INF3430/INF4431 Høsten 2015

Laboppgave 3

ASM Flytskjema

Bruk av moduler i et system og design av tilstandsmaskin

Del 1. ASM flytskjema

Oppgave 1.

I denne oppgaven skal vi lage ASM-flytskjemaer for en FSM som styrer en brusautomat (det skal ikke lages VHDL kode).

Viktig å huske fra Zwolinski side 86-87 at tegnet ”=” betyr å sette et signal til en verdi som varer kun en klokkeperiode og blir resatt til default verdi (vanligvis 0) etter at klokkeperioden er over (eventuelt sette alltid verdi!). Tegnene ”<-” brukes til gi et signal eller en vektor (dvs. mange bit) verdi etter at klokkeperioden er slutt (dvs. i neste klokkeperiode) og beholder verdien inntil den får en ny verdi (dvs. blir et register!). I oppgavene under **skal** begge to tilordningene brukes.

a)
Den første versjonen av maskinen er kun en første prototype hvor vi har forenklet kontrollen slik at brusen er gratis. Entiteten til FSM er oppgitt under og det skal lages en **Moore FSM** som gir et aktivt høyt signal ”servesoda” som varer en klokkeperiode på 20 ns (i.e. 50MHz) når signalet ”reqserve” er aktivt høyt en klokkeperiode (input signalet ”reqserve” er alltid aktivt høyt bare en klokkeperiode om gangen; all ”prell” er allerede fjernet). Signalet ”led” skal være aktivt høyt i 4 sekunder etter at ”reqserve” har vært aktiv.

```
entity sodamachine_simple is
port
(
clk       : in  std_logic;                    --Clock
rst       : in  std_logic;                    --Asynchron reset
reqserve  : in  std_logic;                    --Request serve (i.e. start)
servesoda : out std_logic;                    --Serve soda (i.e. dispence)
led       : out std_logic                     --LED active high 4 seconds
);
end sodamachine_simple;
```

b)
I den neste versjonen av brusautomaten koster brusen ”sodacost” kroner og den skal kunne gi tilbake vekslepenger som 10 eller 1 kronestykker hvor henholdsvis signalene ”coinback10” og deretter coinback1” er aktivt høye så mange klokkeperioder som antall mynter tilsier (dvs. ”coinback1” høyt i 4 perioder blir 4 kronestykker) etter at ”led” har lyst i 4 sekunder. Vi regner nå at det er uendelig med mynter i automaten så det vil alltid kunne gis veksel og signalene ”sodacost” og ”money” kan ikke være større enn 200 kroner som er høyeste summen automaten aksepterer i form av sedler eller mynter. Hvis det er lagt på for lite penger før ”reqserve” trykkes blir alle pengene gitt tilbake og led lyser, men ”servesoda” blir ikke aktivt. Den nye entiteten er oppgitt under og det skal nå lages en **Mealy FSM**.

```
entity sodamachine_advanced is
port
(
clk        : in  std_logic;                   --Clock
rst        : in  std_logic;                   --Asynchron reset
sodacost   : in  std_logic_vector(7 downto 0); --Soda cost (i.e. unit cost)
reqserve   : in  std_logic;                   --Request serve (i.e. start)
money      : in  std_logic_vector(7 downto 0); --Money
servesoda  : out std_logic;                   --Serve soda (i.e. dispence)
led        : out std_logic;                   --LED active high 4 seconds
coinback1  : out std_logic;                   --1 kroner back
coinback10 : out std_logic                    --10 kroner back
);
end sodamachine_advanced;
```

Del 2. Moduler i et system og design av tilstandsmaskiner

Innledning

Målet med denne laboppgaven er at dere skal få erfaring med å lage tilstandsmaskiner i VHDL. For hver deloppgave skal du følge samme designflyt som i laboppgave 2.

Å kunne styre/regulere posisjonen til en robotarm er en grunnleggende funksjon i enhver robot. Vi skal i denne oppgaven lage en IP(Intelectual Property)-modul som skal utgjøre posisjonsregulatoren i en robot. Vi kommer til å benytte denne i flere sammenhenger, i denne oppgaven og i neste der vi skal benytte den sammen med en mikroprosessor.

Som en del av prosessen lærer man

- å sette sammen et større system
- å jobbe med timing constraints
- å synkronisere eksterne og interne signaler
- å benytte ferdige simuleringsmodeller i en testbenk

Litt om posisjonsregulering

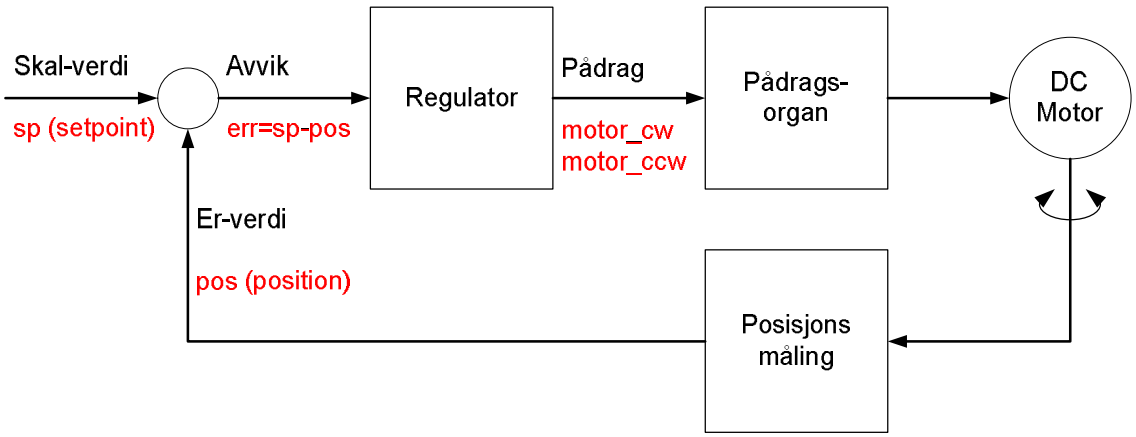


Figure 1. Reguleringssløyfe

Når vi skal styre/regulere posisjonen til en robotarm er det vanlig å gjøre dette ved å benytte en elektrisk motor i hvert ledd.
Vi ønsker å lage en regulering av posisjonen. I en regulering har vi alltid en tilbakekopling med en måling av det vi ønsker å regulere. Hovedtankegangen er: skal man ha kontroll på hvor man skal må man vite hvor man er. I en styring mangler denne tilbakekoplingen (tar sikte og lukker øynene).

Figuren over viser en slik reguleringssløyfe og virkemåten er som følger:

1. Vi påtrykker en skal-verdi, sp (eng. demand eller setpoint) som er en ønsket verdi for posisjonen
2. sp sammenlignes med en målt verdi av posisjonen, pos. Differansen *err=sp-pos* kalles avviket.
3. err brukes som input til regulatoren som behandler dette med en eller annen matematisk formel (som er gunstig for det man skal regulere) og gir ut et resultat som kalles pådrag. Pådraget benyttes til å styre pådragsorganet.
4. Pådragsorganet er gjerne en kraftoverføringssenhet, som er styrt fra regulatoren og som gir en kraft som er stor nok til å skape en bevegelse. I vårt tilfelle består dette av effekttransistorer som er koblet rett på motoren. I andre system kan det være hydrauliske komponenter, pneumatiske osv.

Posisjonsmåler

I vårt system består posisjonsmåleren av en såkalt *"optical shaft encoder"*. Man kan lese mer om denne hos http://www.usdigital.com/products/encoders/incremental/rotary/shaft/s4/.
Disse har en aksling og en fast montert del. Akslingen kobles sammen med motorens aksling. Ut av encoderen får man to pulser, a og b, som er 90 grader faseforskjøvet. Hvilke signal som ligger først er avhengig av retningen akslingen roterer. I oppkoblingen vår har vi følgende relasjon mellom retning og faseforskjellen mellom signalene a og b.

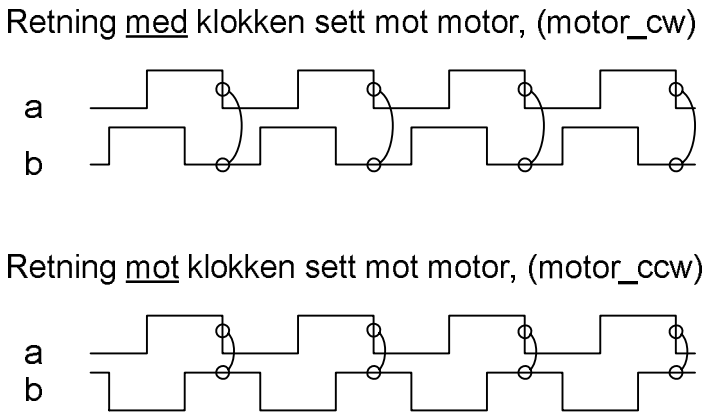


Figure 2. Posisjonsmåling. Deteksjon av perioder og retning

Encoderen vi benytter gir ut 360 prellfrie perioder av signalene a og b pr. omdreining. Dersom vi detekterer signalene som i figuren over gir det oss en oppløselighet på 1grad pr.puls. Vi kan dekke et område fra 0-127 grader ved å benytte en 8-bits *signed* datatype for posisjonen, pos[1].

Regulator

Ofte benyttes regulatorer av typen PID (Proporsjonal-Integral-Differensial). Pådraget kan da uttrykkes som en sum av tre ledd: Pådrag = P + I + D, der P=Kp*err, I= Ki*∑err og D = Kd*Δerr. Spesielt interesserte kan lese mer om dette f.eks. i applikasjonsnote fra Atmel:
http://atmel.nl/dyn/resources/prod_documents/doc2558.pdf
I vårt eksempel skal vi gjøre vesentlige forenklinger, men som fungerer veldig bra i praksis. Regulatoren vår skal fungere på følgende måte:
Dersom avviket er større enn null, full fart med urviseren. Dersom avviket er mindre enn null, full fart mot urviseren. Dersom avvik er null så er aksel i ønsket posisjon og motoren skal være i ro.
Dette fungerer bra fordi motoren vår har en gearkasse med flere hundre ganger utveksling og er på den måten en treg regulering. Å gjøre reguleringen på denne måten tilsvarer en P-regulator med veldig stor forsterkning.

Motor og pådragsorgan

I vårt eksempel skal vi styre motoren med å benytte to signaler, `motor_cw` (clock wise) og `motor_ccw` (counter clock wise) etter følgende tabell:

Table 1. Motorsignaler

Pådrag		**Motor bevegelse**	**Motordriver**
motor_cw	**motor_ccw**		**Lysdioder**
0	0	Ro	Slukket
1	0	Med klokken	Grønt lys
0	1	Mot klokken	Rødt lys
1	1	Ro (ulovlig komb)	Slukket

[1] Ved å detektere hver flanke av a og b kunne man fått en oppløselighet på 0.25 grader/flanke. Men for oss synes det passende å benytte 8-bits tall siden dette gir oss mulighet til å benytte brytere på kortet på en grei måte til å lage setpoint, sp.

Disse signalene er ført ut på expansjonskonnektoren A2 på Spartan kortet. Til A2 konnektoren kobles et ferdig innkjøpt kort, *Module Interface Board (MIB)*. MIB er utstyrt med åtte 6-pins konnektorer, J1-J8. Hver av disse har 3.3V spenning og jord pluss 4 signaler. Det betyr at vi har akkurat plass til signalene `motor_cw`, `motor_ccw` til pådragsorganet og posisjonssignalene, `a` og `b`. Et MIB kort kan med andre ord benyttes til å styre 8 ledd i en robot.
Hver konnektor Ji(i=1,2,..,8) kan kobles til et kort ved navn *Motordriver*. Appendix A inneholder et detaljert kretsskjema for Motordriver kortet. Motordriver inneholder konnektorer for å koble til posisjonsmåler, motor og et 12V powersupply. Det er også lagt inn et galvanisk skille på Motordriverkortet for å unngå problemer med jording. Pådragssignalene er koblet inn på en ferdiginnkjøpt modul bestående bl.a av en IC med effektransistorer. Denne utgjør vårt pådragsorgan. Man kan lese mer om denne modulen på:
http://www.sparkfun.com/commerce/product_info.php?products_id=8907

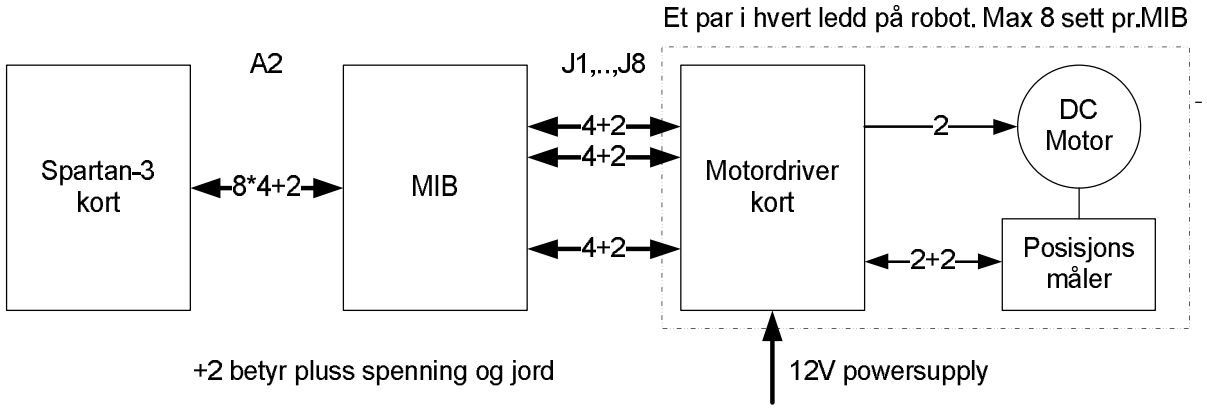


Figure 3. Oversikt over systemet

Figuren over viser en oversikt over systemet. I appendix A finner man en detaljert tabell over sammenhengen mellom signaler, pinnenummer på Ji konnektorene og pinnenummer på FPGA. Det er også tatt med bilder som viser oppkoblingen mellom de forskjellige modulene.

Rapport.

For hver deloppgave skal dere levere:

1. VHDL-kildefil(er).
2. VHDL-testbenk.
3. *Do*-fil(er) for simulering i Modelsim.
4. ".ucf" constraints filer dersom relevant.
5. Place & Route report/Static Timing report der man har gjennomført Place og route.

En kortfattet rapport som oppsummerer hva som er gjort, med problemer/utfordringer.
I tillegg skal dere demonstrere det ferdige systemet i deloppgave 4 for gruppelærer på laben.

Alle innleverte VHDL-filer skal følge navnreglene for vhdl-filer og retningslinjene for indentering som beskrevet i kokeboken.

Oppgave 2.

Konstruer en posisjonsmåler ved hjelp av en tilstandsmaskin og en teller. Benytt vedlagte entitet pos_meas_ent.vhd, og ta utgangspunkt i følgende ASM-flytdiagram. Husk at signalene a og b er asynkrone og trenger derfor synkroniseringsflip-flop'er. Ved asynkron/synkron reset skal posisjonen resettes til 0. Posisjon skal variere mellom 0-127, og man må sørge for at den aldri går utenfor dette området.
HINT: det kan være nyttig å benytte vedlagt simuleringsmodell for motor og posisjonssensor.

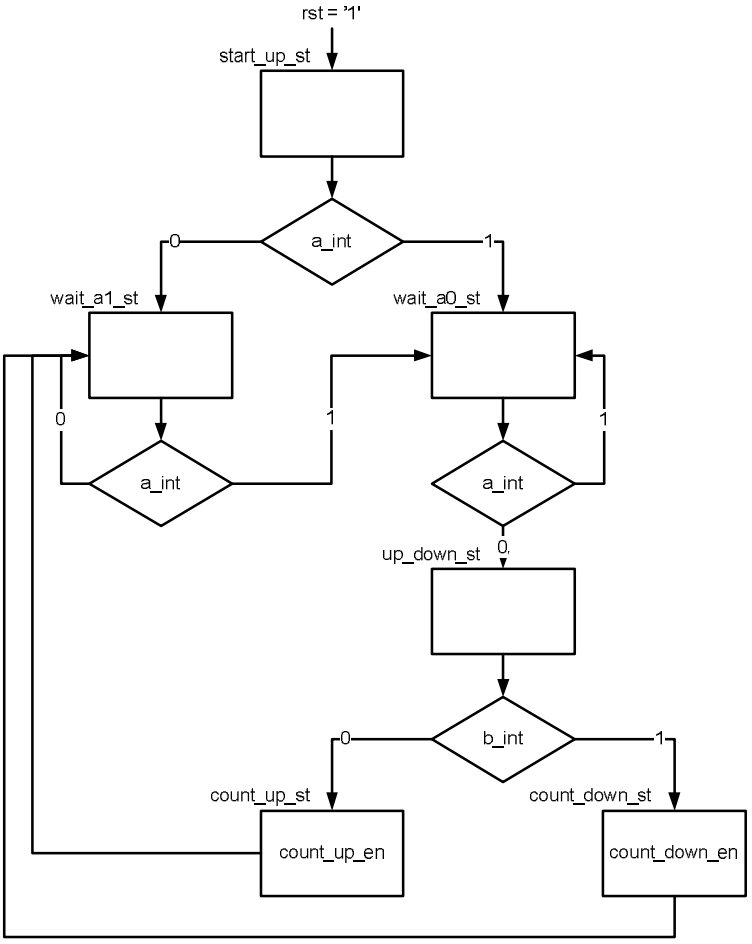


Figure 4.ASM flytskjema for posisjonsmåler

Simuler modulen først grundig med en testbenk. Test modulen deretter på testkortet. Det er spesielt viktig at den aldri går utenfor måleområdet mellom [0,127].

Hint: For å teste pos_meas på kortet kan det være lurt å instantiere den i en test modul, test_pos_meas, som benytter BTN1 og BTN0 for å skape bevelgelse den ene eller andre veien.

Oppgave 3.

Implementer en regulator, p_ctrl, med utgangspunkt i følgende ASM-flydiagram. Bruk vedlagte entitet i p_ctrl_ent.vhd. Simuler modulen grundig med en testbenk. Denne modulen skal bare simuleres og ikke testes på testkortet.

Inngangssignalene `pos` og `sp` trenger synkroniserings flip-flop'er ettersom `sp` er et eksternt signal og `pos` er et internt signal som har en annen klokke enn master klokken til resten av systemet i påfølgende deloppgave 3 og 4 hvor p_ctrl skal brukes.

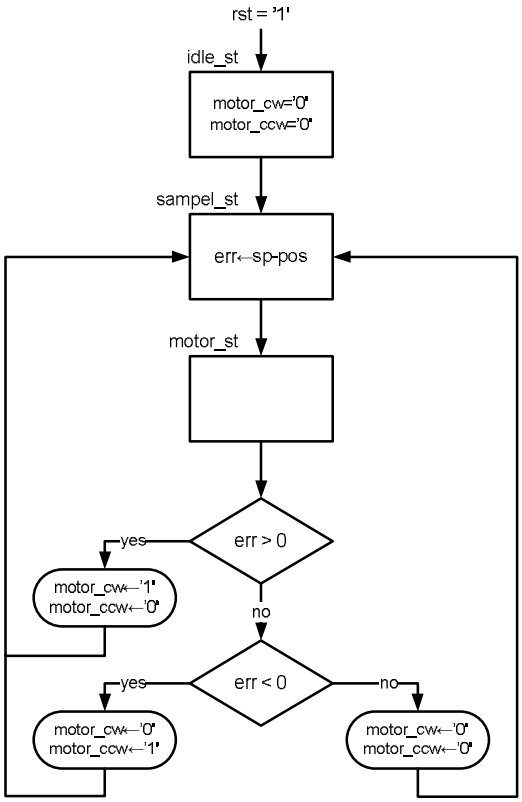


Figure 5. ASM flytdiagram for p_ctrl

Oppgave 4.

Integrer posisjonsmåleren fra a) og regulatoren i fra b) i en ny modul, pos_ctrl. Benytt vedlagt entitet, pos_ctrl_ent.vhd. Legg merke til at p_ctrl skal ha en egen klokke, `mclk_div`, som ikke er master clock. Det skal være mulig å tvangskjøre motor med eller mot klokken ved å benytte to signaler: `force_cw` og `force_ccw`. En multiplekser velger hvilke motorsignaler som skal gjelde etter følgende sannhetstabell:

Table 2. Tvangskjøring av motor (F1)

force_cw	force_ccw	motor_cw	motor_ccw
0	0	cw (fra p_ctrl)	ccw (fra p_ctrl)
0	1	0 (force_cw)	1 (force_ccw)
1	0	1 (force_cw)	0 (force_ccw)
1	1	cw (fra p_ctrl)	ccw (fra p_ctrl)

Tvangskjøring er viktig når man skal nullstille posisjonsmålerene i en kjent posisjon, som typisk vil være en endeposisjon.
Masker bort øverste bit i sp slik at verdien til sp aldri kan gå utover området 0-127 før den kobles til p_ctrl (sp <= '0' & sp(6 downto 0)).

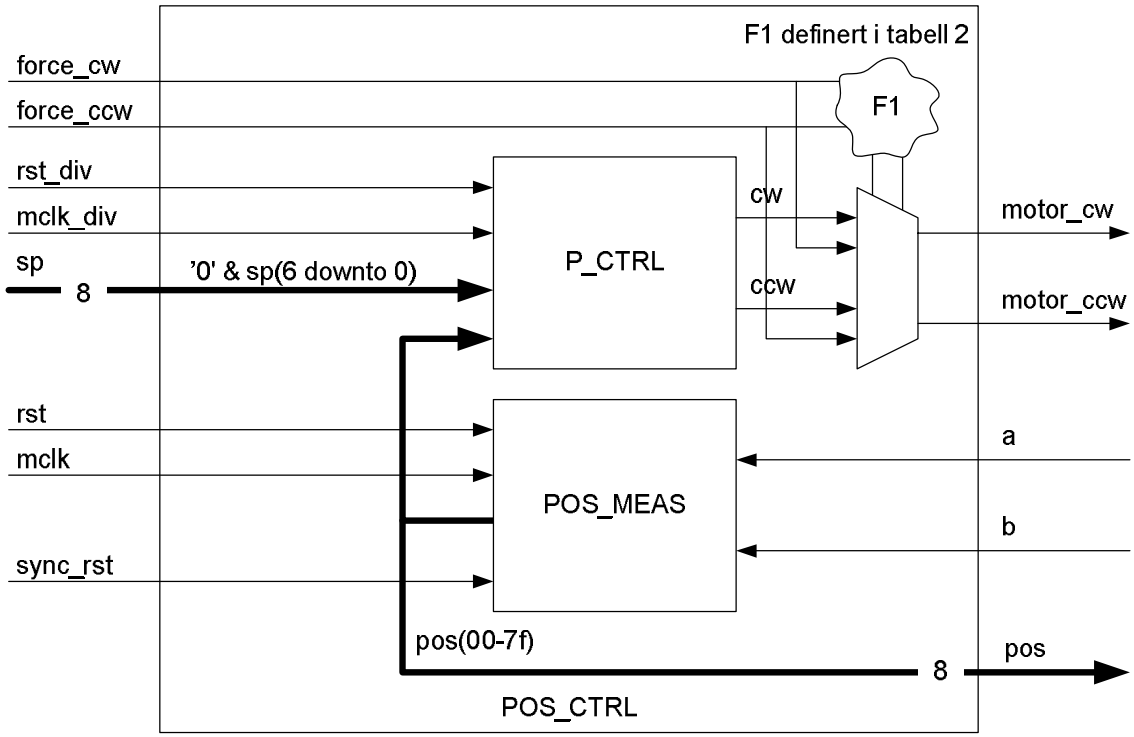


Figure 6. pos_ctrl IP

pos_cntr skal ikke implementeres i FPGA, men simuleres grundig med en testbenk.

Oppgave 5.

Koble sammen et system, pos_seg7_cntr, bestående av pos_ctrl, sjusegmentkontrolleren fra laboppgave 2, samt en Clock Reset Unit (CRU).

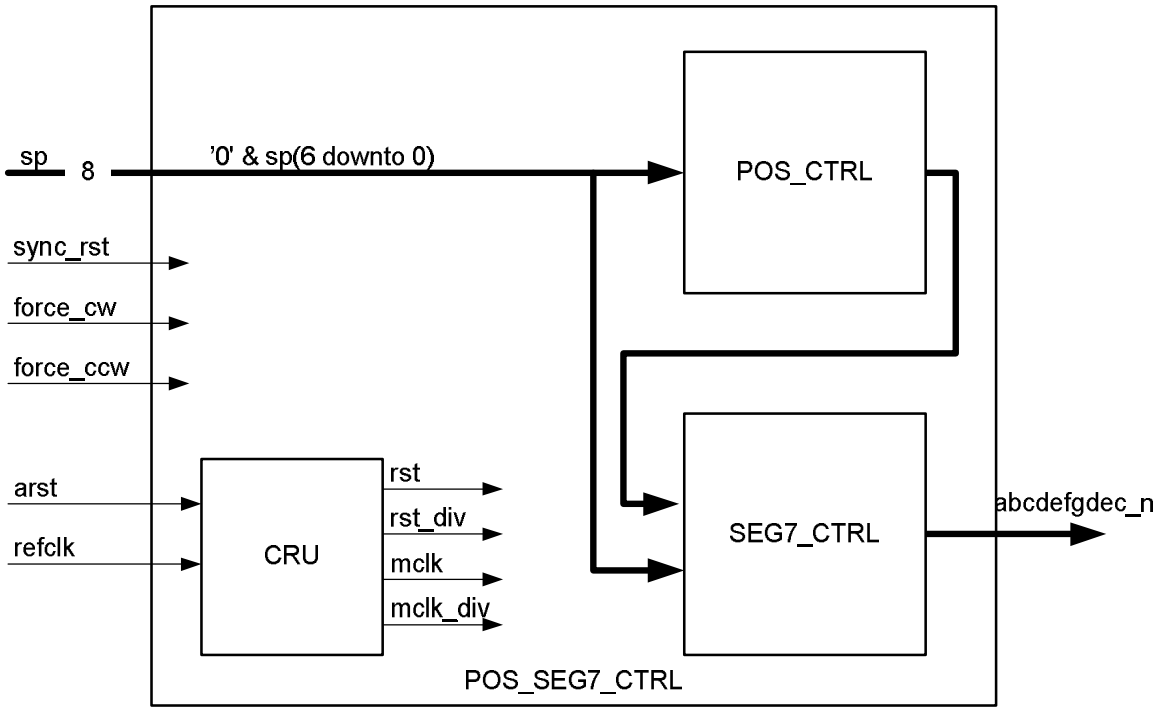


Figure 7. pos_seg7_ctrl

`sp` skal vises i DISP3 og DISP2 og posisjon, `pos`, i DISP1 og DISP0. Setpoint, `sp`, kobles til bryterene SW6-SW0. `force_cw` og `force_ccw` kobles til henholdsvis BTN1 og BTN0.

`sync_rst` kobles til BTN2. Klokker og reset tas fra Clock Reset Unit, CRU. P_ctrl skal benytte klokken `mclk_div` og reset signalet `rst_div.`

Lag en constraints fil (.ucf) som i tillegg til pinnenummer constraints har timing constraints for alle input og output signaler i tillegg til det eksterne klokkesignalet `refclk` og den interne klokken `mclk_div` sine timing constraint. Se forelesningsslidene hvor et eksempel på en .ucf fil er vist.

Hint: Velg Synthesis => Properties => Synthesis Options => Keep Hierarchy => Yes. Da blir hierarkiet fra RTL-koden bevart og klokkenavn og path blir ikke optimalisert bort.

Verifiser at designet er constrained. Eventuelle unconstrained pather skal dokumenteres.

Hint: Analyse gjøres ved ISE timing analyse. I ISE skal man for Implement Design / Properties sette Post-Place and Route til Verbose og for eksempel 20 i både Error report og Unconstrained report (i tillegg bør "Fastest Path" velges). Ved å søke på ”unconstrained” i timing rapporten finnes alle eventuelle unconstrained paths. Se forelesningsslides om constraining.

Pos_seg7_ctrl skal både simuleres og implementeres i FPGA. Den skal også benyttes i laboppgave 4 sammen med bl.a. en mikroprosessor.

LYKKE TIL!

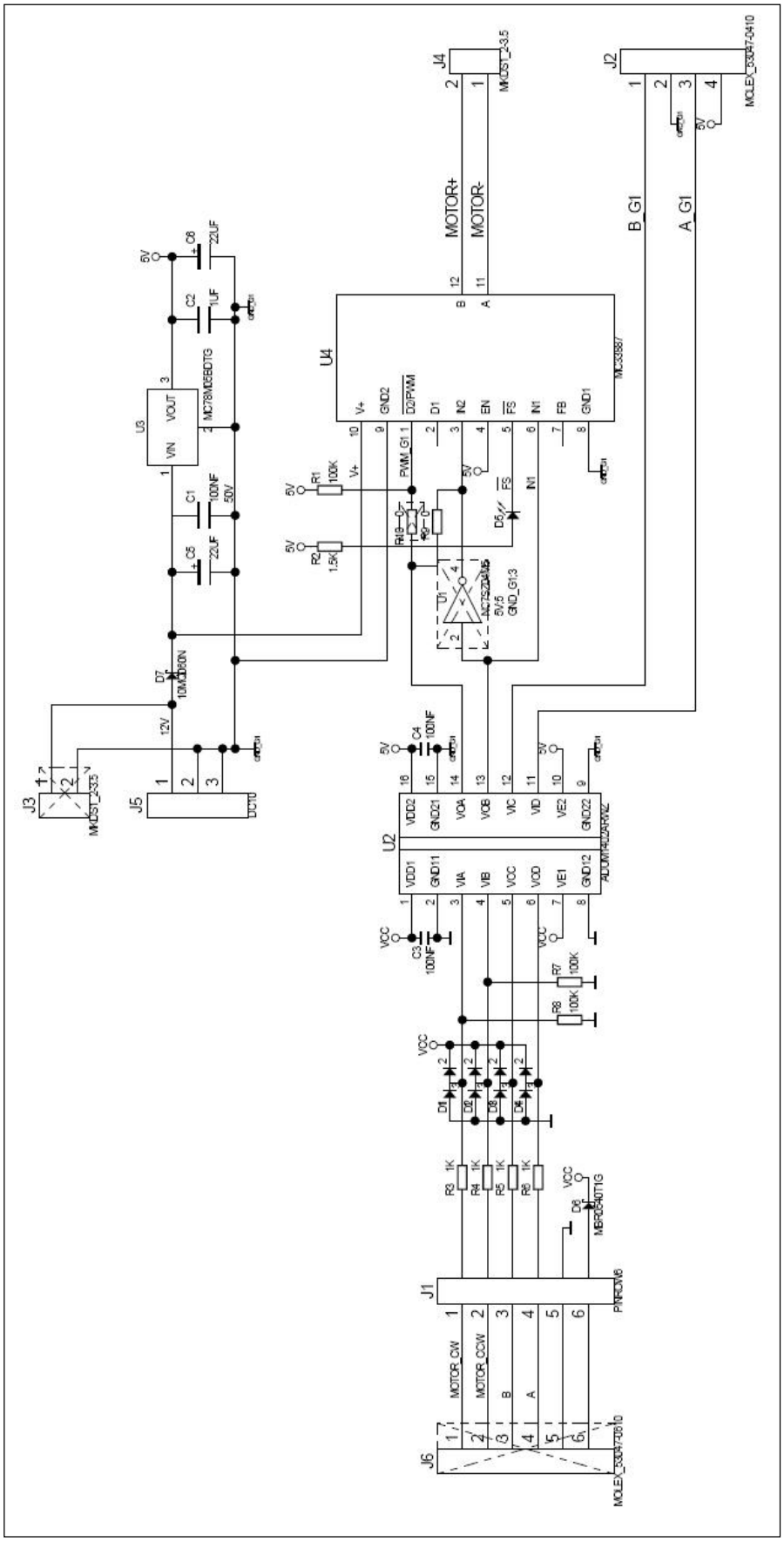
Appendix

Pinnenummer

Table 3. Motorstyringssignaler og pinnenummer

	JX pin	A2 pin	FPGA pin	Signal
J1	1	4	E6	motor_cw
	2	6	C5	motor_ccw
	3	5	D5	a
	4	8	C6	b
J2	1	7	D6	motor_cw
	2	10	C7	motor_ccw
	3	9	E7	a
	4	12	C8	b
J3	1	11	D7	motor_cw
	2	14	C9	motor_ccw
	3	13	D8	A
	4	16	A3	B
J4	1	15	D10	motor_cw
	2	18	A4	motor_ccw
	3	17	B4	A
	4	20	A5	B
J5	1	19	B5	motor_cw
	2	22	B7	motor_ccw
	3	21	B6	A
	4	24	B8	B
J6	1	23	A7	motor_cw
	2	26	A9	motor_ccw
	3	25	A8	A
	4	28	A10	B
J7	1	27	B10	motor_cw
	2	30	B12	motor_ccw
	3	29	B11	a
	4	32	B13	b
J8	1	31	A12	motor_cw
	2	34	B14	motor_ccw
	3	33	A13	a
	4	35	D9	b

Motordriverkort



Tilkoblinger mellom Spartankort ->MIB->Motordriverkort