic;
    cnt     : out std_logic_vector(2 downto 0)
  );
end counter;

architecture rtl of counter is
    signal cnt_i : unsigned(2 downto 0);
begin
  process (rst, mclk) is
  begin
    if (rst = '1') then
      cnt_i   <= (others => '0');
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';
    elsif rising_edge(mclk) then
      cnt_eq3 <= '0';
      cnt_eq7 <= '0';

      if ena='1' then
        cnt_i <= cnt_i + 1;
      end if;
      if cnt_i=3 then
        cnt_eq3 <= '1';
      end if;

      if cnt_i=7 then
        cnt_eq7 <= '1';
      end if;
    end if;
  end process;

  cnt <= std_logic_vector(cnt_i);
end rtl;
```

```
module counter (
// Start å skrive SystemVerilog kode her:
:
:
:
// Slutt å skrive SystemVerilog kode her.
Endmodule
```

Oppgave 8

Modulen *busunit* gjengitt under multiplekser data inn og ut på den bidireksjonale databussen *databus* ved hjelp av three-state buffer og kontrollsignalet *ab_sel*. Et eksempel på bruk av modulen er vist i timingdiagrammet under. Skriv ferdig arkitekturen *rtl* i synthetiserbar VHDL kode så den får en funksjon som vist i eksemplet i timingdiagrammet under.

Timingdiagrammet viser at når *ab_sel*='0' settes verdien fra a *dout* ut på *databus* når *a_dout_ena*='1' og *databus* settes lik 'Z' når *a_dout_ena*='0'. Inngangen *a_din* settes til *databus* når *ab_sel*='0' og alle bit settes til '0' når *ab_sel*='1'. Tilsvarende velges *b_** signalene når *ab_sel*='1'. I figuren viser stiplete linje verdien 'Z'. I diagrammet er *databus* brukt som inngang for de eksterne inngangsverdiene x"1" og x"3" fra testbenk, og *databus* blir satt til 'Z' i testbenken ellers slik at *databus* kan få utgangsverdiene x"2" og x"4" fra *busunit*.




```
entity busunit is
  port (
    ab_sel     : in    std_logic;
    a_dout     : in    std_logic_vector(3 downto 0);
    a_dout_ena : in    std_logic;
    a_din      : out   std_logic_vector(3 downto 0);
    b_dout     : in    std_logic_vector(3 downto 0);
    b_dout_ena : in    std_logic;
    b_din      : out   std_logic_vector(3 downto 0);
    databus    : inout std_logic_vector(3 downto 0)
  );
end busunit;

architecture rtl of busunit is
begin
-- Start å skrive VHDL kode her:
  :
  :
-- Slutt å skrive VHDL kode her.
end rtl;
```

Oppgave 9

I denne oppgaven skal det lages et ASM-flytdiagram som utfører utregning av største felles divisor av to positive heltall *a_in* og *b_in*. Dette blir ofte omtalt som en "Greatest Common Divisor" (GCD) funksjon. For eksempel er gcd(1, 10) lik 1 og gcd(12,9) lik 3.

GCD algoritmen er en iterativ algoritme som kan uttrykkes som pseudokoden:

```
a = a_in;
b = b_in;
while (a /= b) do
  if (a > b) then
    a= a-b;
  else
    b= b-a;
  end if;
end do;
r= a;
```

GCD modulen skal ha entiteten:

```
entity gcd is
port(
  clk   : in  std_logic;
  rst   : in  std_logic;
  start : in  std_logic;
  a_in  : in  unsigned(7 downto 0);
  b_in  : in  unsigned(7 downto 0);
  ready : out std_logic;
  r     : out unsigned(7 downto 0)
);
```

Modulen begynner å beregne en GCD verdi når *start* ='1'. Statussignalet *ready* er lik '1' når modulen er klar til å starte en utregning, og '0' når utregning pågår.

Implementer pseudokoden til GCD modulen som en Moore type tilstandsmaskin (FSM) i et ASM-flytdiagram (det skal i denne deloppgaven IKKE implementeres i VHDL kode).

Oppgave 10

Under er det oppgitt entiteten og arkitekturen *rtl* til *something*. Gi en kort forklaring med ord hvilken funksjon modulen har og hva signalene *status1*, *status2* og *status3* viser?

```
entity something is
  port (
    clock    : in  std_logic;
    rst      : in  std_logic;
    din      : in  std_logic_vector(15 downto 0);
    ctrl1    : in  std_logic;
    ctrl2    : in  std_logic;
    dout     : out std_logic_vector(15 downto 0);
    status1  : out std_logic;
    status2  : out std_logic;
    status3  : out unsigned(3 downto 0)
  );
end something;
```

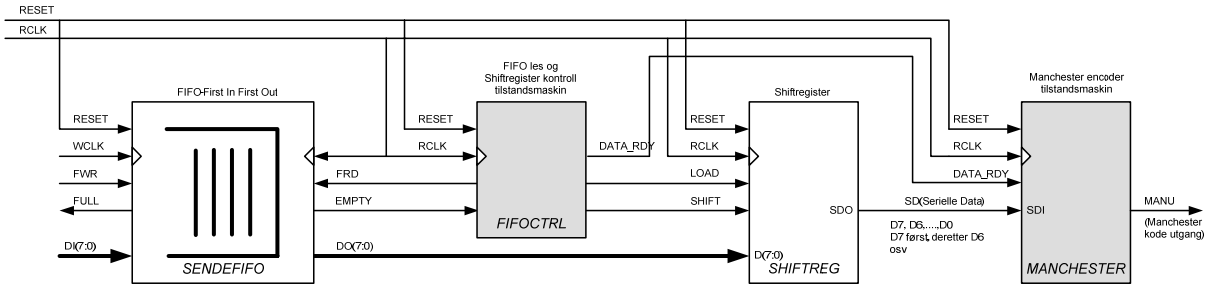
```
architecture rtl of something is

  type REGISTERS_TYPE is array(15 downto 0) of
                       std_logic_vector(15 downto 0);
  signal registers : REGISTERS_TYPE;
  signal status1_i : std_logic;
  signal status2_i : std_logic;
  signal addr_ctrl1: unsigned(3 downto 0);
  signal addr_ctrl2: unsigned(3 downto 0);

begin

  process (clock, rst) is
  begin
    if (rst = '1') then
      for i in 0 to 15 loop
        registers(i) <= (others => '0');
      end loop;
      addr_ctrl1<= (others => '0');
      addr_ctrl2<= (others => '0');
    elsif rising_edge(clock) then

      if (ctrl1 = '1' and status1_i = '0') then
        registers(to_integer(addr_ctrl1)) <= din;
        addr_ctrl1<= addr_ctrl1 + 1;
      end if;

      if (ctrl2 = '1' and status2_i = '0') then
        addr_ctrl2<= addr_ctrl2 + 1;
      end if;

    end if;
  end process;

  process (addr_ctrl1, addr_ctrl2)
  begin
    if (addr_ctrl2 = addr_ctrl1) then
      status2_i<= '1';
    else
      status2_i<= '0';
    end if;

    if (addr_ctrl1 + 1 = addr_ctrl2) then
      status1_i<= '1';
    else
      status1_i<= '0';
    end if;
  end process;

  status1<= status1_i;
  status2<= status2_i;
  status3<= addr_ctrl1 - addr_ctrl2;

  dout<= registers(to_integer(addr_ctrl2));

end architecture rtl;
```

Oppgave 11

Vi skal i denne oppgaven se på sendedelen, transmitdelen, av et tenkt datakommunikasjons-system. Vi skal implementere deler av det (merket med gråskraverte bokser i figuren under).

Systemet består av følgende byggeklosser:

- En sendeFIFO (First In First Out)
- En tilstandsmaskin, FIFOkontroller, for å lese data fra FIFO og skrive inn i et skiftregister.
- Et skiftregister
- En Manchester encoder tilstandsmaskin som gjør om data på serieform til Manchester kode.

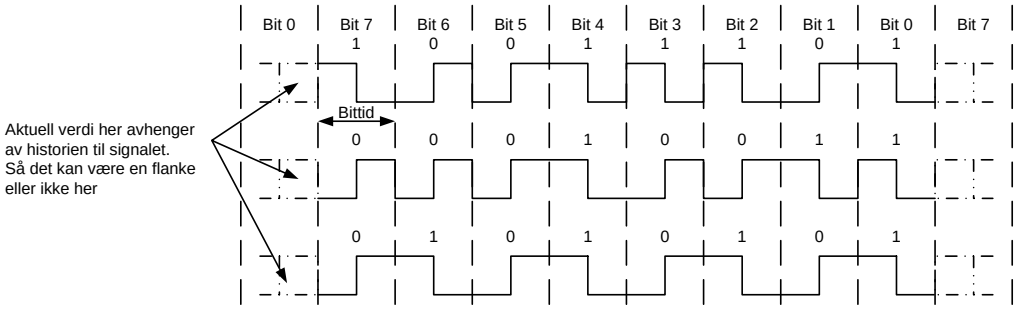


SendeFIFO:
Dette er en synkron FIFO, dvs. har egne klokkeinnganger.
Data skrives inn fra den ene siden og leses ut fra den andre. Den har separat skriveklokke (*WCLK*) og leseklokke (*RCLK*). Det er vanlig å køe opp data med en rask skriveklokke, mens data kan leses ut med en langsommere leseklokke, gjerne tilpasset hastigheten på kommunikasjons-linjen (bitraten).
Skriveporten består av signalene *WCLK*, *FWR*, *FULL* og *DI(7:0)*. Data skrives inn i FIFOen via *DI(7:0)* på positiv flanke av *WCLK* når *FWR* er aktivt. *FULL* er en status utgang som går aktivt høyt når FIFOen er full og benyttes til flytkontroll for unngå å overskrive data i FIFOen.
Data leses ut av FIFOen via *DO(7:0)* kontrollert av positiv flanke av *RCLK* når *FRD* er aktivt. *EMPTY* er et statussignal som går aktivt høyt når FIFOen er tom. *EMPTY* benyttes til flytkontroll for å unngå å lese ut ugyldige data når FIFOen er tom.
I oppgavene nedenfor antar vi at FIFO blir skrivet til av en eller annen FIFO skrivekontroller som noen andre har ansvar for å lage.

FIFOkontroller *FIFOCTRL*:
FIFOCTRL er en tilstandsmaskin som kontrollerer utlesningen fra FIFOen og skal virke på følgende måte:
EMPTY sjekkes og når det går inaktivt går *FRD* aktivt og en byte leses ut av FIFOen.
I perioden etter settes *LOAD* aktivt og *DO* klokkes inn i skiftregisteret via inngangene *D* til internt register *DQ* . Databit DQ(7)skal gå rett ut på den serielle datalinjen ut av skiftregisteret, *SDO*.
SHIFT signalet kontrollerer skifting av bit mot høyre (*DQ(6)->DQ(7), DQ(5)->DQ(6)* osv) og skal være aktivt annenhver klokkeperiode. Utgangssignalet *DATA_RDY* er aktivt så lenge det er gyldige data på *SDO*.

Skiftregisteret *SHIFTREG*:
Skiftregisteret er et et vanlig skiftregister med parallel synkron load, styrt av signalet *LOAD*.
Data skiftes ut på *SDO* når *SHIFT* går aktivt. Databit *DQ(7)* skal gå rett ut på *SDO* etter *LOAD*.

Manchester encoder tilstandsmaskinen *MANCHESTER*:
Funksjonen til Manchester encoderen er å kode de serielle dataene fra skiftregisteret som Manchester kode.
Manchester kode er en meget utbredt kodeteknikk i datakommunikasjon.
Det som er karakteristisk for Manchester koden er at vi alltid har en flanke midt i et bit.
En positiv flanke betyr at bitet er 0, mens en negativ flanke betyr 1.
Har vi flere 0'er eller 1'ere etter hverandre må det skje en transisjon i starten av bitet slik at vi kan få riktig flanke midt i bitet.
Nedenfor er noen eksempler på Manchester kodesekvenser (timingdiagram) for byter med verdiene 9Dh, 13h og 55h.



a) Vekt 5%
Tegn Manchester kodesekvensen(timingdiagram) til bytene AAh, 3Ch og 58h.

b) Vekt 10%
Lag et ASM flytskjema som beskriver Manchester encoder tilstandsmaskinen, *MANCHESTER*.
Den skal virke i henhold til nedre del av timingdiagrammet på neste side merket "*MANCHESTER* Deloppgave b) og c)". Legg merke til at de serielle data bit'ene *SDI* endrer seg annenhver klokkeperiode vist som *D7*, *D6*, osv i figuren.

Du må selv definere eventuelle interne signaler til hjelp for å realisere tilstandsmaskinen.
MANCHESTER skal ha følgende entitet og realiseres som en Mealy maskin:

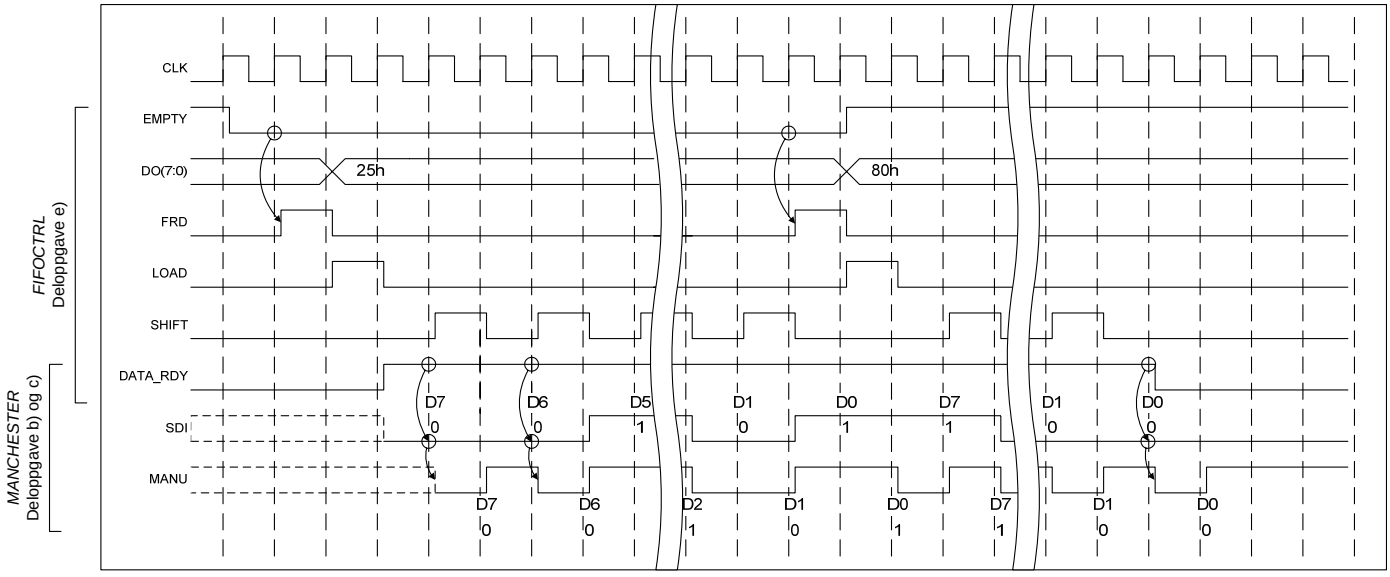
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MANCHESTER is
  port
  (
    RESET     : in std_logic;  --Asynkron reset
    CLK       : in std_logic;  --Klokke
    DATA_RDY  : in std_logic;  --Viser gyldige data inn
    SDI       : in std_logic;  --Serielle input data
    MANU      : out std_logic; --Manchester kodet sekvens ut
  );
end ;
```

c) Vekt 10%
Implementer tilstandsmaskinen i b) som en to-process tilstandsmaskin i VHDL.

d) Vekt 15%
Lag en testbenk i VHDL som tester entiteten *MANCHESTER.*
Merk at denne oppgaven kan løses uavhengig av de andre deloppgavene.

e) Vekt 10%
Lag et ASM flytskjema som beskriver *FIFOCTRL* tilstandsmaskinen. *FIFOCTRL* skal virke i henhold til øvre del av timingdiagrammet nederst på denne siden og merket "*FIFOCTRL* Deloppgave e)".
FIFOCTRL skal ha følgende entitet og skal realiseres som en Mealy maskin. Det skal IKKE lages VHDL kode.

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

entity FIFOCTRL is
  port
  (
    RESET     : in std_logic;  --Asynkron reset
    CLK       : in std_logic;  --Klokke
    EMPTY     : in std_logic;  --Fra FIFO og indikerer at FIFO er tom
    FRD       : out std_logic; --Lesestrobe til FIFO
    LOAD      : out std_logic; --LOAD signal til shiftregister
    SHIFT     : out std_logic; --SHIFT signal til shiftregister
    DATA_RDY  : out std_logic  --Indikerer gyldige serielle data til
                               --Manchester encoder

  );
end ;
```



Timingdiagram for *FIFOCTRL* og *MANCHESTER*

INF4431. Oppgavesvar for kandidat nr: _____

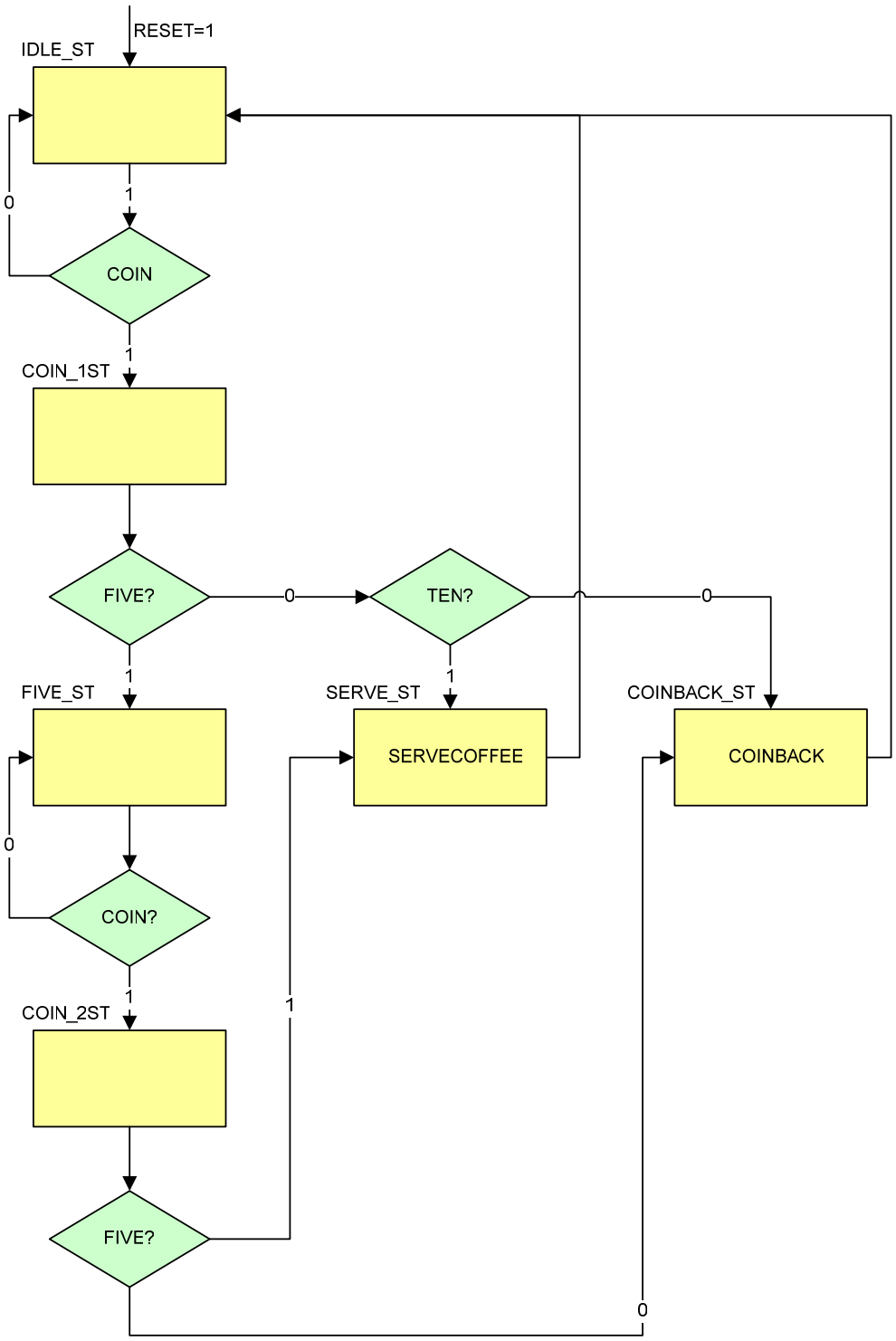
Oppgave	A	B	C	D	E
1					
2					
3					
4					
5					

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.

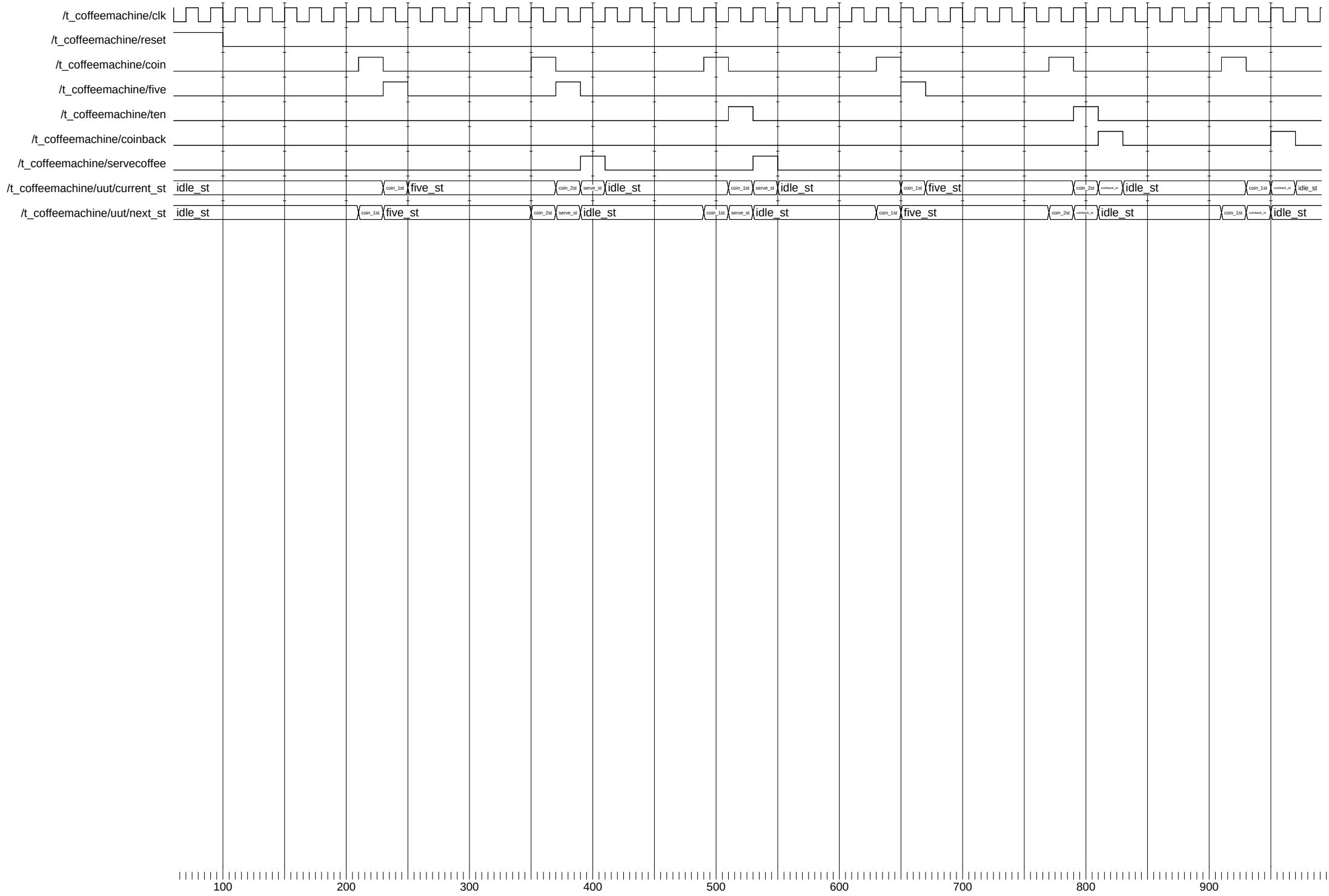
Et annet alternativ er å hente input stimuli fra fil.

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)

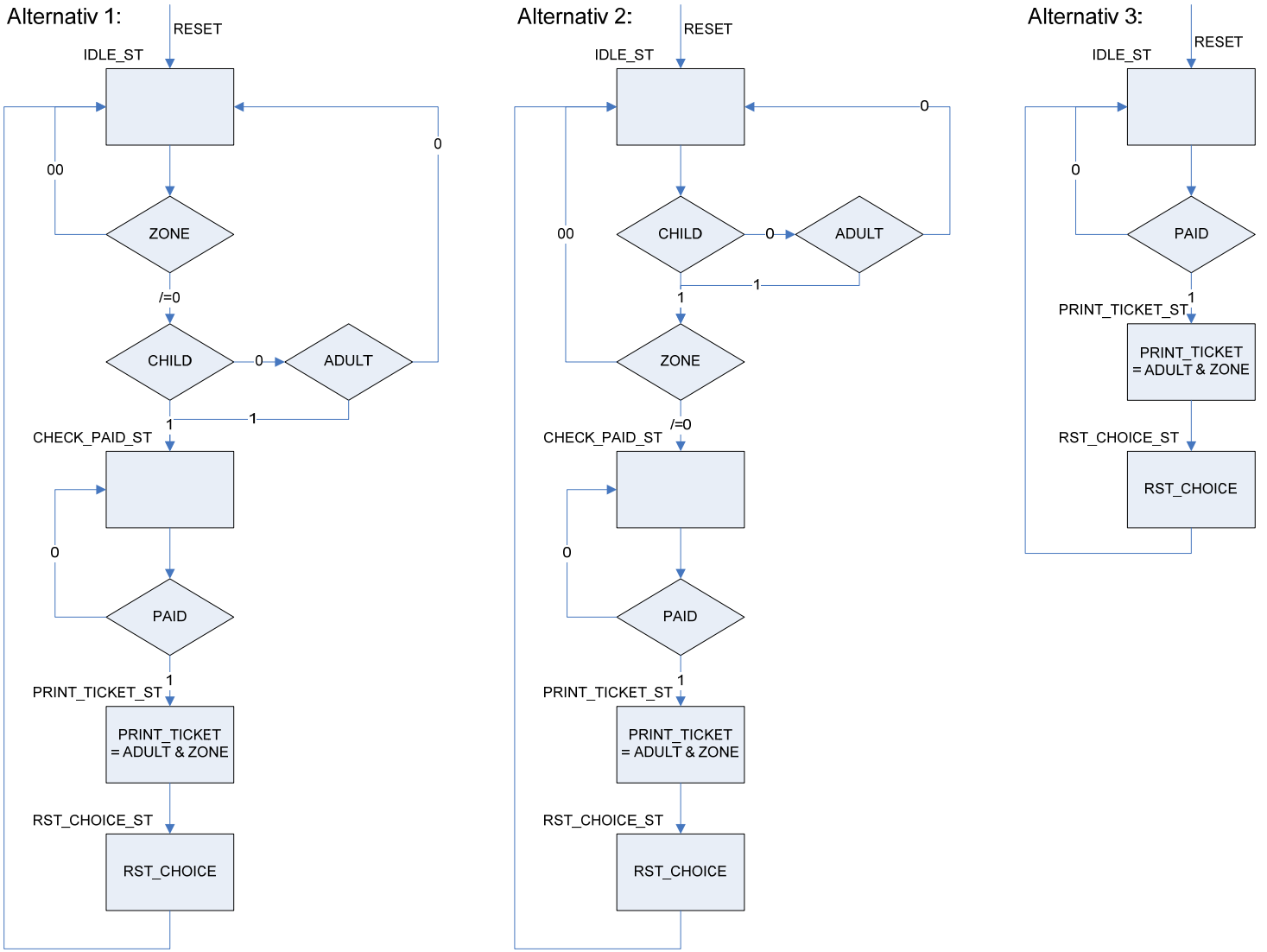
Tilstandsmaskinen implementert i oppgave 15b) er et eksempel på en Moore-maskin. I en Moore-maskin avhenger utgangene bare av nåværende tilstand i motsetning til en Mealy-maskin der utgangene avhenger av både nåværende tilstand og inngangene.

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).

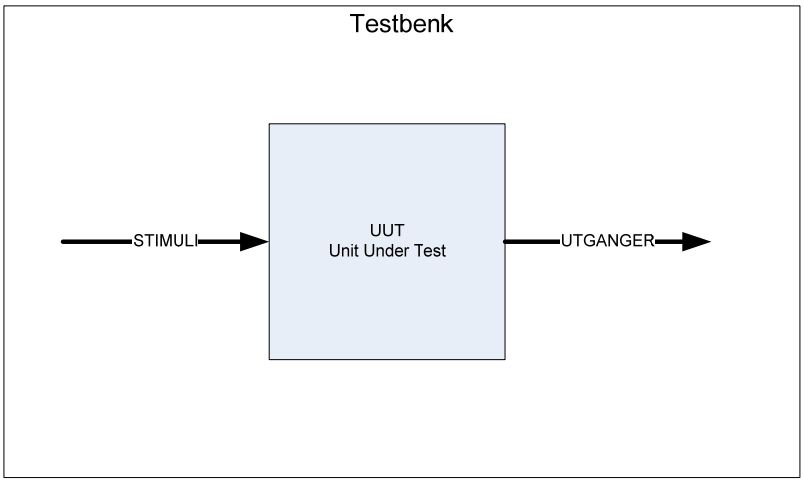

Alternativ 1:
RESET
IDLE_ST
00
ZONE
/=0
CHILD
0
ADULT
0
1
1
CHECK_PAID_ST
0
PAID
1
PRINT_TICKET_ST
PRINT_TICKET
= ADULT & ZONE
RST_CHOICE_ST
RST_CHOICE
Alternativ 2:
RESET
IDLE_ST
0
00
CHILD
0
ADULT
1
1
ZONE
/=0
CHECK_PAID_ST
0
PAID
1
PRINT_TICKET_ST
PRINT_TICKET
= ADULT & ZONE
RST_CHOICE_ST
RST_CHOICE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

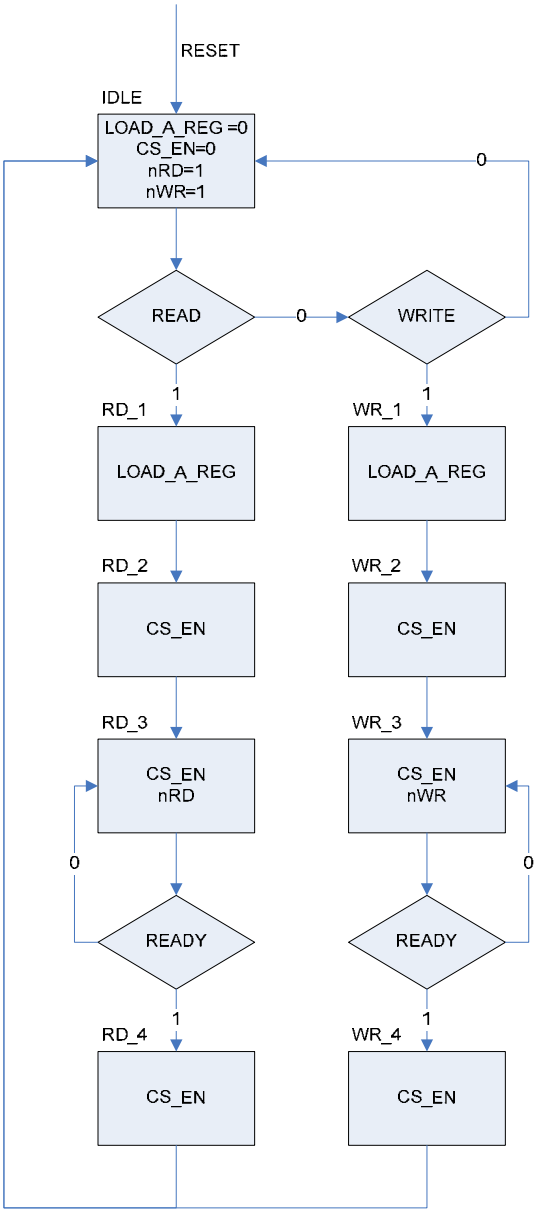
Oppgave	A	B	C	D	E
1	O		O	O	
2		O	O		
3	O	O		O	
4		O		O	
5			O		
6	O			O	
7		O	O		
8	O	O	O		
9		O		O	
10		O	O	O	
11	O			O	
12		O			
13	O	O	O		
14	O	O		O	

Oppgave 15.

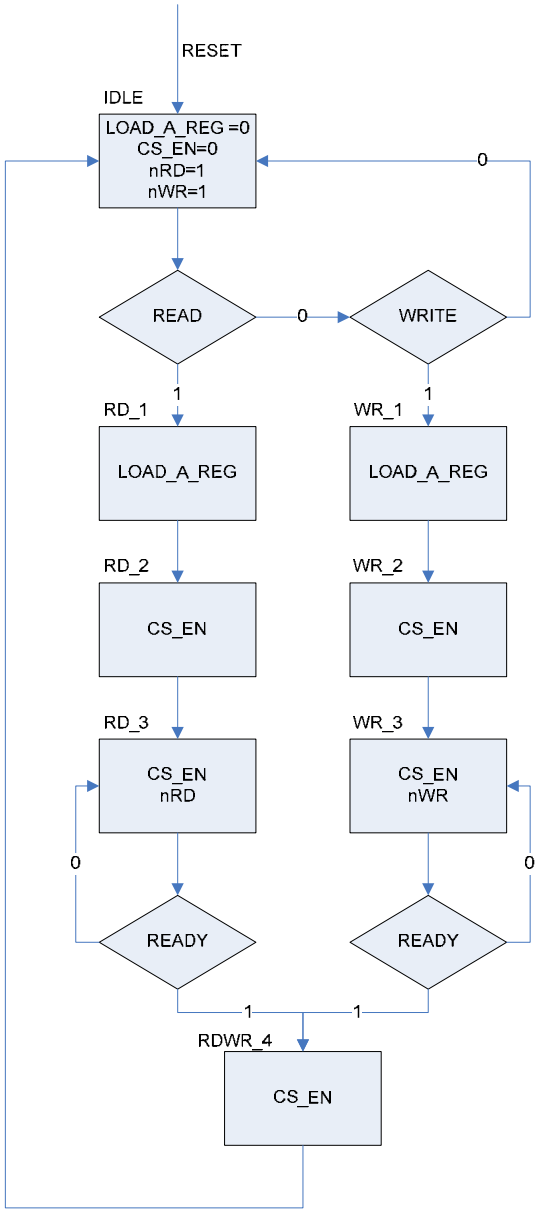
a).

```
--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).

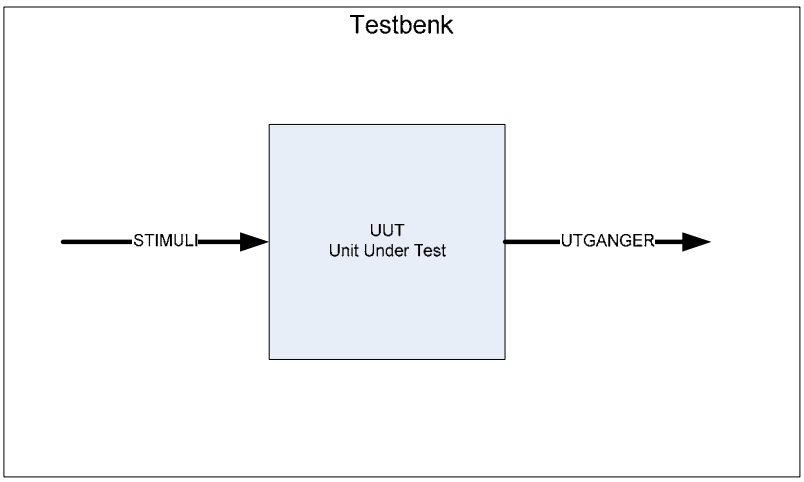
```
  type IO_CONTROLLER_TYPE is (IDLE, RD_1,RD_2,RD_3,RD_4,
                              WR_1,WR_2,WR_3,WR_4);
--eller
  --type IO_CONTROLLER_TYPE is (IDLE, RD_1,RD_2,RD_3,,
  --                            WR_1,WR_2,WR_3,RDWR_4);
  signal IO_CTRL_ST,IO_CTRL_NEXT_ST   : IO_CONTROLLER_TYPE;

  --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

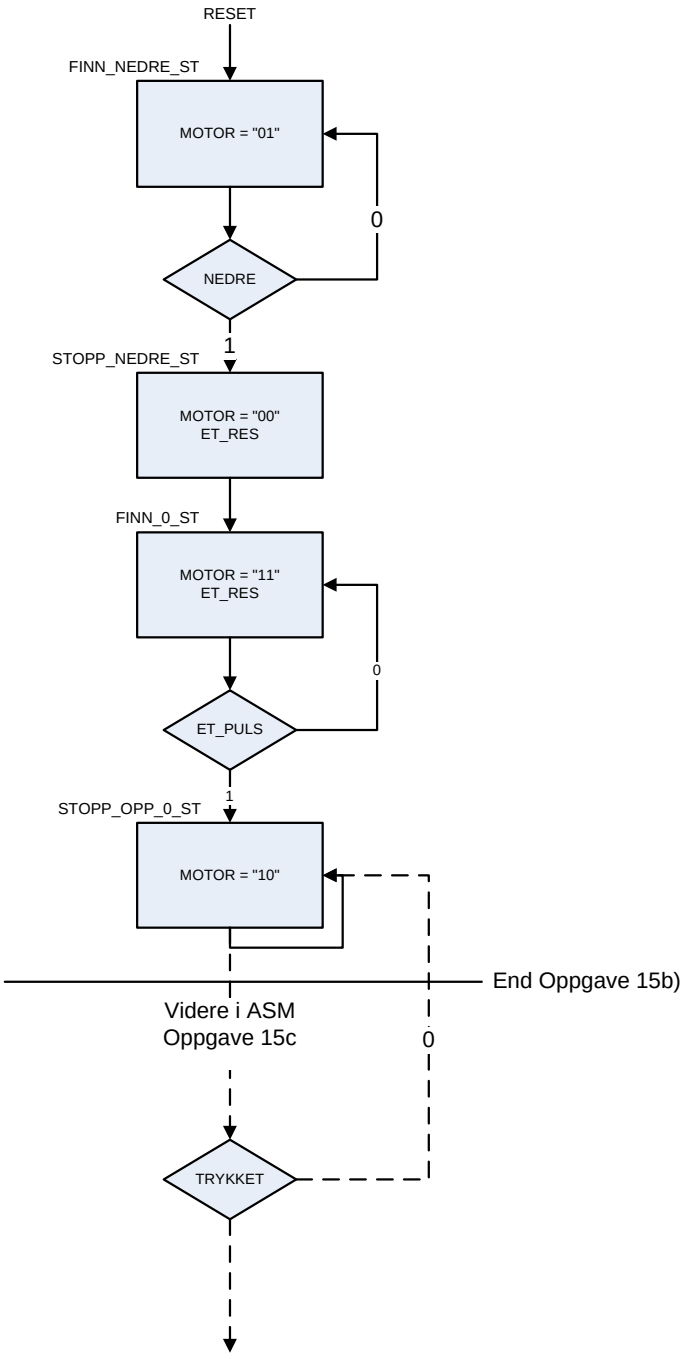
Oppgave	A	B	C	D
1		X		X
2		X		
3	X		X	X
4		X		X
5	X			
6			X	X
7	X	X		
8		X	X	X
9	X		X	X
10	X		X	
11	X			X
12	X	X		X
13		X		
14		X		

Oppgave 15

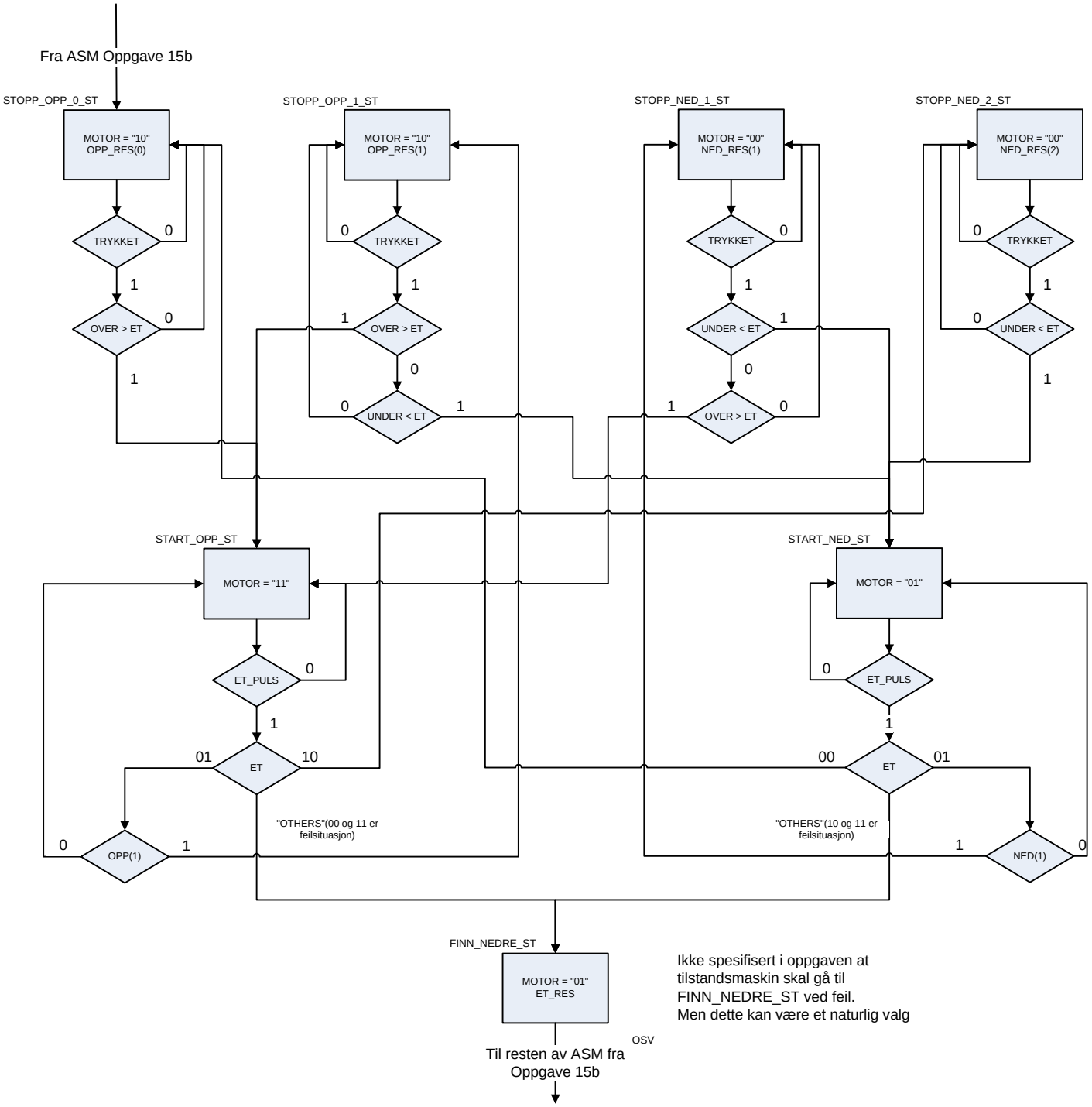
a)

```
-- 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)

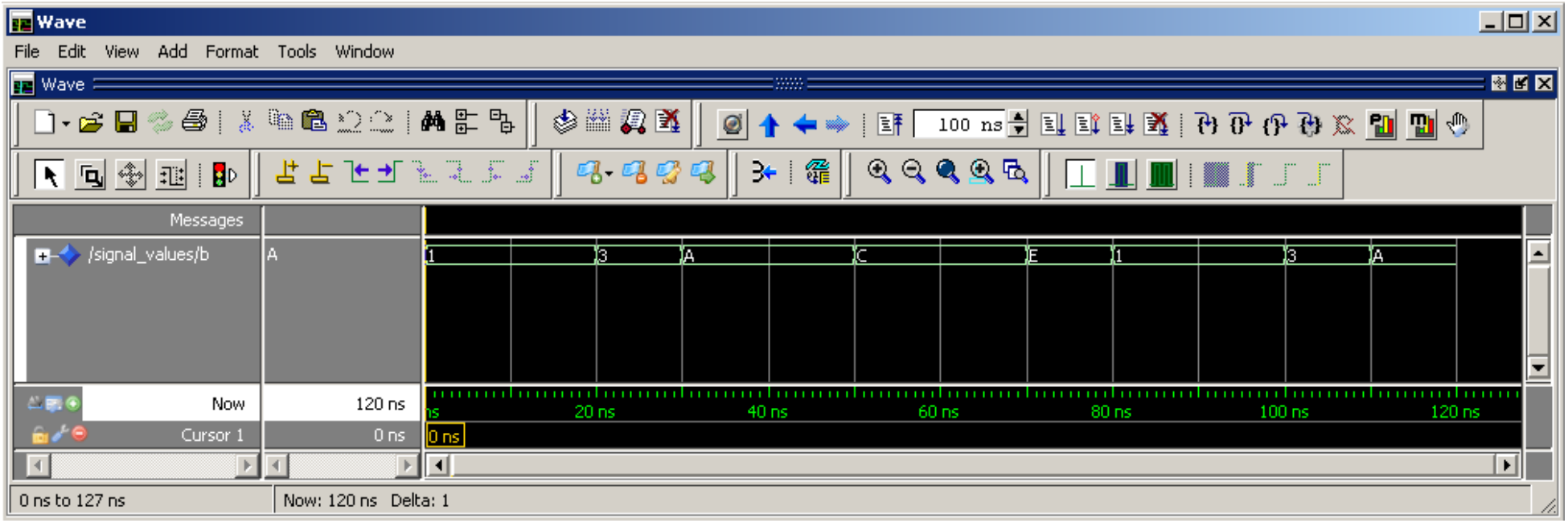
Man kan benytte f.eks. signalet `et_puls` til å sette en teller til en verdi som kan lage en passende pause . Telleren teller ned til 0. Når telleren er 0 er signalet `timeout` aktivt. `timeout` benyttes som en betingelse sammen med `trykket` og `over/under` for å lage en forsinket start.

INF3430. Fasit eksamen Høst 2009. Oppgave 1 – 6.

Oppgave	A	B	C	D	E
1	X		X		
2			X		
3			X	X	
4	X				X
5				X	X
6		X			

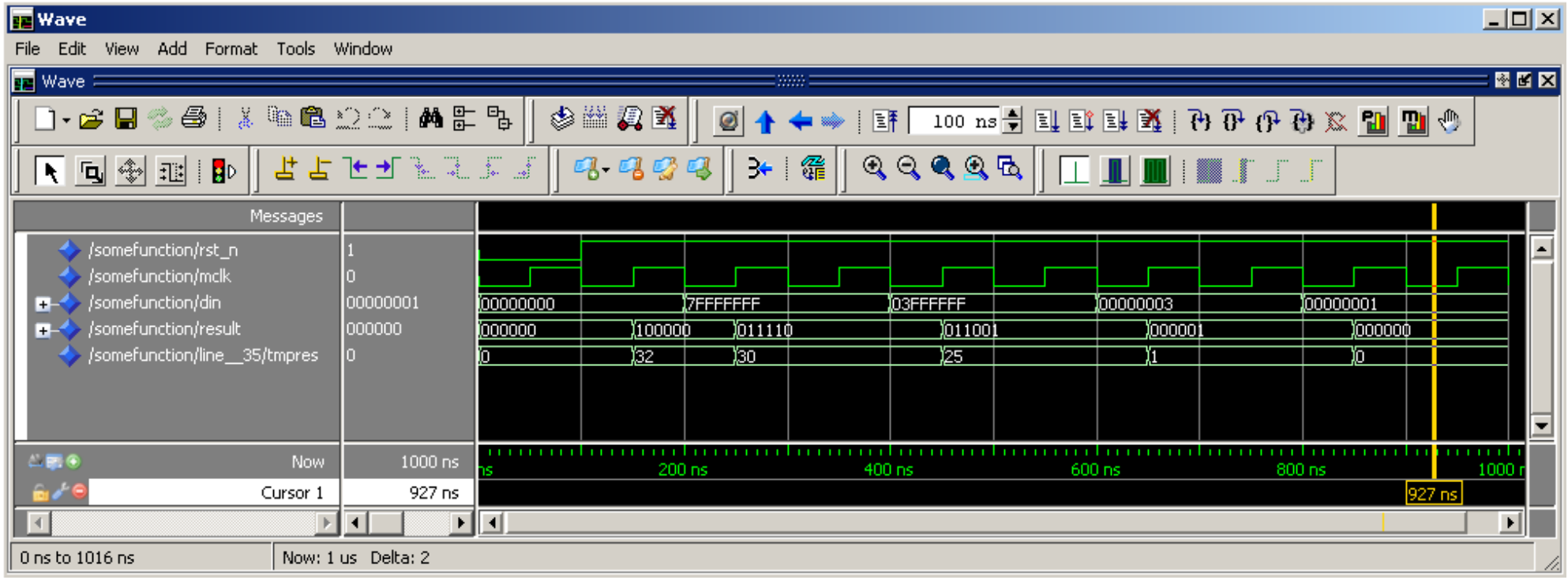
INF3430 Eksamen H09 VHDL besvarelse

Oppgave 7: signal_values



INF3430 Eksamen H09 VHDL besvarelse

Oppgave 8: **Funksjonen er en "Leading-one-detector" som returnerer posisjonen til første '1' verdi i vektor parameteren. Hvis alle bit er '0' returnere en verdi som er en større enn mulig tallområde.**



INF3430 Eksamen H09 VHDL besvarelse

Oppgave 9: procedure something .

NB: Bare procedure something kreves i besvarelsen.

```
architecture rtl of someprocedure is

  procedure something(a     : in  std_logic_vector;
                      value : out integer) is
    constant ALEN  : integer := a'length;
  begin
    value:= ALEN;
    for i in ALEN-1 downto 0 loop
      if a(i)='1' then
        value := i;
        exit;
      end if;
    end loop;
  end;

begin

  process (rst_n, mclk) is
    variable tmpres : integer;
  begin
    if (rst_n = '0') then
      tmpres := 0;
      result <= (others => '0');
    elsif rising_edge(mclk) then
      something(din, tmpres);
      result <= std_logic_vector(to_unsigned(tmpres,6));
    end if;
  end process;

end rtl;
```

INF3430 Eksamen H09 VHDL besvarelse

Oppgave 10 : to_concurrent

```
architecture rtl2 of to_concurrent is
begin

  abc_out <= (a and b) or c or d;
  d_out   <= d when ena='1' else 'Z';

end rtl2;
```

INF3430 Eksamen H09 VHDL besvarelse

Oppgave 11 : Dager i måneden

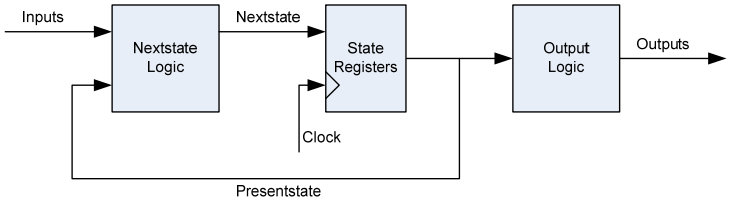
```
architecture rtl of dager is
begin

  P_DAYS: process(mnd, skuddaar)
  begin
    -- Skriv VHDL kode her:

    case mnd is
      when FEB =>
        if skuddaar then
          antdager <= "11101";
        else
          antdager <= "11100";
        end if;
      when JAN | MAR | MAY | JUL |
           AUG | OCT | DEC =>
        antdager <= "11111";
      when others =>
        antdager <= "11110";
    end case;

  end process;
end rtl;
```

Oppgave 12 a)

Maskinen beskrevet er en Mealy maskin pga. utgangene avhenger av inngangene i tillegg til nåværende tilstand.

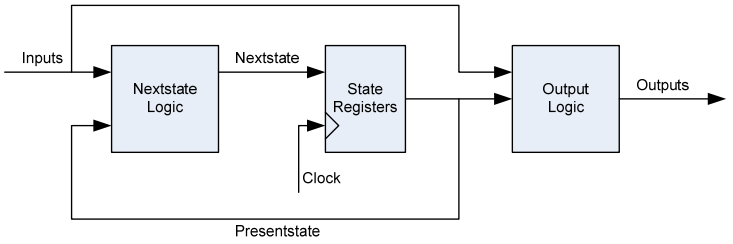
Moore maskin

- I en Moore tilstandsmaskin er utgangene en ren funksjon av nåværende tilstand (Presentstate)
- I en Moore maskin er det en klokkeperiode forsinkelse fra inngang til utgang

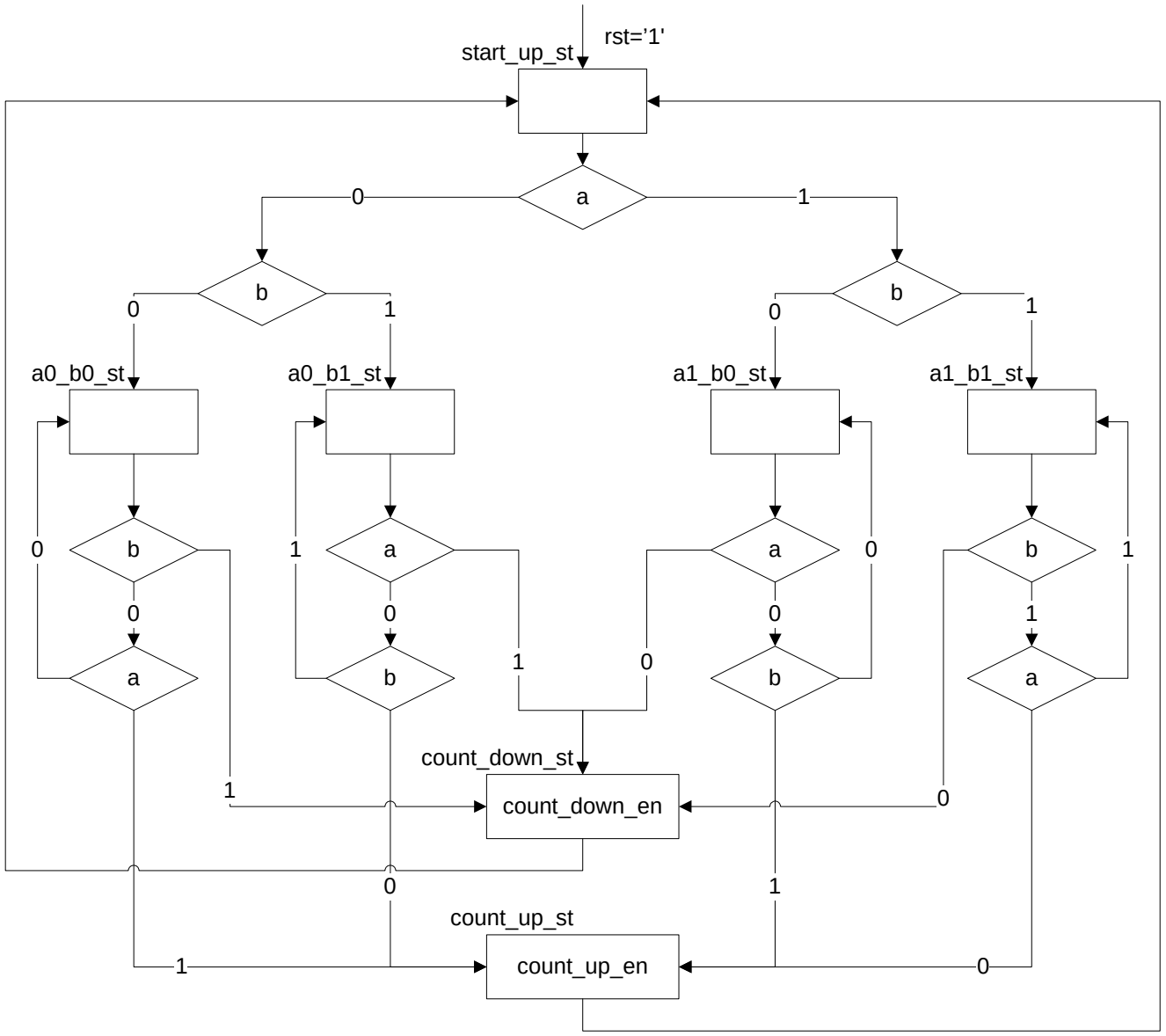


Mealy maskin

- I en Mealy tilstandsmaskin er utgangene en funksjon av nåværende tilstand og inngangene.A
- Raskere
  - Mindre forsinkelse fra inngang til utgang
- Mer komplisert dekoding av utganger



Oppgave 12b)


start_up_st
rst='1'
a
0
1
b
b
0
1
0
1
a0_b0_st
a0_b1_st
a1_b0_st
a1_b1_st
b
a
a
b
0
1
0
1
0
1
0
1
a
b
b
a
1
0
count_down_st
count_down_en
1
0
0
1
count_up_st
count_up_en
1
0

```vhdl
--************************************
-- INF3430/4430 Eksamen H2009
-- Fasit oppgave 12a
--************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity oppgave12a is
  port (
    rst     : in  std_logic;            -- Reset
    clk     : in  std_logic;            -- Clock
    c, d, e : in  std_logic;
    x, y, z : out std_logic
    );
end oppgave12a;

architecture oppgave12a_rtl of oppgave12a is
  type oppgave12a_states is (s0_st, s1_st, s2_st);
  signal current_st, next_st : oppgave12a_states;

begin

  state_reg : process(clk, rst)
  begin
    if rst = '1' then
      current_st <= s0_st;
    elsif rising_edge(clk) then
      current_st <= next_st;
    end if;
  end process state_reg;

  comb : process(c, current_st, d, e)
  begin
    x       <= '0';
    y       <= '0';
    z       <= '0';
    next_st <= s0_st;
    case current_st is
      when s0_st =>
        next_st <= s0_st;
        if c = '1' then
          next_st <= s1_st;
          x       <= '1';
        end if;
      when s1_st =>
        next_st <= s1_st;
        if d = '0' then
          if e = '1' then
            next_st <= s2_st;
            y       <= '1';
          end if;
        elsif d = '1' then
          next_st <= s2_st;
        end if;
      when s2_st =>
        next_st <= s0_st;
        z       <= '1';
    end case;
  end process comb;
end architecture oppgave12a_rtl;

--************************************
-- INF3430/4430 Eksamen H2009
-- Fasit oppgave 12c
--************************************

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity oppgave12c is
```

```
  74   generic (
  75     width : natural := 16
  76     );
  77   port (
  78     -- System Clock and Reset
  79     rst : in  std_logic;                -- Reset
  80     clk : in  std_logic;                -- Clock
  81     a   : in  std_logic;
  82     b   : in  std_logic;
  83     pos : out std_logic_vector(width-1 downto 0)
  84     );
  85 end oppgave12c;
  86 
  87 --************************************
  88 --  oppgave12c, Moore alternativ
  89 --************************************
  90 library IEEE;
  91 use IEEE.std_logic_1164.all;
  92 use IEEE.numeric_std.all;
  93 
  94 architecture rtl_moore of oppgave12c is
  95 
  96   type pos_states is (start_up_st, a0_b0_st, a0_b1_st, a1_b0_st, a1_b1_st,
  97                       count_up_st, count_down_st);
  98   signal next_st, current_st : pos_states;
  99   signal count_up_en         : std_logic;
 100   signal count_down_en       : std_logic;
 101   signal pos_i               : signed(width-1 downto 0);
 102 
 103 begin
 104 
 105   state_reg : process(clk, rst)
 106   begin
 107     if rst = '1' then
 108       current_st <= start_up_st;
 109     elsif rising_edge(clk) then
 110       current_st <= next_st;
 111     end if;
 112   end process;
 113 
 114   comb_moore : process(a, b, current_st)
 115   begin
 116     count_up_en   <= '0';
 117     count_down_en <= '0';
 118     case current_st is
 119       when start_up_st =>
 120         if a = '0' and b = '0' then
 121           next_st <= a0_b0_st;
 122         elsif a = '0' and b = '1' then
 123           next_st <= a0_b1_st;
 124         elsif a = '1' and b = '0' then
 125           next_st <= a1_b0_st;
 126         else
 127           next_st <= a1_b1_st;
 128         end if;
 129       when a0_b0_st =>
 130         if b = '1' then
 131           next_st <= count_down_st;
 132         elsif a = '1' then
 133           next_st <= count_up_st;
 134         else
 135           next_st <= a0_b0_st;
 136         end if;
 137       when a0_b1_st =>
 138         if a = '1' then
 139           next_st <= count_down_st;
 140         elsif b = '0' then
 141           next_st <= count_up_st;
 142         else
 143           next_st <= a0_b1_st;
 144         end if;
 145       when a1_b0_st =>
 146         if a = '0' then
```

```
147           next_st <= count_down_st;
148         elsif b = '1' then
149           next_st <= count_up_st;
150         else
151           next_st <= a1_b0_st;
152         end if;
153       when a1_b1_st =>
154         if b = '0' then
155           next_st <= count_down_st;
156         elsif a = '0' then
157           next_st <= count_up_st;
158         else
159           next_st <= a1_b1_st;
160         end if;
161       when count_down_st =>
162         count_down_en <= '1';
163         next_st       <= start_up_st;
164       when count_up_st =>
165         count_up_en <= '1';
166         next_st     <= start_up_st;
167     end case;
168   end process;
169
170   pos_counter : process(clk, rst)
171   begin
172     if rst = '1' then
173       pos_i <= (others => '0');
174     elsif rising_edge(clk) then
175       if count_up_en = '1' and pos_i < 1439 then
176         pos_i <= pos_i + 1;
177       elsif count_down_en = '1' and pos_i > 0 then
178         pos_i <= pos_i - 1;
179       end if;
180     end if;
181   end process;
182
183   pos <= std_logic_vector(pos_i);
184
185 end architecture rtl_moore;
186
187
188 --************************************
189 --  oppgave12c, Mealy alternativ
190 --  Ikke spørt etter
191 --************************************
192
193 library IEEE;
194 use IEEE.std_logic_1164.all;
195 use IEEE.numeric_std.all;
196
197 architecture rtl_mealy of oppgave12c is
198
199   type pos_states is (start_up_st, a0_b0_st, a0_b1_st, a1_b0_st, a1_b1_st);
200   signal next_st, current_st : pos_states;
201   signal count_up_en         : std_logic;
202   signal count_down_en       : std_logic;
203   signal pos_i               : signed(width-1 downto 0);
204
205 begin
206
207   state_reg : process(clk, rst)
208   begin
209     if rst = '1' then
210       current_st <= start_up_st;
211     elsif rising_edge(clk) then
212       current_st <= next_st;
213     end if;
214   end process;
215
216   comb_mealy : process(a, b, current_st)
217   begin
218     count_up_en   <= '0';
219     count_down_en <= '0';
```

```
220     case current_st is
221       when start_up_st =>
222         if a = '0' and b = '0' then
223           next_st <= a0_b0_st;
224         elsif a = '0' and b = '1' then
225           next_st <= a0_b1_st;
226         elsif a = '1' and b = '0' then
227           next_st <= a1_b0_st;
228         else
229           next_st <= a1_b1_st;
230         end if;
231       when a0_b0_st =>
232         if b = '1' then
233           next_st       <= a0_b1_st;
234           count_down_en <= '1';
235         elsif a = '1' then
236           next_st     <= a1_b0_st;
237           count_up_en <= '1';
238         else
239           next_st <= a0_b0_st;
240         end if;
241       when a0_b1_st =>
242         if a = '1' then
243           next_st       <= a1_b1_st;
244           count_down_en <= '1';
245         elsif b = '0' then
246           next_st     <= a0_b0_st;
247           count_up_en <= '1';
248         else
249           next_st <= a0_b1_st;
250         end if;
251       when a1_b0_st =>
252         if a = '0' then
253           next_st       <= a0_b0_st;
254           count_down_en <= '1';
255         elsif b = '1' then
256           next_st     <= a1_b1_st;
257           count_up_en <= '1';
258         else
259           next_st <= a1_b0_st;
260         end if;
261       when a1_b1_st =>
262         if b = '0' then
263           next_st       <= a1_b0_st;
264           count_down_en <= '1';
265         elsif a = '0' then
266           next_st     <= a0_b1_st;
267           count_up_en <= '1';
268         else
269           next_st <= a1_b1_st;
270         end if;
271     end case;
272
273   end process;
274
275   pos_counter : process(clk, rst)
276   begin
277     if rst = '1' then
278       pos_i <= (others => '0');
279     elsif rising_edge(clk) then
280       if count_up_en = '1' and pos_i < 1439 then
281         pos_i <= pos_i + 1;
282       elsif count_down_en = '1' and pos_i > 0 then
283         pos_i <= pos_i - 1;
284       end if;
285     end if;
286   end process;
287
288   pos <= std_logic_vector(pos_i);
289
290 end architecture rtl_mealy;
291
292 --********************************
```

```
293 -- INF3430/4430 Eksamen H2009
294 -- Testbenk (ikke krav)
295 --*******************************
296
297 library ieee;
298 use ieee.std_logic_1164.all;
299 use ieee.numeric_std.all;
300
301 entity tb_oppgave12c is
302 end tb_oppgave12c;
303
304 architecture testbench of tb_oppgave12c is
305
306   component oppgave12c is
307     generic (
308       width : natural := 16
309       );
310     port (
311       -- System Clock   and     Reset
312       rst : in  std_logic;              -- Reset
313       clk : in  std_logic;              -- Clock
314       a   : in  std_logic;
315       b   : in  std_logic;
316       pos : out std_logic_vector(width-1 downto 0)
317       );
318   end component oppgave12c;
319
320   constant my_width : integer   := 16;
321   signal rst        : std_logic := '0';  -- Reset
322   signal clk        : std_logic := '0';  -- Clock
323   signal a          : std_logic := '0';
324   signal b          : std_logic := '0';
325   signal pos        : std_logic_vector(my_width-1 downto 0);
326   constant phase90  : time      := 200 ns;
327   constant max_val  : integer   := 1439;
328   constant min_val  : integer   := 0;
329   constant max_time : integer   := (max_val+1)/4;
330
331 begin
332
333   clk <= not clk after 10 ns;  -- 50HHz klokke
334
335   UUT : oppgave12c
336     generic map(my_width)
337     port map(
338       rst => rst,
339       clk => clk,
340       a   => a,
341       b   => b,
342       pos => pos
343       );
344
345   STIMULI : process
346
347     procedure up(periods : natural) is
348     begin
349       --wait for phase90;
350       for i in 0 to periods loop
351         a <= '1';
352         wait for phase90;
353         b <= '1';
354         wait for phase90;
355         a <= '0';
356         wait for phase90;
357         b <= '0';
358         wait for phase90;
359       end loop;
360     end;
361
362     procedure down(periods : natural) is
363     begin
364       --wait for phase90;
365       for i in 0 to periods loop
```

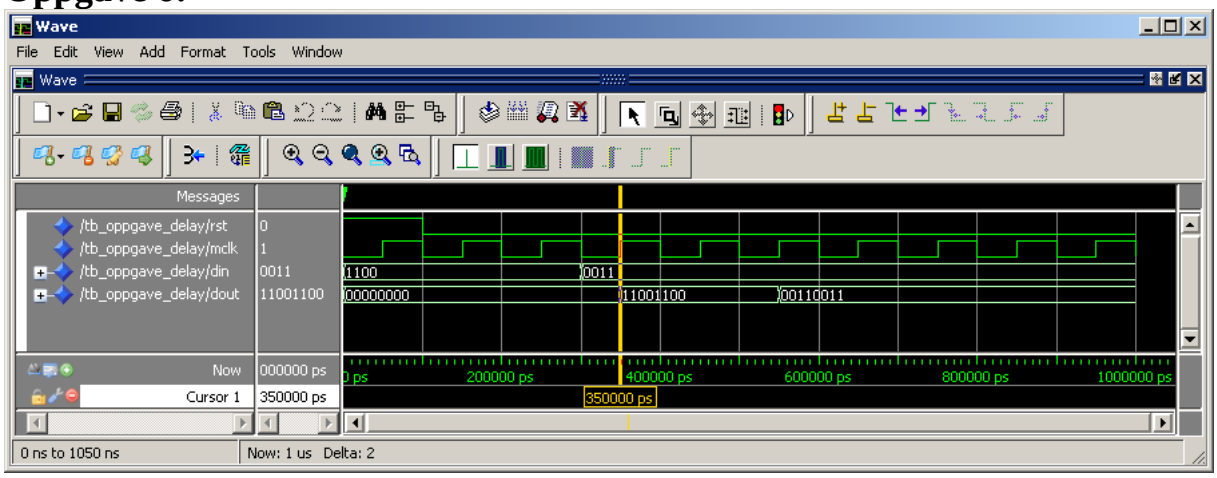
```
366         a <= '0';
367         wait for phase90;
368         b <= '1';
369         wait for phase90;
370         a <= '1';
371         wait for phase90;
372         b <= '0';
373         wait for phase90;
374       end loop;
375     end;
376
377   begin
378     rst <= '1', '0' after 100 ns;
379     wait for phase90*2;
380     --Simulating upwards to     above max       value
381     up(max_time);
382     --Simulating downwards to   below min       value
383     wait for 100 us;
384     down(max_time);
385     --Simulating up and down to see if the position
386     --counter changes direction correctly
387     wait for 100 us;
388     up(20);
389     wait for 100 us;
390     down(10);
391     wait for 100 us;
392     up(2);
393     wait for 100 us;
394     down(100);
395     wait;
396
397   end process STIMULI;
398 end architecture testbench;
399
400 --Ikke forventet bruk av configurations
401 configuration cfg_test_oppgave12c_moore of tb_oppgave12c is
402   for testbench
403     for UUT : oppgave12c use entity work.oppgave12c(rtl_moore);
404     end for;
405   end for;
406 end configuration;
407
408 configuration cfg_test_oppgave12c_mealy of tb_oppgave12c is
409   for testbench
410     for UUT : oppgave12c use entity work.oppgave12c(rtl_mealy);
411     end for;
412   end for;
413 end configuration;
```

Fasit INF3430 H2010.

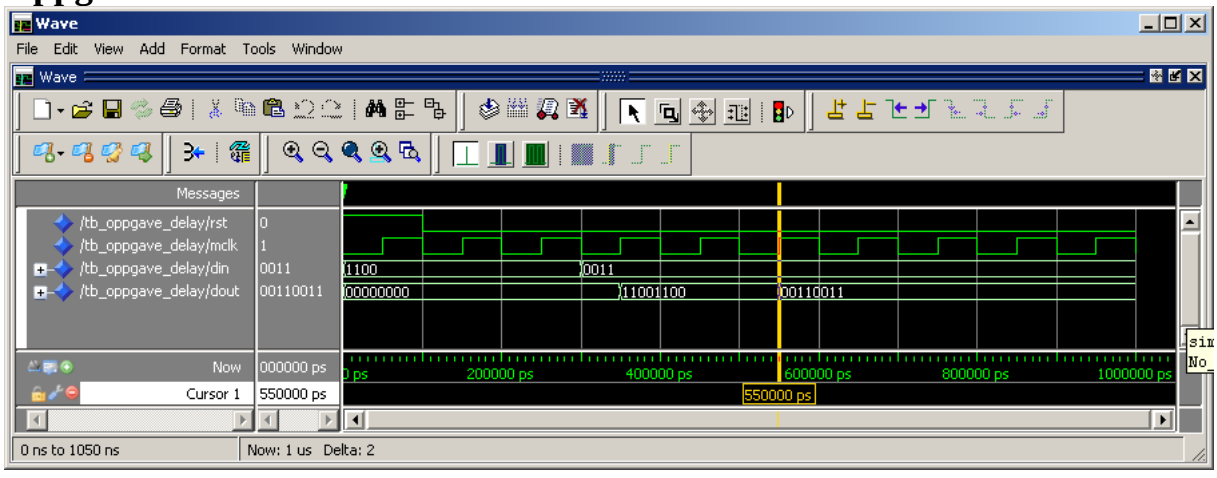
Oppgave 1 -7:

Oppgave	A	B	C	D	E
1		X			
2		X	X		
3	X	X			X
4	X	X		X	
5			X		
6			X		
7				X	

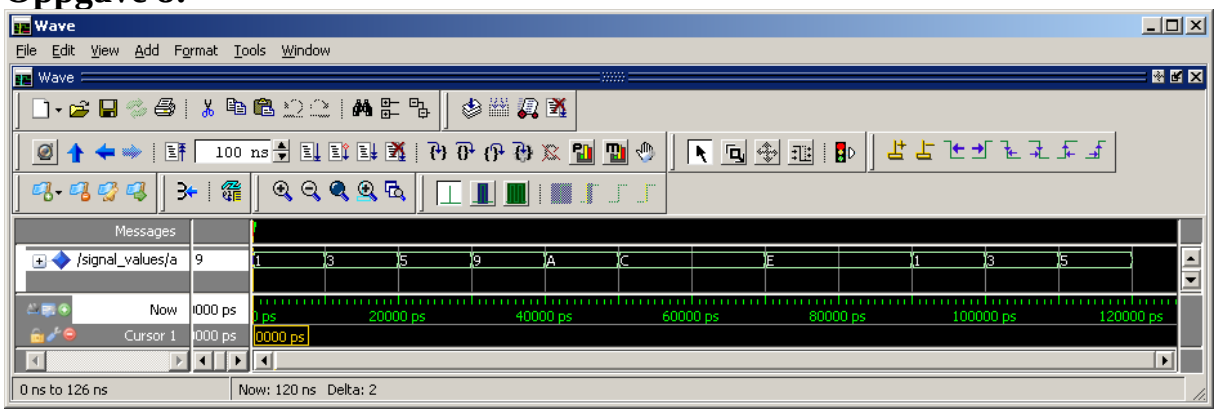
Oppgave 6:



Oppgave 7:



Oppgave 8:



Oppgave 9:

```
process (rst, mclk) is
  variable cnt_i : unsigned(7 downto 0);

begin
  if (rst = '1') then
    cnt   <= (others => '0');
  elsif rising_edge(mclk) then
    cnt_i := (others => '0');
    for i in 0 to 31 loop
      if data(i)='1' then
        cnt_i := cnt_i + 1;
      end if;
    end loop;
    cnt <= std_logic_vector(cnt_i);
  end if;

end process;
```

Oppgave 10:

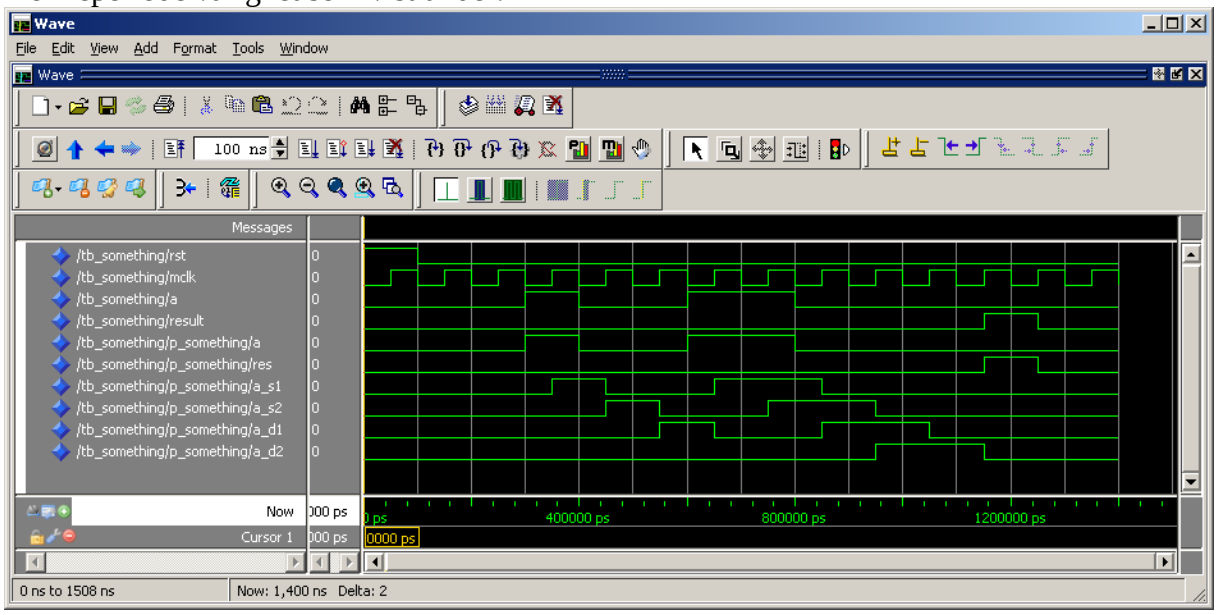
```
procedure pcount (signal a   :  in std_logic_vector;
                  signal cnt : out std_logic_vector) is

  variable cnt_i : unsigned(7 downto 0);
begin
  cnt_i := (others => '0');
  for i in a'range loop
    if a(i)='1' and cnt_i<x"FF" then
      cnt_i := cnt_i + 1;
    end if;
  end loop;
  cnt <= std_logic_vector(cnt_i);

end procedure;
```

Oppgave 11:

```
architecture rtl_3 of oppgave_counter is

  function fcount (signal a : in std_logic_vector)
    return std_logic_vector is

    variable cnt_i : unsigned(7 downto 0);
  begin
    cnt_i := (others => '0');
    for i in a'range loop
      if a(i)='1' and cnt_i<x"FF" then
        cnt_i := cnt_i + 1;
      end if;
    end loop;
    return std_logic_vector(cnt_i);

  end function;

begin

  process (rst, mclk) is
  begin
    if (rst = '1') then
      cnt <= (others => '0');
    elsif rising_edge(mclk) then
      cnt <= fcount(data);
    end if;
  end process;

end rtl_3;
```

Oppgave 12:

Something filtrerer inngangsignalet først for å fjerne evnt. uønskede ”spikere” på inngangssignalet. Inngangssignalet må derfor være høyt minst to påfølgende klokkeperioder for at logikken skal detektere ’1’ til ’0’ overgang og lage et høytgående strobesignal med en klokkeperiode varighet som vist under.



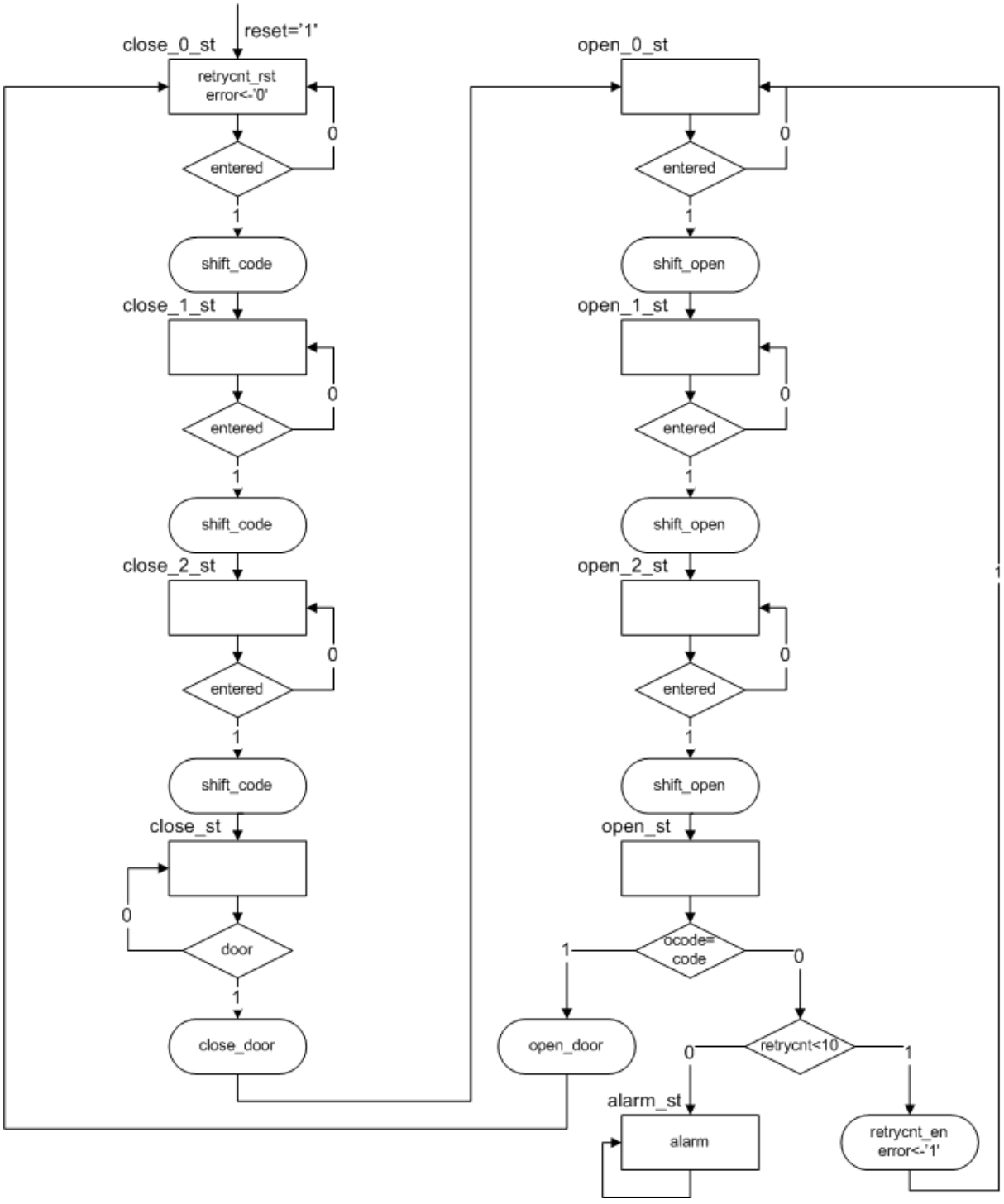
Oppgave 13

a)

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

package code_lock_pkg is
  type code_type is array (0 to 2) of unsigned(3 downto 0);
end package code_lock_pkg;

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.code_lock_pkg.all;

entity code_sreg is
  port
  (
    reset      : in std_logic;
    clk        : in std_logic;
    shift_code : in std_logic;
    code_in    : in std_logic_vector(3 downto 0);
    code_out   : out code_type
  );
end ;

architecture rtl of code_sreg is
 signal code_i: code_type;
begin

--Start å skrive VHDL kode her
--Start Oppgave 13a)
opencode:
process(reset,clk)
begin
  if reset = '1' then
    for i in 0 to 2 loop
      code_i(i) <= (others => '0');
    end loop;
  elsif rising_edge(clk) then
    if shift_code = '1' then
      code_i(0) <= unsigned(code_in);
      for i in 1 to 2 loop
        code_i(i) <= code_i(i-1);
      end loop;
    end if;
  end if;
end process;
code_out <= code_i;

--Slutt å skrive VHDL kode her
--Slutt Oppgave 13a)
end architecture rtl;
```

b)

Kodelåsen kan realiseres på mange måter. Istenfor close(open)_i_st, i=1,2,3 kan man f.eks. benytte en tilstand pluss en teller. På den måten er det også lett å utvide tilstandsmaskinen til N antall siffer.



c)

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

```
entity codelock is
  port
  (
    reset     : in std_logic;
    clk       : in std_logic;
    code_in   : in std_logic_vector(3 downto 0);
    entered   : in std_logic;
    door      : in std_logic;
    open_door : out std_logic;
    close_door: out std_logic;
    error     : out std_logic;
    alarm     : out std_logic
  );
end codelock;

architecture rtl of codelock is

  signal shift_code     : std_logic;
  signal code           : code_type;
  signal shift_code_out : std_logic;
  signal code_out       : code_type;

  type codelock_states is (close_0_st,close_1_st,close_2_st, close_st,
                           open_0_st,open_1_st,open_2_st,open_st,alarm_st);

  signal curr_st,next_st          :codelock_states;
  signal retrycnt                 : unsigned(3 downto 0);
  signal retrycnt_en,retrycnt_rst : std_logic;
  signal error_en                 : std_logic;
  signal error_i                  : std_logic;

  component code_sreg is
    port
    (
      reset      : in std_logic;
      clk        : in std_logic;
      shift_code : in std_logic;
      code_in    : in std_logic_vector(3 downto 0);
      code_out   : out code_type
    );
  end component code_sreg;

begin

code_shift_reg: code_sreg
port map
  (
    reset      => reset,
    clk        => clk,
    shift_code => shift_code,
    code_in    => code_in,
    code_out   => code
  );

code_out_shift_reg: code_sreg
port map
  (
    reset      => reset,
    clk        => clk,
    shift_code => shift_code_out,
    code_in    => code_in,
    code_out   => code_out
```

```
  );

retry_counter:
process(reset,clk)
begin
  if reset = '1' then
    retrycnt <= (others => '0');
  elsif rising_edge(clk) then
    if retrycnt_rst = '1' then
      retrycnt <= (others => '0');
    elsif retrycnt_en = '1' then
      retrycnt <= retrycnt + 1;
    end if;
  end if;
end process;

error_reg:
process(reset,clk)
  begin
  if reset = '1' then
    error <= '0';
  elsif rising_edge(clk) then
    if error_en = '1' then
      error <= error_i;
    end if;
  end if;
end process error_reg;

--Start Oppgave 13c)
current_state_reg: process(reset,clk)
begin
  if reset = '1' then
    curr_st <= close_0_st;
  elsif rising_edge(clk) then
    curr_st <= next_st;
  end if;
end process;

next_state_comb:process(curr_st,entered,door,code,code_out)
begin

  shift_code    <= '0';
  shift_code_out<= '0';
  open_door     <= '0';
  close_door    <= '0';
  error_en      <= '0';
  error_i       <= '0';
  alarm         <= '0';
  retrycnt_en   <= '0';
  retrycnt_rst  <= '0';
  case curr_st is
    when  close_0_st =>
      error_en <= '1';
      retrycnt_rst <= '1';
      if entered = '1' then
        shift_code <= '1';
        next_st <= close_1_st;
      else
        next_st <= close_0_st;
      end if;
    when  close_1_st =>
      if entered = '1' then
```

```
        shift_code <= '1';
        next_st <= close_2_st;
      else
        next_st <= close_1_st;
      end if;
    when  close_2_st =>
      if entered = '1' then
        shift_code <= '1';
        next_st <= close_st;
      else
        next_st <= close_2_st;
      end if;
    when  close_st =>
      if door = '1' then
        close_door <= '1';
        next_st <= open_0_st;
      else
        next_st <= close_st;
      end if;
    when  open_0_st =>
      if entered = '1' then
        shift_code_out <= '1';
        next_st <= open_1_st;
      else
        next_st <= open_0_st;
      end if;
    when  open_1_st =>
      if entered = '1' then
        shift_code_out <= '1';
        next_st <= open_2_st;
      else
        next_st <= open_1_st;
      end if;
    when  open_2_st =>
      if entered = '1' then
        shift_code_out <= '1';
        next_st <= open_st;
      else
        next_st <= open_2_st;
      end if;
    when  open_st =>
      if code_out = code then
        open_door <= '1';
        next_st <= close_0_st;
      elsif retrycnt < 10 then
        next_st <= open_0_st;
        retrycnt_en <= '1';
        error_en <= '1';
        error_i <= '1';
      else
        next_st <= alarm_st;
      end if;
    when alarm_st =>
      alarm <= '1';
      next_st <= alarm_st;
    when others =>
      next_st <= close_0_st;

  end case;
end process;

--Slutt Oppgave 13c)
```

```
end rtl;
```

d)

```
--Start Oppgave 13d)
library IEEE;
use IEEE.std_logic_1164.all;

entity  tb_codelock is
end ;

architecture testbench of tb_codelock is

component codelock is
  port
  (
    reset     : in std_logic;
    clk       : in std_logic;
    code_in   : in std_logic_vector(3 downto 0);
    entered   : in std_logic;
    door      : in std_logic;

    open_door : out std_logic;
    close_door: out std_logic;
    error     : out std_logic;
    alarm     : out std_logic
  );
end component codelock;

signal  reset       :  std_logic := '0';
signal  clk         :  std_logic := '0';
signal  code_in     :  std_logic_vector(3 downto 0) := X"0";
signal  entered     :  std_logic := '0';
signal  door        :  std_logic := '0';
signal  open_door   :  std_logic;
signal  close_door  :  std_logic;
signal  error       :  std_logic;
signal  alarm       :  std_logic;

begin

UUT: codelock
port map
(
  reset       => reset,
  clk         => clk,
  code_in     => code_in,
  entered     => entered,
  door        => door,
  open_door   => open_door,
  close_door  => close_door,
  error       => error,
  alarm       => alarm
);

clk <= not clk after 10 ns;

stimuli: process
```

```
  procedure enter_code(value     : in std_logic_vector;  --kode
                       lock_door : in integer; --1 dersom dør lukkes
                       next_time : in time     --tid for neste verdi
(relativt til now)
                      ) is
  begin
    for i in 0 to 2 loop
      code_in <= value(4*i to 4*i+3);
      entered <= '1','0' after 20 ns;
      wait for next_time;
    end loop;
    --Lukker safedør etter å ha tastet inn koden
    if lock_door = 1 then
      wait for 100 ns;
      door <= '1','0' after 100 ns;
      wait for 200 ns;
    end if;
  end;

begin
  reset <= '1','0' after 100 ns;
  wait for 200 ns;
  wait until rising_edge(clk);
  --taster inn kode og lukker safedør
  enter_code(X"123",1,40 ns);

  --taster inn kode for å åpne dør
  enter_code(X"123",0,40 ns);

  wait for 200 ns;
  wait until rising_edge(clk);
  --taster inn kode og lukker safedør
  enter_code(X"123",1,40 ns);

  for i in 0 to 3 loop
    wait for 200 ns;
    --prøver å åpne med feil kode
    enter_code(X"321",0,40 ns);
  end loop;
  enter_code(X"123",0,40 ns);

  --Skal prøve alarm funksjon
  wait for 200 ns;
  wait until rising_edge(clk);
  --taster inn kode og lukker safedør
  enter_code(X"123",1,40 ns);

  for i in 0 to 12 loop
    wait for 200 ns;
    --prøver å åpne med feil kode
    enter_code(X"321",0,40 ns);
  end loop;

  --wait;
end process;

end architecture testbench;
--Slutt Oppgave 13d)
```

INF3430/INF4431 H2011.

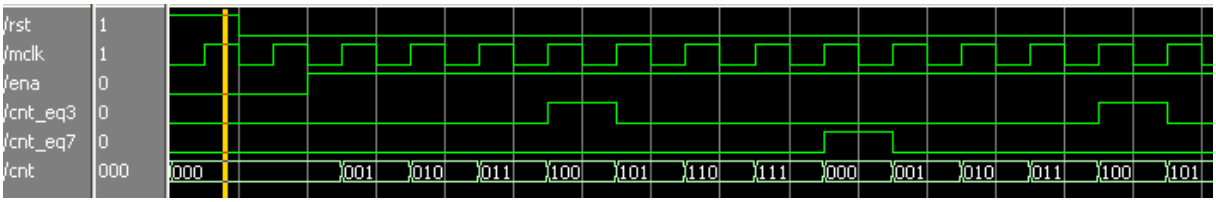
Fasit oppgave 1 - 5:

Oppgave	A	B	C	D	E
1		X			
2	X	X			X
3		X			X
4		X	X		X
5	X		X		

Fasit oppgave 6 for INF3430:

Oppgave	A	B	C	D	E
6	X			X	

Fasit oppgave 7 for INF3430 og oppgave 6 for INF4431:



Fasit oppgave 7 for INF4431:

```
module counter (
  input  logic rst,
  input  logic mclk,
  input  logic ena,
  output logic cnt_eq3,
  output logic cnt_eq7,
  output logic [2:0] cnt
);

logic [2:0] cnt_i;
logic cnt_eq3_i, cnt_eq7_i;
```

```
always_comb
  begin
    cnt_i= cnt+1;
    cnt_eq3_i= '0;
    if (cnt==3)
      cnt_eq3_i= '1;
    cnt_eq7_i= '0;
    if (cnt==7)
      cnt_eq7_i= '1;
  end

always_ff @(posedge mclk, posedge rst)
  if (rst)
    begin
      cnt        <= '0;
      cnt_eq3    <= '0;
      cnt_eq7    <= '0;
    end else begin
      if (ena)
        cnt <= cnt_i;
      cnt_eq3    <= cnt_eq3_i;
      cnt_eq7    <= cnt_eq7_i;
    end

endmodule
```

eller <u>alternativ</u> løsning:

```
module counter (
  input  logic rst,
  input  logic mclk,
  input  logic ena,
  output logic cnt_eq3,
  output logic cnt_eq7,
  output logic [2:0] cnt
);

always_ff @(posedge mclk, posedge rst)
  if (rst)
    begin
      cnt     <= '0;
      cnt_eq3 <= '0;
      cnt_eq7 <= '0;
    end else begin
      if (ena)
        cnt <= cnt+1;

      cnt_eq3<= '0;
      if (cnt==3)
        cnt_eq3<= '1;

     cnt_eq7<= '0;
      if (cnt==7)
        cnt_eq7<= '1;

    end
endmodule
```

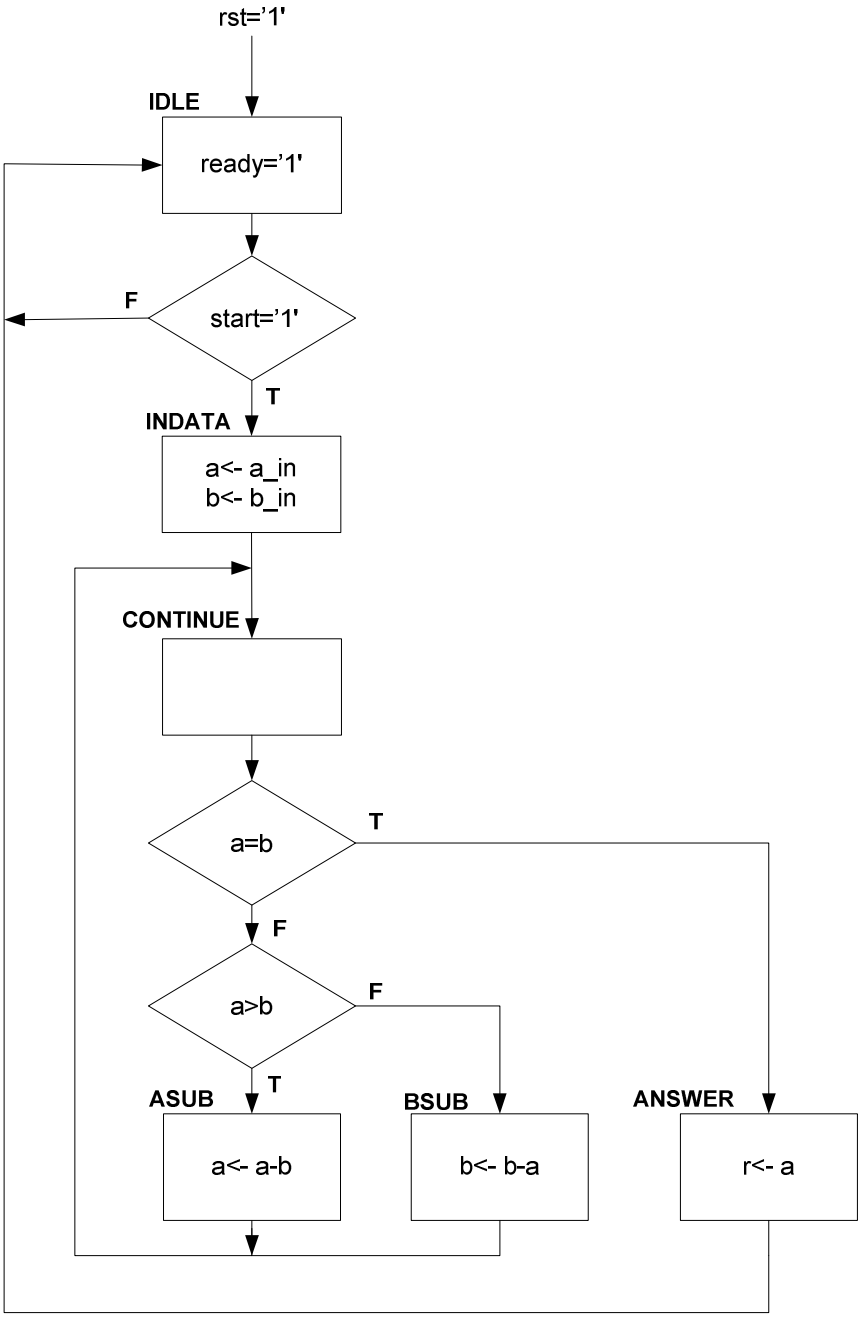
Fasit oppgave 8:

```
architecture rtl of busunit is
begin

   databus <= a_dout  when ab_sel='0' and  a_dout_ena='1' else
              b_dout  when ab_sel='1' and  b_dout_ena='1' else
              (others => 'Z');
   a_din   <= databus when ab_sel='0' else (others => '0');
   b_din   <= databus when ab_sel='1' else (others => '0');

end rtl;
```

eller <u>alternativ</u> løsning:

```
architecture rtl of busunit is
begin

   databus <= a_dout  when ab_sel='0' and  a_dout_ena='1' else
              (others => 'Z');
   a_din   <= databus when ab_sel='0' else (others => '0');


   databus <= b_dout  when ab_sel='1' and  b_dout_ena='1' else
              (others => 'Z');
   b_din   <= databus when ab_sel='1' else (others => '0');

end rtl;
```

Fasit oppgave 9:



Fasit oppgave 10:

Modulen Something er en register basert FIFO (First-In-First-Out) kø.

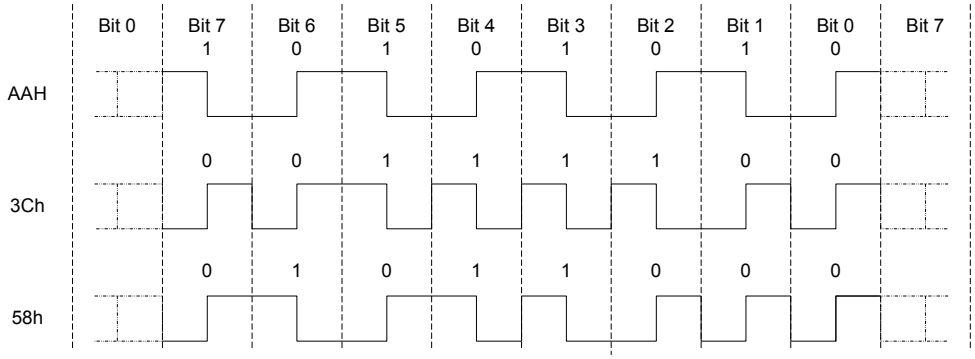
Data fra inngangssignalet *din* blir klokket inn i FIFO'en når signalet *ctrl1* er '1' og FIFO'en samtidig ikke er full når signalet *status1* er '0'.

Signalet *dout* viser alltid første element i FIFO'en og *status2* signalet er lik '1' når FIFO'en er tom.

Signalet *status3* viser antall ord i FIFO'en.

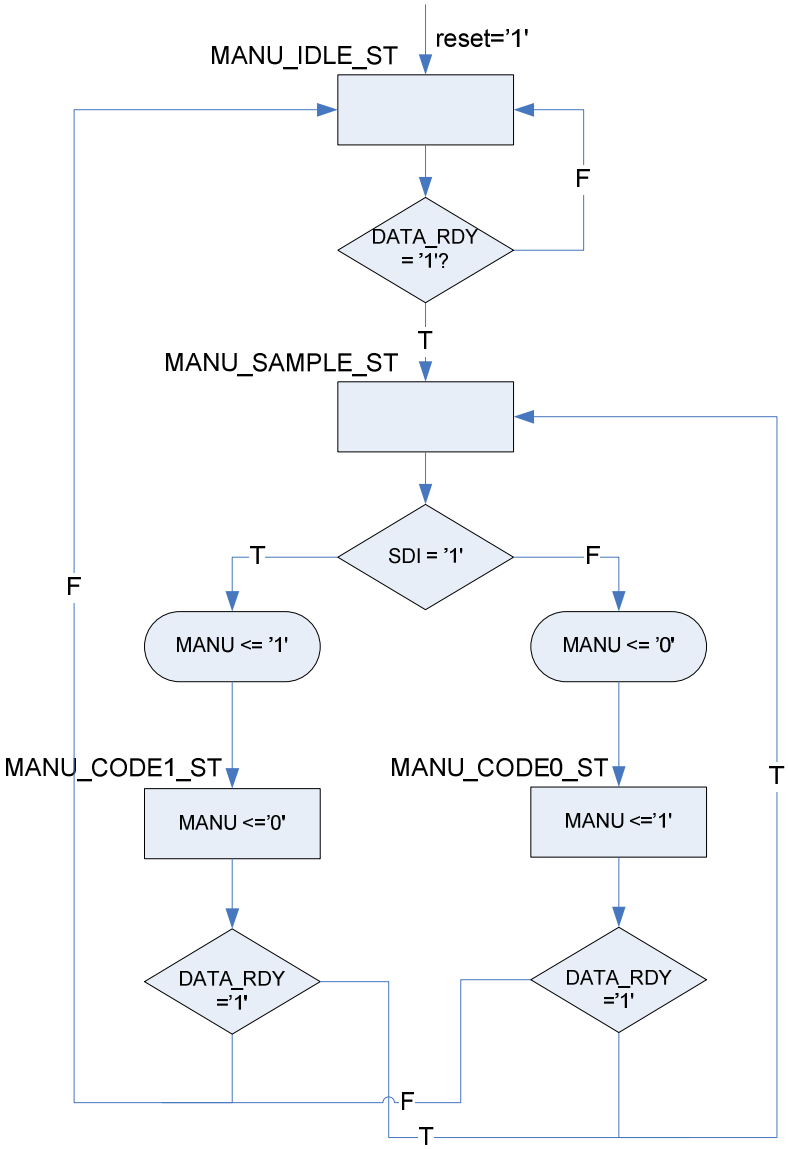
Fasit oppgave 11:

a)

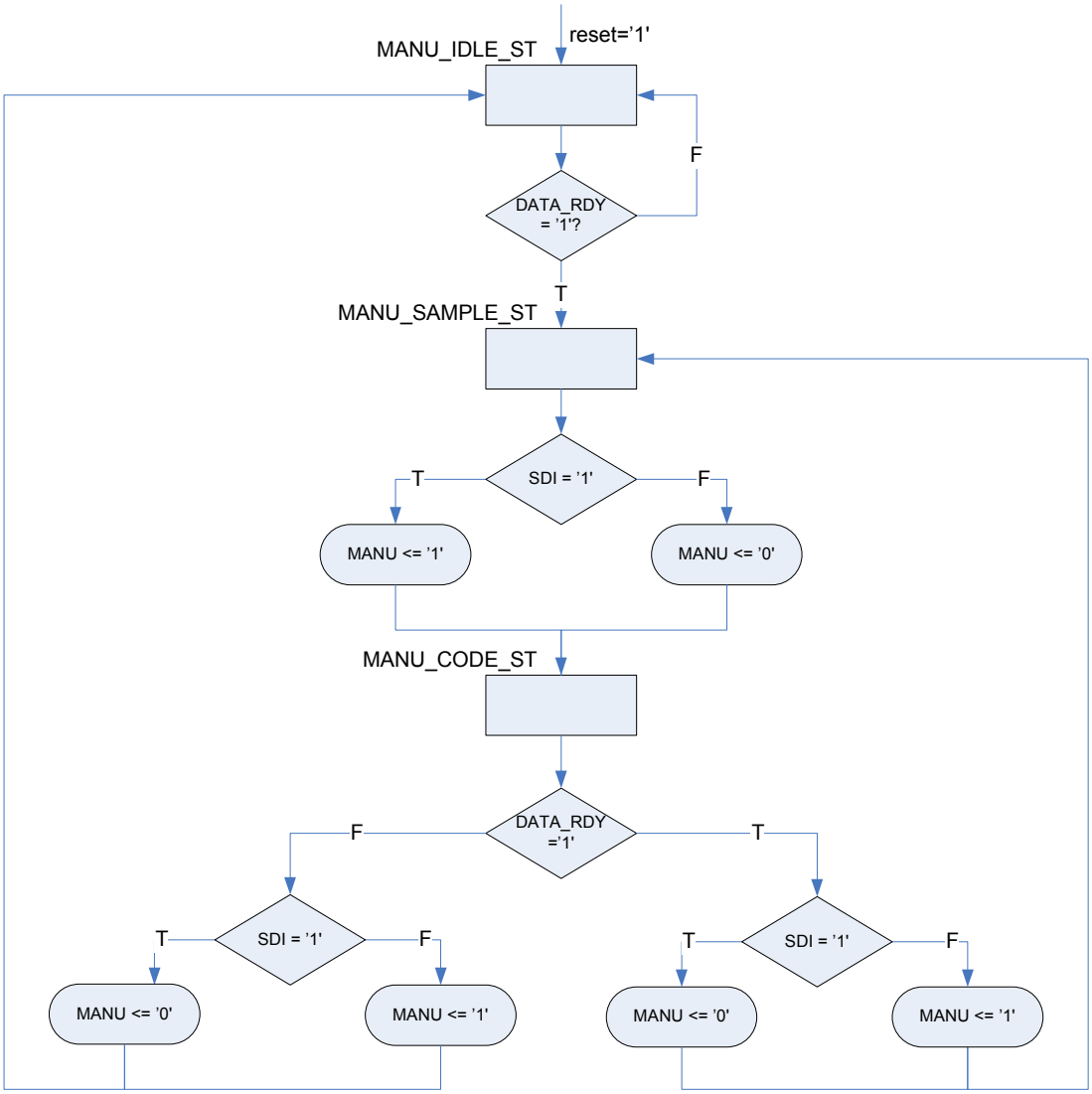


b)

Alt 1



Alt 2



c)

Oppgave 11c.vhd

```
--Oppgave 12c
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity MANCHESTER is
  port
  (
    RESET     : in std_logic;  --Asynkron reset
    CLK       : in std_logic;  --Klokke
    DATA_RDY  : in std_logic;  --Viser gyldige data inn
    SDI       : in std_logic;  --Serielle input data
    MANU      : out std_logic --Manchester kodet sekvens ut
  );
end MANCHESTER;

architecture RTL_MANCHESTER of MANCHESTER is

  --Alt 1 datattype
  type MANCHSTER_ST_TYPE is (MANU_IDLE_ST,MANU_SAMPLE_ST,MANU_CODE0_ST,MANU_CODE1_ST);
  --Alt 2 datatype
```

```
  --type MANCHSTER_ST_TYPE is (MANU_IDLE_ST,MANU_SAMPLE_ST,MANU_CODE_ST);
  signal MANU_CURR_ST,MANU_NEXT_ST : MANCHSTER_ST_TYPE;

begin

  MANU_ST_REG: process(RESET,CLK)
  begin
    if RESET = '1' then
      MANU_CURR_ST <= MANU_IDLE_ST;
    elsif rising_edge(CLK) then
      MANU_CURR_ST <= MANU_NEXT_ST;
    end if;
  end process;

  MANU_COMB_ST: process(MANU_CURR_ST,DATA_RDY)
  begin
    case MANU_CURR_ST is
      when MANU_IDLE_ST =>
        if DATA_RDY = '1' then
          MANU_NEXT_ST <= MANU_SAMPLE_ST;
        else
          MANU_NEXT_ST <= MANU_IDLE_ST;
        end if;
--Alt 1
      when MANU_SAMPLE_ST =>
        if SDI = '1' then
          MANU <= '1';
          MANU_NEXT_ST <= MANU_CODE1_ST;
        else
          MANU <= '0';
          MANU_NEXT_ST <= MANU_CODE0_ST;
        end if;
      when MANU_CODE0_ST =>
        MANU <= '1';
        if DATA_RDY = '0' then
          MANU_NEXT_ST <= MANU_IDLE_ST;
        else
          MANU_NEXT_ST <= MANU_SAMPLE_ST;
        end if;
      when MANU_CODE1_ST =>
        MANU <= '0';
        if DATA_RDY = '0' then
          MANU_NEXT_ST <= MANU_IDLE_ST;
        else
          MANU_NEXT_ST <= MANU_SAMPLE_ST;
        end if;
--End alt 1
--Alt 2
--      when MANU_SAMPLE_ST =>
--        MANU_NEXT_ST <= MANU_CODE_ST;
--        if SDI = '1' then
--          MANU <= '1';
--        else
--          MANU <= '0';
--        end if;
--      when MANU_CODE_ST =>
--        if DATA_RDY = '0' then
--          MANU_NEXT_ST <= MANU_IDLE_ST;
--          if SDI = '1' then
--            MANU <= '0';
--          else
--            MANU <= '1';
--          end if;
--        else
--          MANU_NEXT_ST <= MANU_SAMPLE_ST;
--          if SDI = '1' then
--            MANU <= '0';
--          else
--            MANU <= '1';
--          end if;
--        end if;
--End alt 2
```

```
      when others =>
        MANU_NEXT_ST <= MANU_IDLE_ST;
    end case;
  end process;
end architecture RTL_MANCHESTER;
```

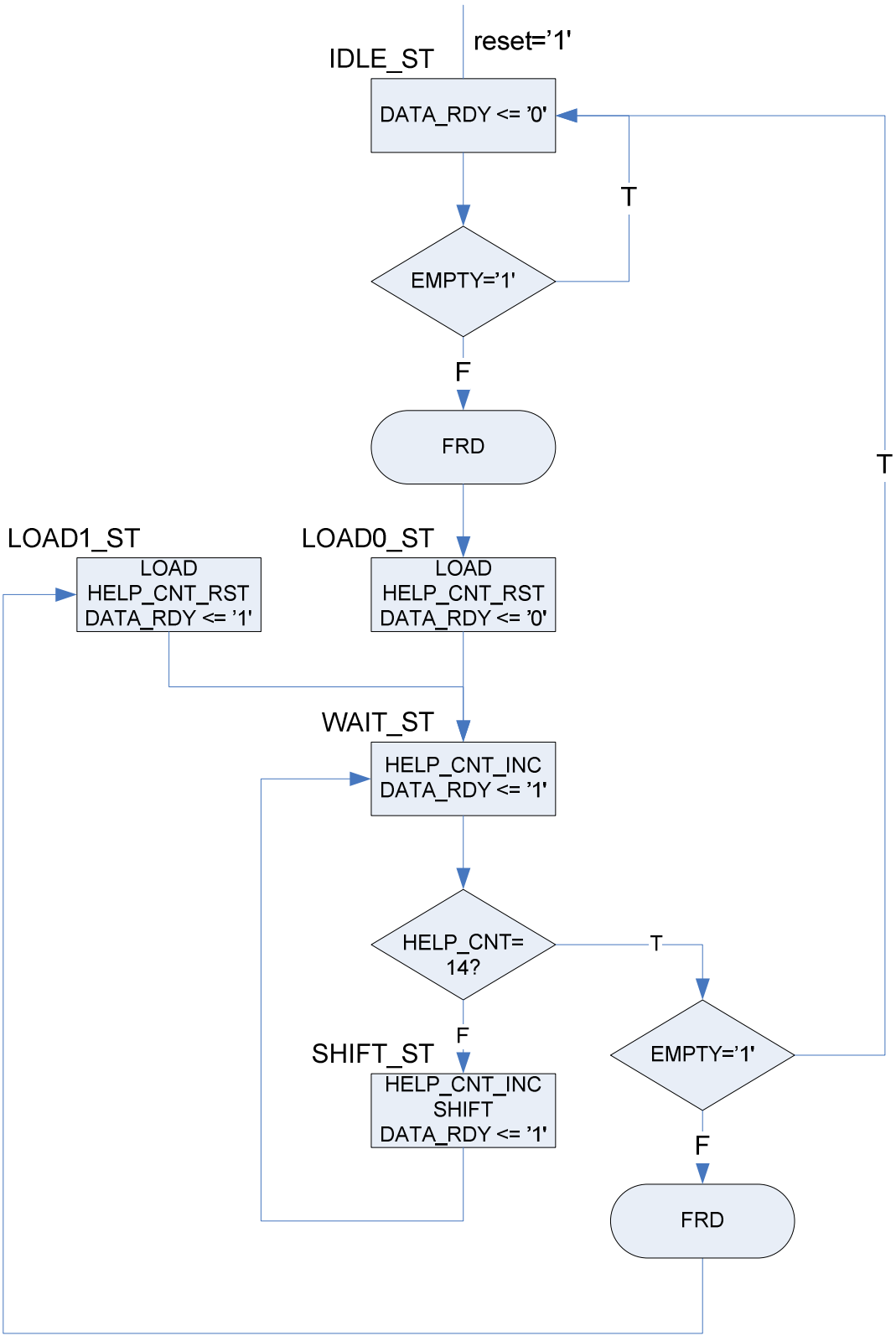
d)

Oppgave11d.vhd

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

entity MANCHESTER_tb is
end MANCHESTER_tb;

architecture TB_MANCHESTER of MANCHESTER_tb is

  component MANCHESTER
    port (
      RESET    : in  std_logic;
      CLK      : in  std_logic;
      DATA_RDY : in  std_logic;
      SDI      : in  std_logic;
      MANU     : out std_logic);
  end component;

  -- component ports
  signal RESET    : std_logic := '-';
  signal CLK      : std_logic := '0';
  signal DATA_RDY : std_logic := '0';
  signal SDI      : std_logic;
  signal MANU     : std_logic := '-';

  signal BOUNDARY : std_logic := '0'; --used to synchronize the verify
                                      --process to the MANU data
  constant N      : integer:=255;
  type TEST_DATA is array (0 to N) of std_logic_vector(7 downto 0);
  signal TEST_INPUT : TEST_DATA;
  constant Period : time := 20 ns;
begin  -- TB_MANCHESTER

  -- component instantiation
  DUT: MANCHESTER
    port map (
      RESET    => RESET,
      CLK      => CLK,
      DATA_RDY => DATA_RDY,
      SDI      => SDI,
      MANU     => MANU);

  -- clock generation
  CLK <= not CLK after Period/2.0;

  -- waveform generation
  STIMULI: process
  begin
    -- insert signal assignments here
    -- Initialize test_input
    for i in 0 to N loop
      TEST_INPUT(i) <= std_logic_vector(to_unsigned(i,8));
    end loop;  -- i

    wait for 200 ns;
    RESET <= '1', '0' after 100 ns;
    wait for 200 ns;
    wait until rising_edge(CLK);
    DATA_RDY <= '1' after 1 ns;
    for i in 0 to N loop
```

```
      BOUNDARY <= not BOUNDARY after Period;
      for j in 7 downto 0 loop
        SDI <= std_logic(TEST_INPUT(i)(j)) after 1 ns;
        wait until rising_edge(CLK);
        if i = N and j =0 then
          DATA_RDY <= '0'  ;
        end if;
        wait until rising_edge(CLK);
      end loop;  -- j
    end loop;  -- i
    wait;
  end process STIMULI;

  --Ikke krav til selvsjekkende testbenk i oppgaven
  VERIFY_OUTPUT: process
    variable bit0, bit1 : std_logic;
    variable error : integer := 0;
    variable CHECK_DATA : TEST_DATA;
  begin
    wait until BOUNDARY = '1';
    wait for Period/2;
    for i in 0 to N loop
      for j in 7 downto 0 loop
        bit0 := MANU;
        wait for Period;
        bit1 := MANU;
        wait for Period;
        --wait for Period;
        if bit0 = '0' and bit1 = '1' then
          CHECK_DATA(i)(j) := '0';
        else
          CHECK_DATA(i)(j) := '1';
        end if;
      end loop;
    end loop;
    wait for 100 ns;
    for i in 0 to N loop
      for j in 7 downto 0 loop
        if CHECK_DATA(i)(j) /= TEST_INPUT(i)(j) then
          error := error + 1;
        end if;
      end loop;
    end loop;
    if error = 0 then
      report "Passed";
    else
      report "Not Passed #errors=" & integer'image(error);
    end if;
  end process VERIFY_OUTPUT;
end TB_MANCHESTER;
```

e)


IDLE_ST
reset='1'
DATA_RDY <= '0'
T
EMPTY='1'
F
FRD
T
LOAD1_ST
LOAD
HELP_CNT_RST
DATA_RDY <= '1'
LOAD0_ST
LOAD
HELP_CNT_RST
DATA_RDY <= '0'
WAIT_ST
HELP_CNT_INC
DATA_RDY <= '1'
HELP_CNT=
14?
T
F
SHIFT_ST
HELP_CNT_INC
SHIFT
DATA_RDY <= '1'
EMPTY='1'
F
FRD