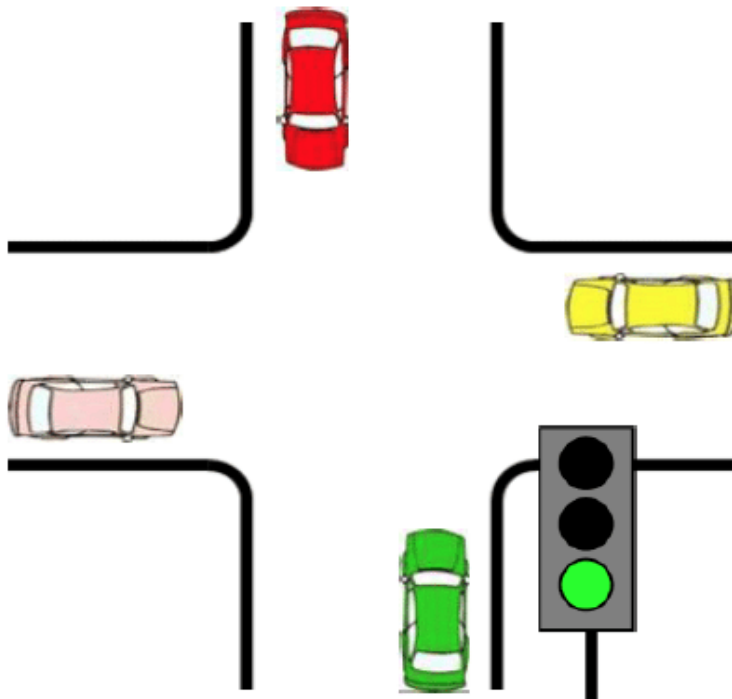


# DIGITAL TEKNIKK

## *RAPPORT*

### *Lyskryss*



OSLOMET

Av

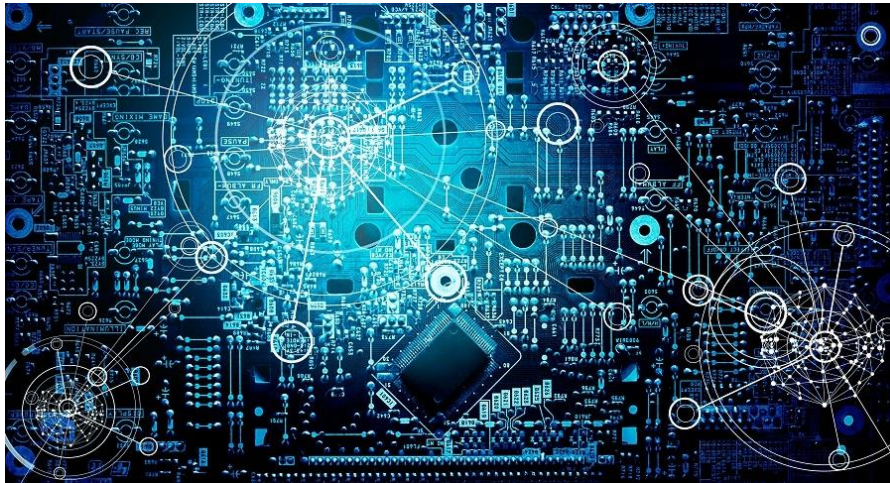
*Shoaib Ahmed*

*S319527*

## Sammendrag

Hensikten med dette prosjektet er først og fremst å bruke kunnskapen vi har fått fra laboratorietimene vi har hatt på skolen. Dette prosjektet går innom alle temaene vi har gjennomført på laboratoriet, slik at vi kan vise hva vi har lært i løpet av semesteret. I tillegg til kunnskap så var oppgavens hensikt å lage et lyskryss som skal kunne kontrollere biltrafikken. Dette lyskrysset består av en hovedvei og en sidevei. For å få dette til å fungere så har jeg laget fire forskjellige VHDL-filer, som er koblet sammen. Jeg har laget en tilstandsmaskin, en timer, en prescaler og en teller. Jobben til disse 4 VHDL-filene er:

1. Tilstandsmaskinen styrer trafikklysene på hovedvei og sidevei.
2. Timeren styrer transisjonene mellom de forskjellige tilstandene.
3. Telleren brukes som tidsreferanse for timeren.
4. Prescaleren reduserer kortets oscillatorfrekvens til 1 Hz. Dette svarer til en periode på et sekund.



*Fig 1.0*

## Innhold

Sammendrag .....	2
Innledning .....	4
Fremgangsmåte .....	4
DEL 1 .....	5
DEL 2 .....	11
DEL 3 .....	13
DEL 4 .....	15
DEL 5 .....	16
Konklusjon .....	18
Kildehenvisning .....	19

## Innledning

"Hvordan lage et lyskryss som skal bestå av en hovedvei og sidevei på Quartos programvaren?"

Dette var problemstillingen for prosjektet. Formålet med dette prosjektet er å lage et fungerende lyskryss ved å samle sammen 4 VHDL-filer som man lager underveis. Disse skal til slutt kobles sammen.

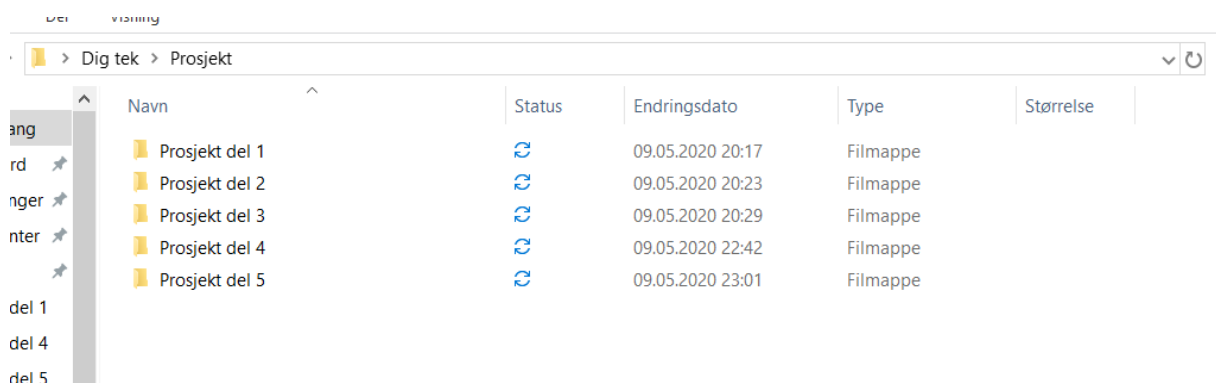
## Fremgangsmåte

For dette prosjektet, var intensjonen å bruke et **DE1-SoC Altera** for å teste underveis om dette prosjektet fungerer, men dette kunne vi ikke gjøre grunnet den pågående situasjonen med Covid-19.

Dette prosjektet ble besvart ved å bruke programvaren, *Quartus Prime Lite Edition*

Jeg har valgt å sette sammen 4 punkter fordi det virker mest naturlig for å vise besvarelsen min. Disse 4 punktene er *Teori*, *Kravspesifikasjon*, *Systemløsning* og *Systemprøving*. Dette kommer til å bli forklart ved at jeg bryter ned del for del der jeg tar med det obligatoriske.

Jeg startet først ved å lage en mappe på skrivebordet som heter *Dig.tek prosjekt*. Inni mappen lagde jeg 5 mapper til de 5 forskjellige delene. Dette er obligatorisk å gjøre før du starter med prosjektet, slik at du ikke får alle filene i en mappe som kan forstyrre simuleringen til en gitt del i prosjektet.



Navn	Status	Endringsdato	Type	Størrelse
Prosjekt del 1		09.05.2020 20:17	Filmappe	
Prosjekt del 2		09.05.2020 20:23	Filmappe	
Prosjekt del 3		09.05.2020 20:29	Filmappe	
Prosjekt del 4		09.05.2020 22:42	Filmappe	
Prosjekt del 5		09.05.2020 23:01	Filmappe	

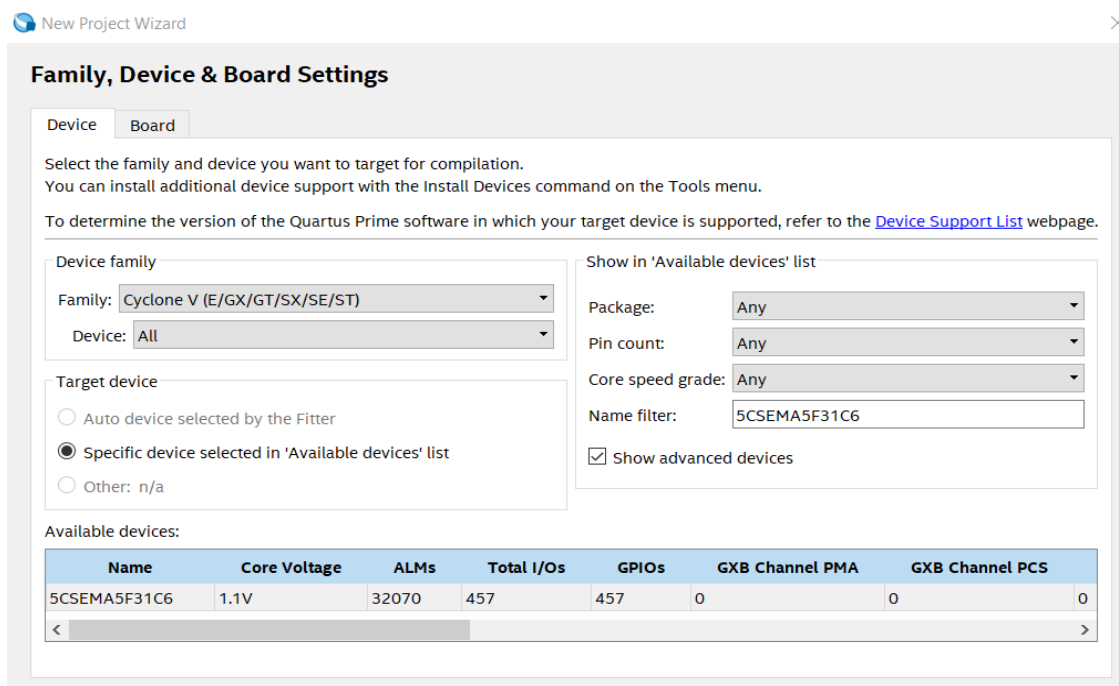
Fig 2.0: Mappefordelingen.

NB! Del 1 blir beskrevet detaljert på hvilke innstillinger man velger og hvordan man starter. Dette gjør jeg ikke videre grunnet at man bruker de samme innstillingene på starten.

## DEL 1

### Tilstandsmaskin

Tilstandsmaskinen er til for å styre trafikklysene på hovedvei og sidevei. Den er programmert slik at den er begrenset til 6 forskjellige tilstander (State 1-State 6). Jeg startet denne oppgaven med å lage et nytt prosjekt i mappen, prosjekt del 1. Deretter gikk jeg videre til *family, device & board settings* for å velge dette kortet *5CSEMA5F31C6*.



**Family, Device & Board Settings**

Device | Board

Select the family and device you want to target for compilation.  
You can install additional device support with the Install Devices command on the Tools menu.  
To determine the version of the Quartus Prime software in which your target device is supported, refer to the [Device Support List](#) webpage.

Device family

Family: Cyclone V (E/GX/GT/SX/SE/ST)

Device: All

Target device

☐ Auto device selected by the Fitter

☒ Specific device selected in 'Available devices' list

☐ Other: n/a

Show in 'Available devices' list

Package: Any

Pin count: Any

Core speed grade: Any

Name filter: 5CSEMA5F31C6

☒ Show advanced devices

Available devices:

Name	Core Voltage	ALMs	Total I/Os	GPIOs	GXB Channel PMA	GXB Channel PCS
5CSEMA5F31C6	1.1V	32070	457	457	0	0

Fig 2.1.0: Family, Device & Board Settings, med riktig device som blir brukt i oppgaven.

Når dette er valgt, trykker man på neste så kommer denne siden opp:

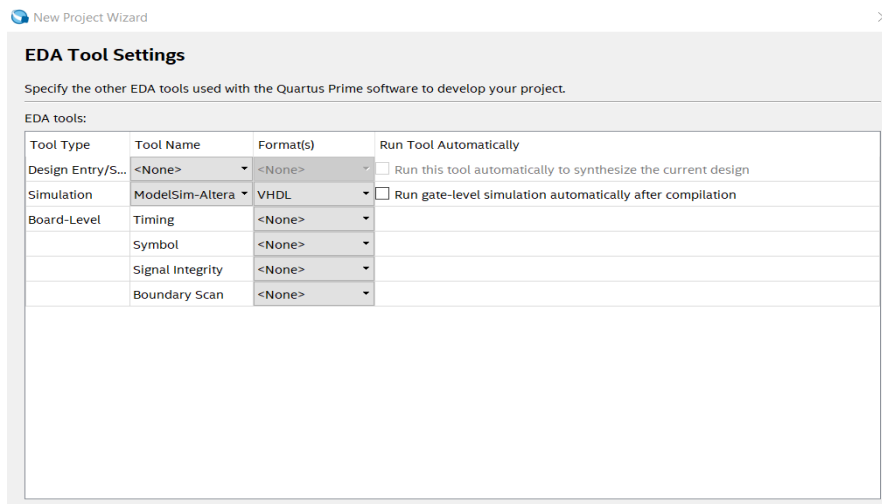


Fig 2.1.1: Her velger man **Modelsim-Altera** som **Tool Name**, og **VHDL** som **Format**.

Når dette er gjennomført, skal man trykke på *file* helt øverst i hjørnet, new og deretter *State Machine File* for å starte på oppgaven.

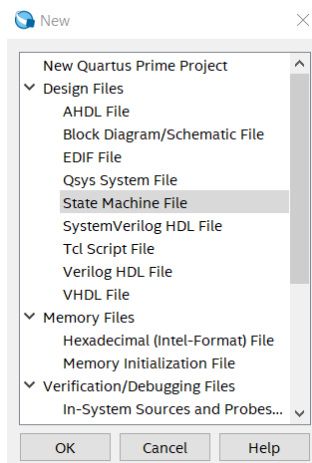


Fig 2.1.2: Filmeny. Her velger du filen som oppgaven krever. I dette tilfelle en **State Machine File**.

Når **State Machine File** er åpnet, skal man først legge til 6 stater. Disse statene skal ligge vertikalt vedsiden av hverandre og de skal kobles sammen ved å bruke **Transition Tool**. Når dette er gjort, skal man sette inn 6 inputs og 6 outputs. Inputsene skal døpes time1 til time6 for alle de 6. Outputsene skal man sette som **m\_green, m\_yellow, m\_red, s\_green, s\_yellow, s\_red**. M står for Main road og S står Side road. Etter at dette er gjort skal man gå inn på **State Machine Wizzard**. På første fanen under **General**, er det viktig å huke på **Asynchronous** og huke av på **reset is active- high**. Man huke av på **Asynchronous** for at denne tillater systemet å kjøre flere elementer samtidig, mens med **Synchronous** lar den kjøre

et element om gangen. Deretter trykker på man på **Transistions** for å skrive ned de forskjellige inputsene etter stigende rekkefølge på statsene. F.eks fra **State1** til **State2**, skal du skrive **time1** også fortsetter man videre. Når dette er gjort skal man gå videre til **Actions** for å skrive ned alle verdiene til outputsene i de forskjellige statene. Dette ble gjort ved å følge dette skjemaet her:

Til-stand	Hovedvei Grønn	Hovedvei Gul	Hovedvei Rød	Sidevei Grønn	Sidevei Gul	Sidevei Rød	Varighet 5 bit	Varighet 6 bit
State1	1	0	0	0	0	1	12 sek	32 sek
State2	0	1	0	0	0	1	2 sek	3 sek
State3	0	0	1	0	1	1	2 Sek	3 Sek
State4	0	0	1	1	0	0	12 sek	20 sek
State5	0	0	1	0	1	0	2 sek	3 sek
State6	0	1	1	0	0	1	2 sek	3 sek

Fig 2.1.3: Denne forteller oss om alle verdiene til Outputsene i forhold til alle 6 stater.

Når man skrevet ned alle statsene og alle verdiene for de forskjellige outputsene, så trykker man på apply og får ut dette på **State Machine Filen**.

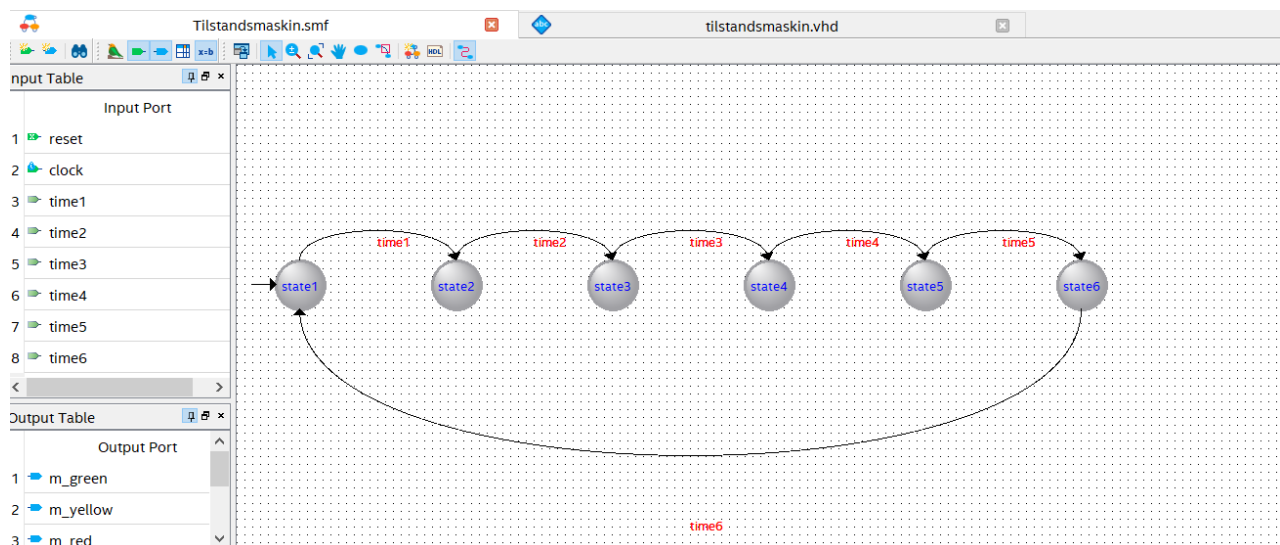


Fig 2.1.4: Visuell representasjon av alle 6 stater koblet sammen med de forskjellige inputsene (time1-6) skrevet mellom hver state. Dette resulterer en Tilstandsmaskin.

Nå er det klart for å compilere tilstandsmaskinen. Dette er svært viktig å gjøre før man lager en VHDL. Hvis dette ikke blir gjort er det en stor sannsynlighet for at den ikke får med seg inputsene og outputsene, ergo ikke får generert VHDL-filen ordentlig. Etter at den er compilert for første gang, får man ut mange error og warnings, og det er fordi man ikke har

laget en VHDL. Dette er fordi uten en VHDL, så vet ikke tilstandsmaskinen hva den skal gjøre. En VHDL inneholder koder som vil fortelle eksakt hvordan et annet system skal fungere og hva den skal gjøre av oppgaver. Når VHDL filen er på plass, er det på tide å kompilere alt sammen, og da sitter du igjen med 0 error.

```
Text Editor - C:/Users/shoa/OneDrive/Skrivebord/Dig tek/Prosjekt/Prosjekt del 1/Tilstandsmaskin - Tilstandsmaskin - [tilstandsmaskin.vhd]
File Edit View Project Processing Tools Window Help

18 LIBRARY ieee;
19 USE ieee.std_logic_1164.all;
20
21 ENTITY Tilstandsmaskin IS
22 PORT (
23     reset : IN STD_LOGIC := '0';
24     clock : IN STD_LOGIC;
25     time1 : IN STD_LOGIC := '0';
26     time2 : IN STD_LOGIC := '0';
27     time3 : IN STD_LOGIC := '0';
28     time4 : IN STD_LOGIC := '0';
29     time5 : IN STD_LOGIC := '0';
30     time6 : IN STD_LOGIC := '0';
31     m_green : OUT STD_LOGIC;
32     m_yellow : OUT STD_LOGIC;
33     m_red : OUT STD_LOGIC;
34     s_green : OUT STD_LOGIC;
35     s_yellow : OUT STD_LOGIC;
36     s_red : OUT STD_LOGIC
37 );
38 END Tilstandsmaskin;
39
40 ARCHITECTURE BEHAVIOR OF Tilstandsmaskin IS
41     TYPE type_fstate IS (state1,state2,state3,state4,state5,state6);
42     SIGNAL fstate : type_fstate;
43     SIGNAL reg_fstate : type_fstate;
44 BEGIN
45     PROCESS (clock,reset,reg_fstate)
46     BEGIN
47         IF (reset='0') THEN
48             fstate <= state1;
49         ELSIF (clock='1' AND clock'event) THEN
50             fstate <= reg_fstate;
51         END IF;
52     END PROCESS;
53
54     PROCESS (fstate,time1,time2,time3,time4,time5,time6)
55     BEGIN
56         m_green <= '0';
57         m_yellow <= '0';
58         m_red <= '0';
59         s_green <= '0';
60         s_yellow <= '0';
61         s_red <= '0';
62         CASE fstate IS
63             WHEN state1 =>
64                 IF ((time1 = '1')) THEN
65                     reg_fstate <= state2;
66                     -- Inserting 'else' block to prevent latch inference
67                 ELSE
68                     reg_fstate <= state1;
69                 END IF;
70
71                 s_red <= '1';
72                 s_yellow <= '0';
73                 s_green <= '0';
74                 m_red <= '0';
75                 m_yellow <= '0';
76
77                 m_green <= '1';
78             WHEN state2 =>
79                 IF ((time2 = '1')) THEN
80                     reg_fstate <= state3;
81                     -- Inserting 'else' block to prevent latch inference
82                 ELSE
83                     reg_fstate <= state2;
84                 END IF;
85
86                 s_red <= '1';
87                 s_yellow <= '0';
88                 s_green <= '0';
89                 m_red <= '0';
90                 m_yellow <= '1';
91
92                 m_green <= '0';
93             WHEN state3 =>
94                 IF ((time3 = '1')) THEN
95                     reg_fstate <= state4;
96                     -- Inserting 'else' block to prevent latch inference
97                 ELSE
98                     reg_fstate <= state3;
99                 END IF;
100
101                 s_red <= '1';
102                 s_yellow <= '1';
103                 s_green <= '0';
104                 m_red <= '1';
105                 m_yellow <= '0';
106                 m_green <= '0';
107
108                 s_red <= '1';
109                 s_yellow <= '1';
110                 s_green <= '0';
111                 m_red <= '1';
112                 m_yellow <= '0';
113                 m_green <= '0';
114
115                 s_red <= '1';
116                 s_yellow <= '1';
117                 s_green <= '0';
118                 m_red <= '1';
119                 m_yellow <= '0';
120                 m_green <= '0';
```



```

120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188

    WHEN state4 =>
    IF ((time4 = '1')) THEN
        reg_fstate <= state5;
        -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= state4;
    END IF;

    s_red <= '0';
    s_yellow <= '0';
    s_green <= '1';
    m_red <= '1';
    m_yellow <= '0';
    m_green <= '0';
    WHEN state5 =>
    IF ((time5 = '1')) THEN
        reg_fstate <= state6;
        -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= state5;
    END IF;

    s_red <= '0';
    s_yellow <= '1';
    s_green <= '0';
    m_red <= '1';
    m_yellow <= '0';
    m_green <= '0';
    WHEN state6 =>
    IF ((time6 = '1')) THEN
        reg_fstate <= state1;
        -- Inserting 'else' block to prevent latch inference
    ELSE
        reg_fstate <= state6;
    END IF;

    s_red <= '1';
    s_yellow <= '0';
    s_green <= '0';

    m_red <= '1';
    m_yellow <= '1';
    m_green <= '0';
    WHEN OTHERS =>
        m_green <= 'X';
        m_yellow <= 'X';
        m_red <= 'X';
        s_green <= 'X';
        s_yellow <= 'X';
        s_red <= 'X';
        report "Reach undefined state";
    END CASE;
END PROCESS;
END BEHAVIOR;

```

Fig 2.1.5: Dette er VHDL koden for Tilstandsmaskinen.

Når VHDL filen er ferdig, er det veldig viktig å sette denne som **Top Level Entity**. Dette er for å gi beskjed til Quartos Programvaren at VHDL filen er det som styrer programmet.

Nå er det på tide å simulere waveformen. Man åpner en **University program VWF** og setter inn alle inputs og outputs.

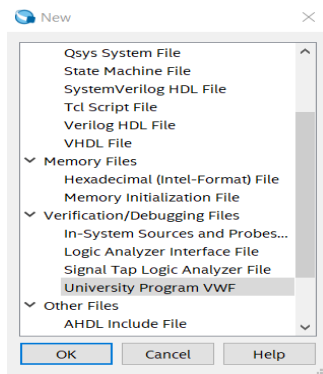


Fig 2.1.6: *University Program VWF* bruker man for å lage Waveform.

Når dette er gjort, får du ut denne:

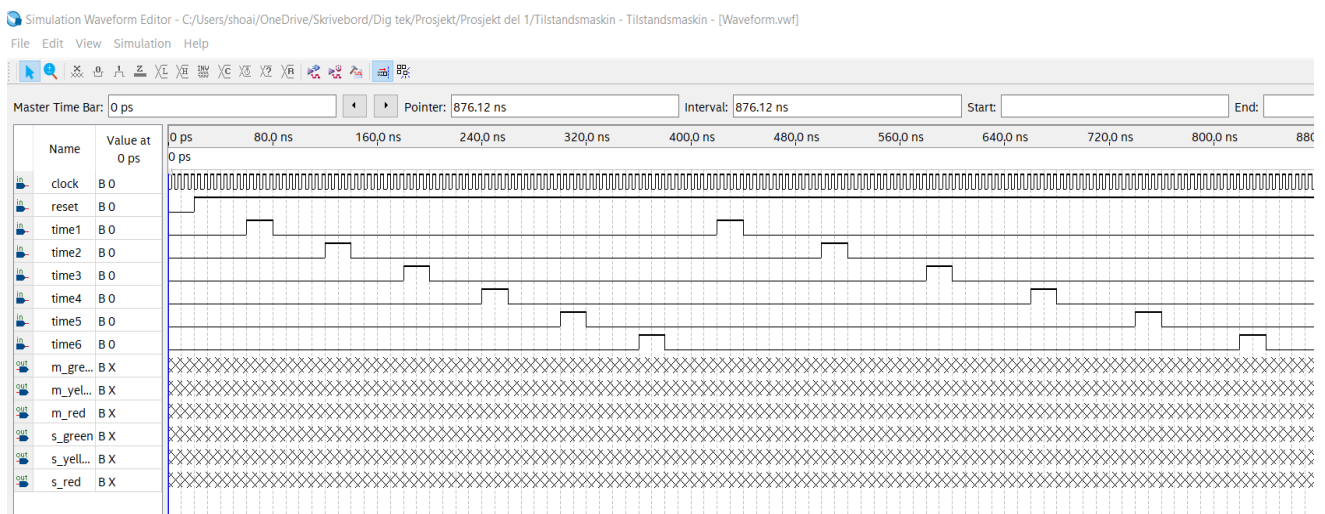


Fig 2.1.7: Waveformen før den er simulert. Illustrert med alle inputsene og outputsene.

Når du simulerer denne waveformen, får du dette som resultat:

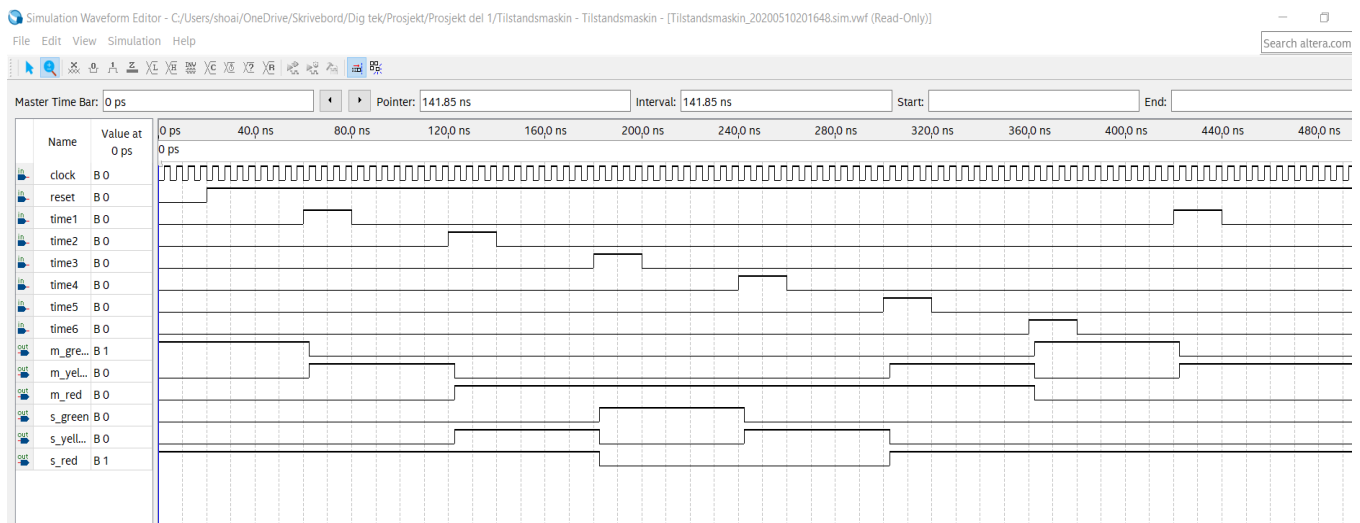


Fig 2.1.8: Waveformen etter den er simulert.

## DEL 2

### Timer

Timeren sin jobb er å la tilstandsmaskinen til å skifte tilstand til riktig tidspunkt. Timeren skal også styre transisjonene mellom de forskjellige tilstandene. Det første man gjør er å lage et nytt prosjekt med de samme innstillingene, og deretter opprette en VHDL-fil med de bibliotekene som er nødvendig for at programmet skal forstå kodene som blir skrevet inn.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5
6  entity Timer is
7  generic (n: natural:=6);
8  port(
9  count: in std_logic_vector(n-1 downto 0); -- 6bit counter input-en
10 tr1: out std_logic; --OUTPUT1
11 tr2: out std_logic; --OUTPUT2
12 tr3: out std_logic; --OUTPUT3
13 tr4: out std_logic; --OUTPUT4
14 tr5: out std_logic; --OUTPUT5
15 tr6: out std_logic; --OUTPUT6
16 );
17 end entity;
18
19 architecture behave of Timer is
20 begin
21 process (count) is
22 begin
23 case count is
24 when "100000" =>
25     tr1 <='1';
26 when "100011" =>
27     tr2 <='1';
28 when "100110" =>
29     tr3 <='1';
30 when "111010" =>
31     tr4 <='1';
32 when "111101" =>
33     tr5 <='1';
34 when "000000" =>
35     tr6 <='1';
36 when others =>
37     tr1 <='0';
38     tr2 <='0';
39     tr3 <='0';
40     tr4 <='0';
41     tr5 <='0';
42     tr6 <='0';
43 end case;
44 end process;
45 end architecture;
```

Fig 2.2.1: Dette er VHDL for timeren. Man definerer alle bibliotekene som koden nedunder kan bli forstått. Jeg har definert 6 utganger som er kalt tr1, tr2, tr3, tr4 og tr5 og tr6.

Når denne er lagd, kan man lage waveformen for denne VHDL-filen. Man går inn på *Nodelist* og setter inn: Count [0] –[5] og tr1–tr6.

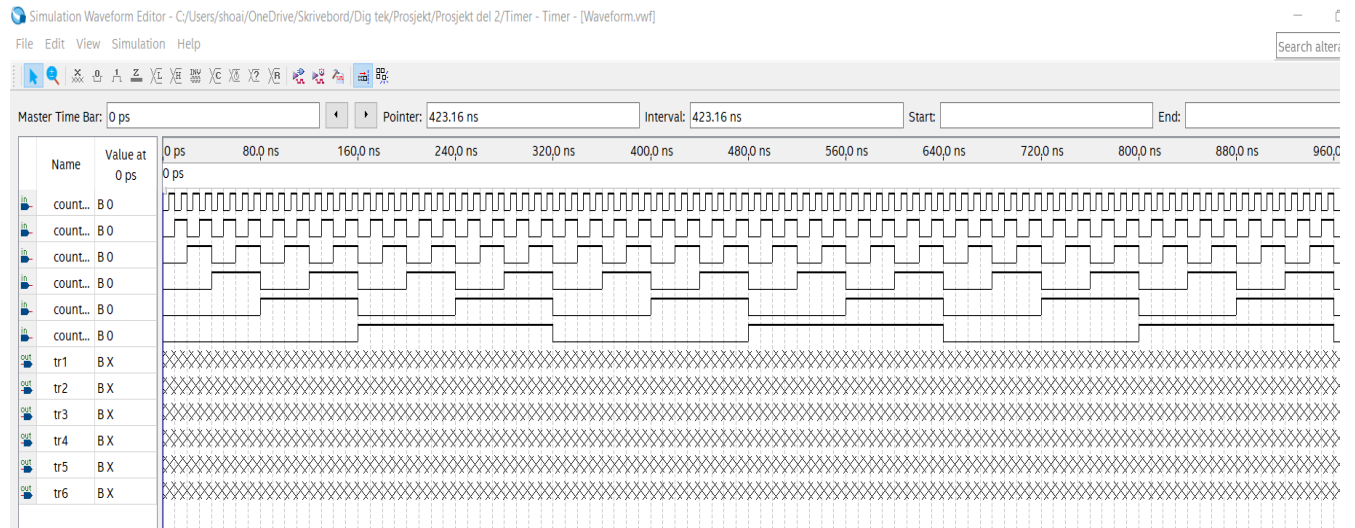


Fig 2.2.2: Usimulert Waveform med inputs med signal.

Deretter simulerer man denne waveformen, slik at resultatet blir som følgende:

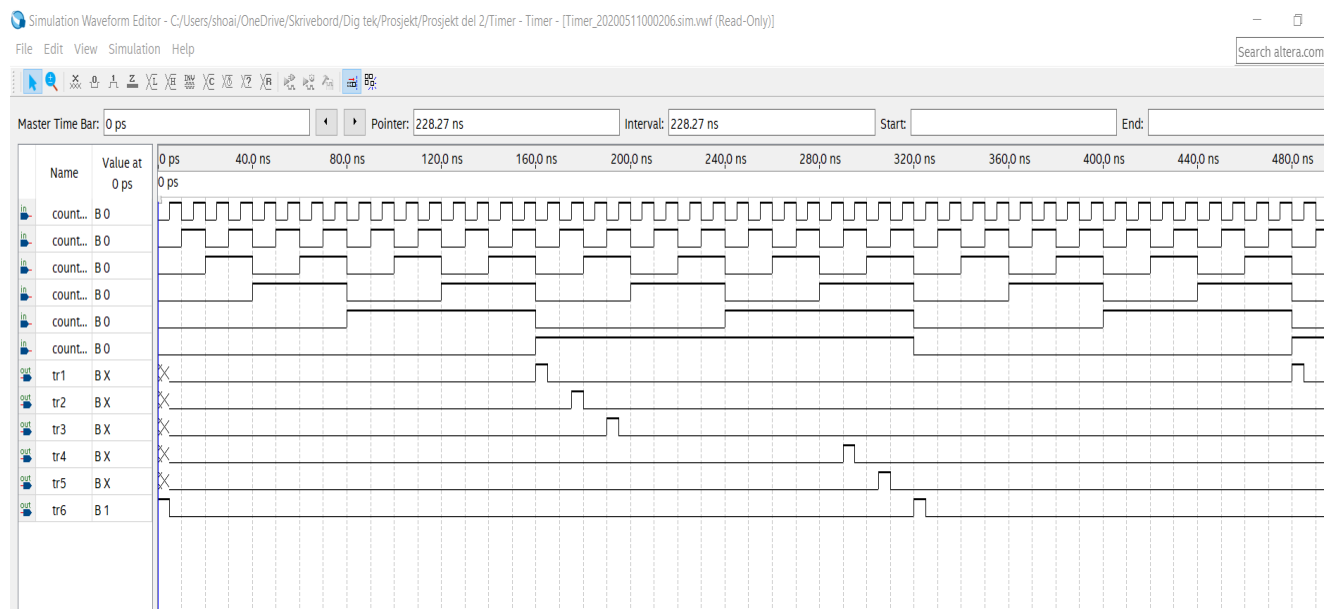


Fig 2.2.3 Simulert Waveform.

## DEL 3

### Prescaler

En prescaler er en elektronisk telle krets som har sitt formål å minke høye elektroniske signaler til lave ved hjelp av heltallsdelen. Prescaleren er der for at den skal ta den grunnleggende klokkefrekvensen og deretter dele den inn med en viss verdi for så til slutt å sende den inn i timeren.

Jeg startet opp Quartus prime med samme kort og samme innstillinger. Det første jeg gjorde var å legge til bibliotekene som er nødvendig for at programmet skal forstå kodene som blir skrevet. Deretter lager jeg en entitet med en utgang som jeg heter *Prescaler\_out* og en inngang som heter *clock\_50*.

I oppgaven er det anbefalt å bruke *generic* i entitet beskrivelsen, dette er for at det skal være mulig å gjøre en endring av verdiene til *prescaler\_value*. En integer blir brukt i forhold hvordan entiteten skal fungere. Denne blir kalt *c*, og den er lik 0. Verdien øker med 1 for hver gang *clock\_50* går fra en lav verdi til høy.

Nedenunder ser du et bilde av VHDL-filen som er generert:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_unsigned.all;
4
5  entity Prescaler is
6  | generic (prescaler_value: Integer := 10);
7  | port (
8  |
9  |   clock_50: in std_logic;
10 |   prescaler_out: out std_logic
11 | );
12 |
13 | end entity;
14
15 | architecture hz of Prescaler is
16 | begin
17 |   process (clock_50)
18 |   | variable c: integer :=0;
19 |   | begin
20 |   |   if rising_edge(clock_50) then
21 |   |   |   c:= c+1;
22 |   |   |   if (c=prescaler_value ) then
23 |   |   |   |   prescaler_out <='1';
24 |   |   |   |   c:=0;
25 |   |   |   |   else
26 |   |   |   |   |   prescaler_out <= '0';
27 |   |   |   |   |
28 |   |   |   |   end if;
29 |   |   |   |   end if;
30 |   |   |   end process;
31 |   end architecture;
32
```

Fig 2.3.1: VHDL-fil for del 3.

Når denne er simulert og kodinga fungerer som den skal uten error, skal man lage en waveform for denne.

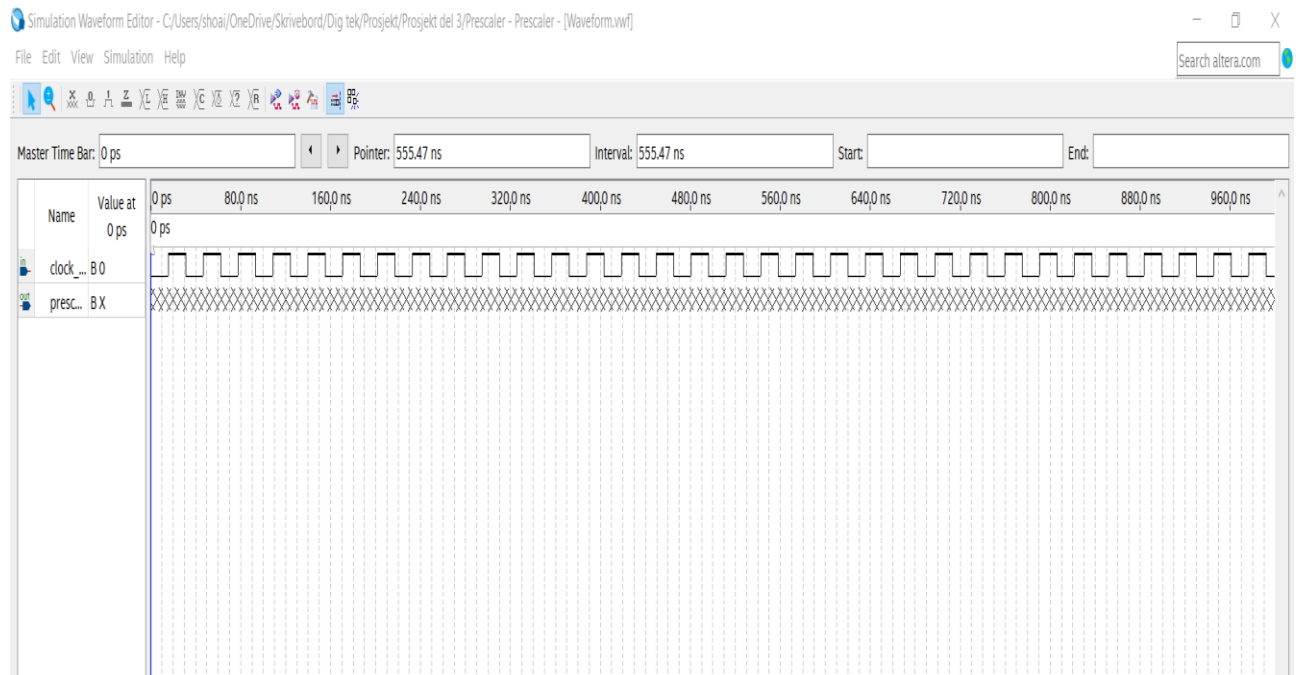


Fig 2.3.2: Waveform før den er simulert.

Deretter skal man simulere denne waveformen slik at resultatet blir slik:

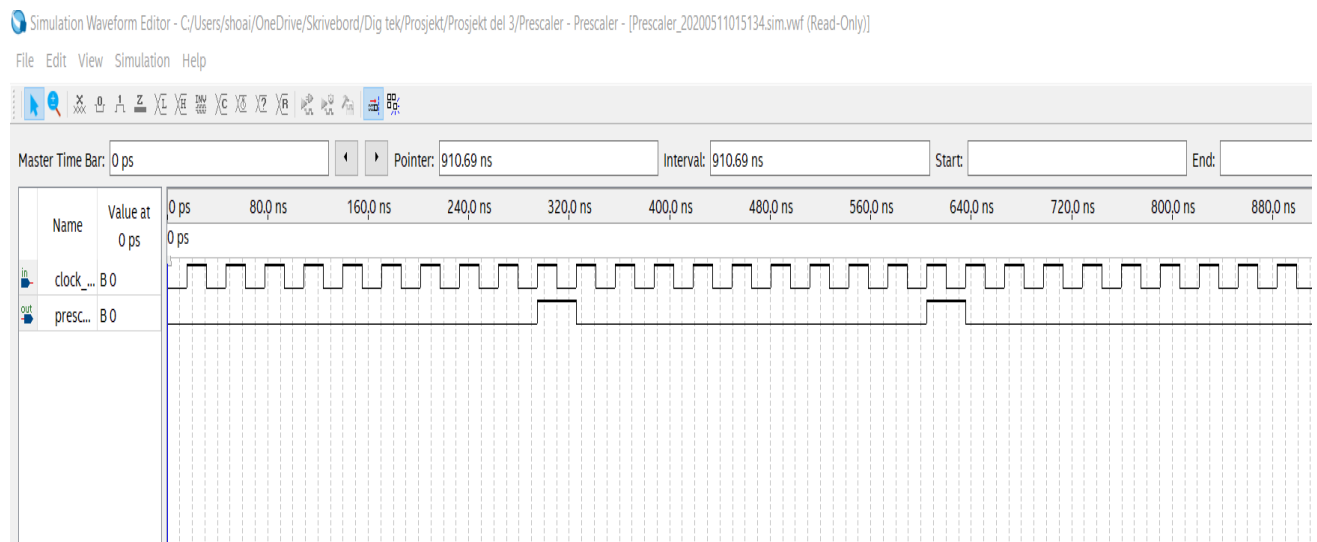
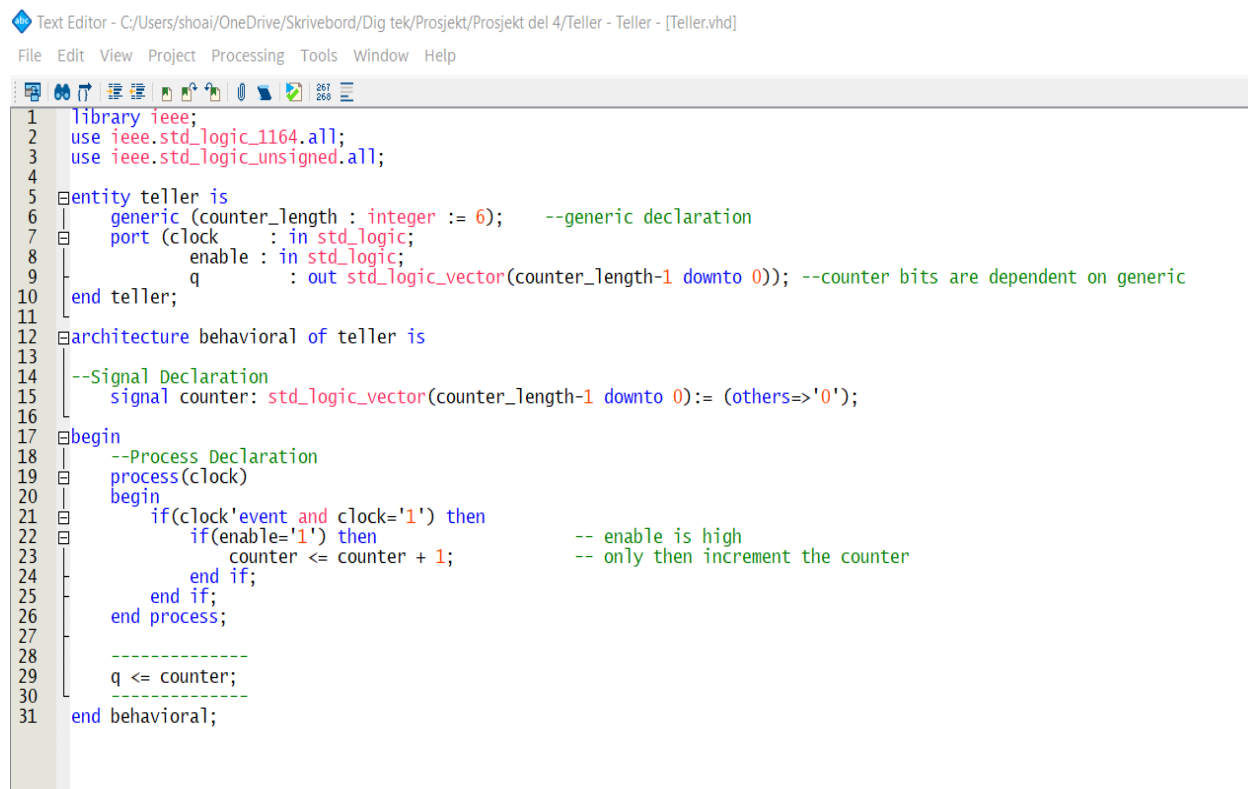


Fig 2.3.3: Resultatet etter å ha simulert Waveformen.

## DEL 4

### Teller

Telleren blir tilført en klokkeinnang på 50Mhz. Når dette skjer vil enablen gjøre at telleren kun øker sin verdi hver gang enable er høy. Dette fører til at telleren vil skifte verdi hvert sekund. For at dette skal bli gjennomført så må man starte et nytt prosjekt med de samme innstillingene som før med en ny VHDL-fil. På VHDL-filen skal det skrives opp bibliotekene man trenger for at koden skal bli forstått av programmet. På filen skal det lages en entitet innang for clock, enable og en utgang "q". Utgang q skal være like lang som bit-telleren er. I dette tilfelle vil det være 6. Når koden er gjennomført, må det kompileres og deretter waveforme den.



```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.std_logic_unsigned.all;
4
5 entity teller is
6     generic (counter_length : integer := 6);    --generic declaration
7     port (clock      : in std_logic;
8          enable      : in std_logic;
9          q           : out std_logic_vector(counter_length-1 downto 0)); --counter bits are dependent on generic
10 end teller;
11
12 architecture behavioral of teller is
13
14     --Signal Declaration
15     signal counter: std_logic_vector(counter_length-1 downto 0) := (others=>'0');
16
17 begin
18     --Process Declaration
19     process(clock)
20     begin
21         if(clock'event and clock='1') then
22             if(enable='1') then
23                 counter <= counter + 1;    -- enable is high
24                                         -- only then increment the counter
25             end if;
26         end if;
27     end process;
28
29     q <= counter;
30
31 end behavioral;
```

Fig 2.4.1: VHDL Kode.

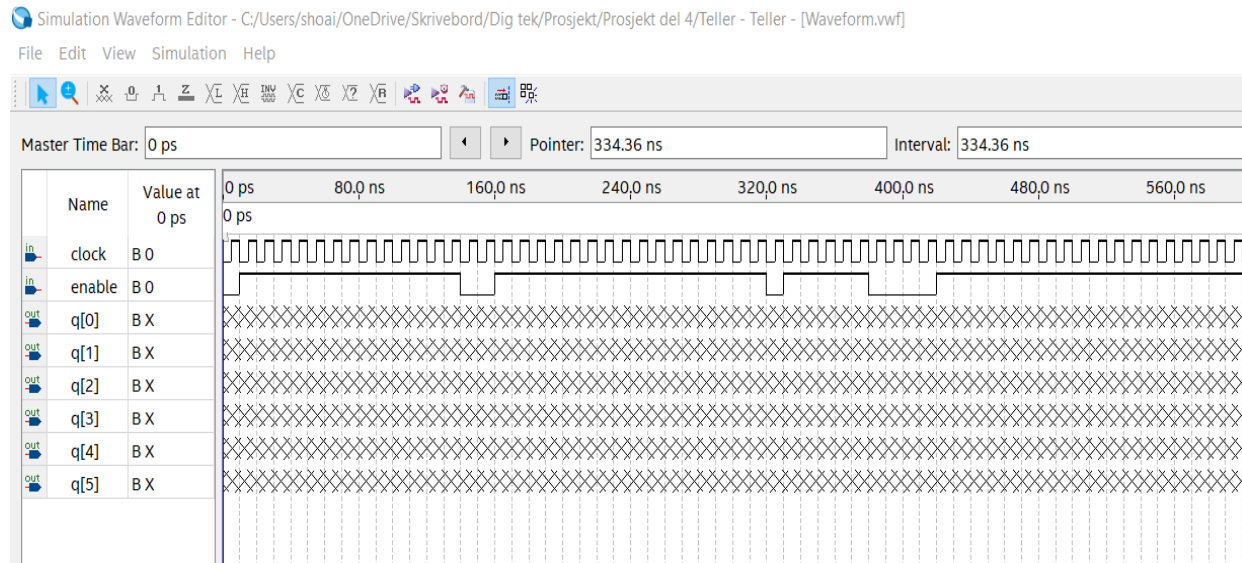


Fig 2.4.2: Usimulert Waveform.

Etter det skal man simulere waveformen.

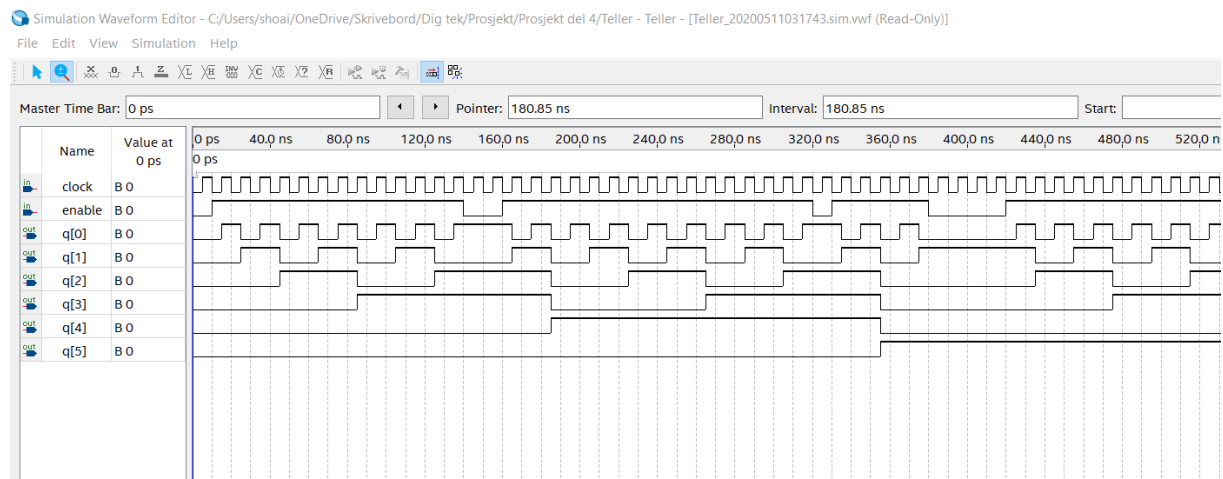


Fig 2.4.3: Simulert Waveform.

## DEL 5

### Lyskryss

Når alle del-oppgavene fra 1-4 er simulert og gjennomført, skal alle VHDL-filene settes sammen i en fil. Det som er viktig her er at man setter alle VHDL-filene i samme mappe som del 5. Dette er fordi at kretsen som man lager skal gjenkjenne de andre VHDL-filene. Når det er gjort skal man åpne et *Block Diagram/Schematic file*. Deretter skal man hente inn hver enkelt VHDL-fil som man kan åpne som en fane vedsiden av. Videre skal man trykke på *file*, *Create/ Update* og deretter *Create Symbol Files For Current File*. Dette gjør man for å



opprette VHDL-filene som et symbol som man legger til på blokkskjemaet. Til slutt skal alle utganger, innganger og koblinger lages slik at du får sluttproduktet.

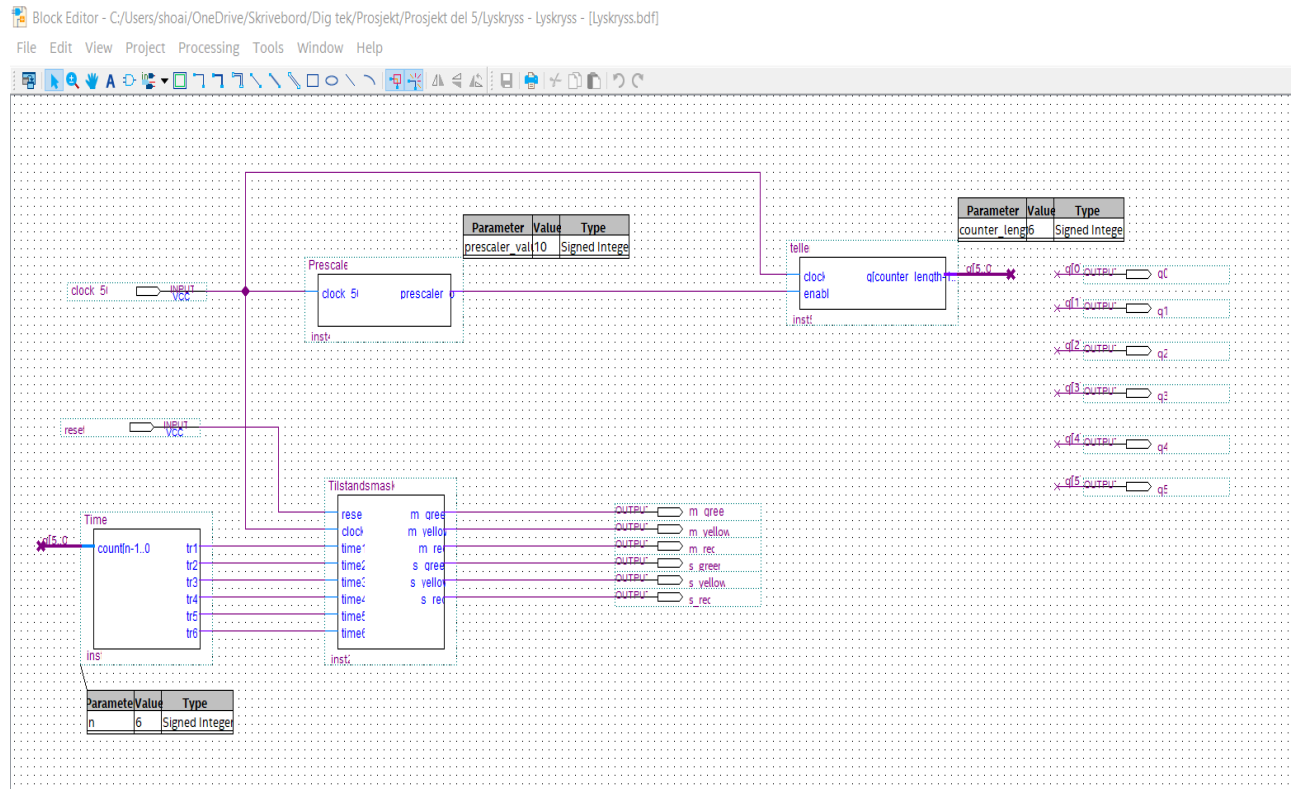


Fig 2.5.1: Blokkskjema med alle VHDL-filene som symbol.

Når dette er gjennomført, foretar man en siste simulering:

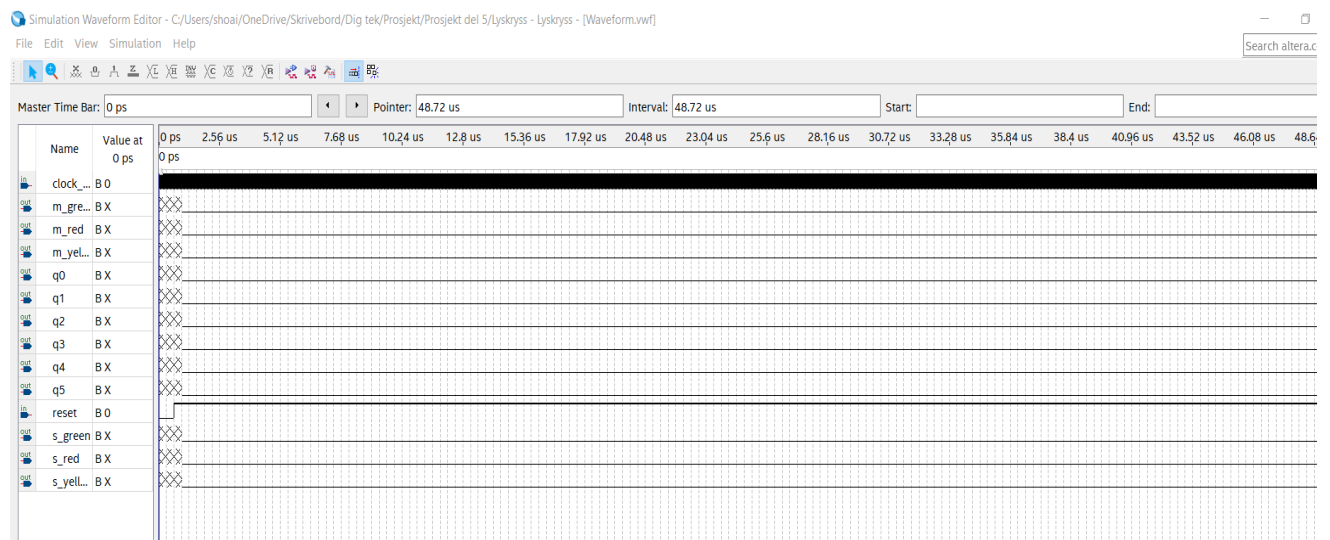


Fig 2.5.2: Waveform ikke simulert.

Deretter skal man simulere siste waveformen. Det man får ut som svar er lyskrysset med en hovedvei og en sidevei.

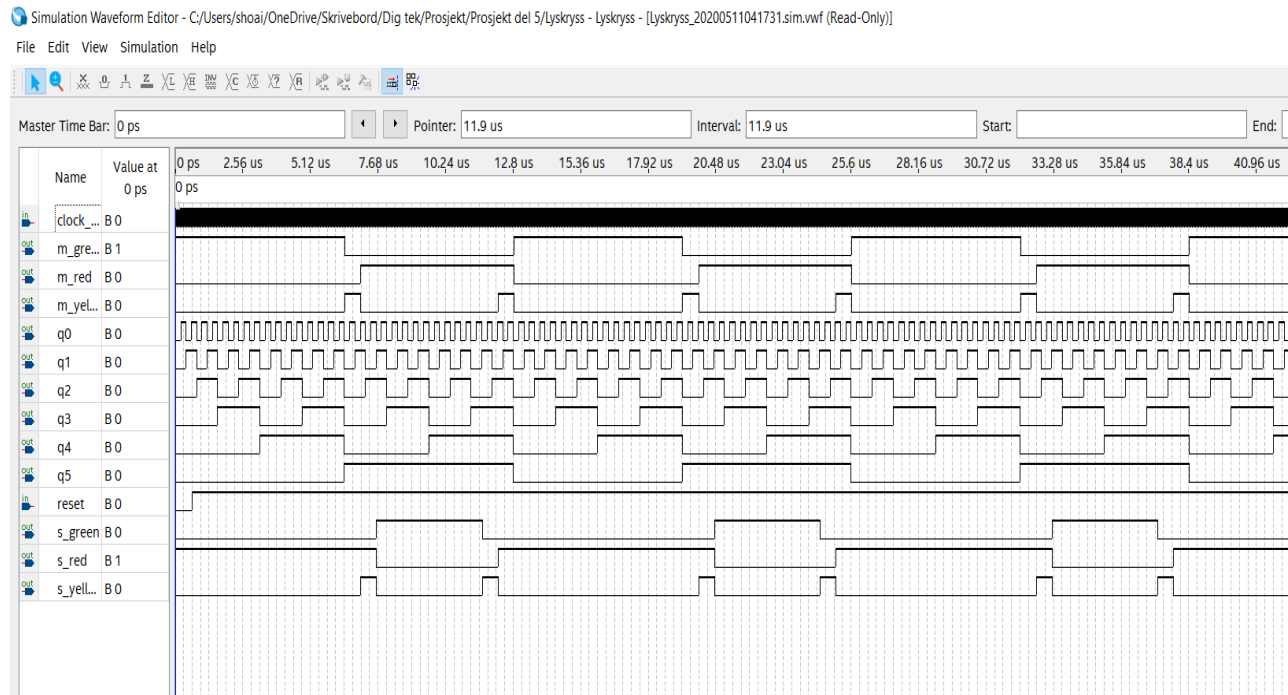


Fig 2.5.3: Ferdig simulert Waveform.

## Konklusjon

Kravene som var stilt av kravspesifikasjonene er fylt og oppgaven er fullført. Lyskryss ble laget, samt programmert slik at trafikklyset lyser på en hovedvei og en sidevei. Dette var en veldig utfordrende oppgave som testet ferdighetene man har lært gjennom semesteret. Ferdighetene ble spesielt testet når man skulle kode i VHDL. Det å jobbe med dette prosjektet har gitt meg en bedre forståelse på hvordan VHDL-koding fungerer og hvor mye den kan brukes til.

## Kildehenvisning

### Forsidebilde:

- [https://www.google.com/search?q=lyskryss&sxsrf=ALeKk00d4p8rk66dWzaJ9pZOD4aKT4ZBRw:1589174712297&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjIm9mRiavpAhVys4sKHfNWBgEQ\\_AUoAXoECAsQAw&biw=1536&bih=754&dpr=1.25#imgrc=NfiZRJaKa2hQXM](https://www.google.com/search?q=lyskryss&sxsrf=ALeKk00d4p8rk66dWzaJ9pZOD4aKT4ZBRw:1589174712297&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjIm9mRiavpAhVys4sKHfNWBgEQ_AUoAXoECAsQAw&biw=1536&bih=754&dpr=1.25#imgrc=NfiZRJaKa2hQXM)

### OsloMet-logo:

- [https://www.google.com/search?q=oslomet&tbm=isch&ved=2ahUKEwiOkcqSiavpAhWlWJoKHcK2CYUQ2-cCegQIABAA&oq=oslomet&gs\\_lcp=CgNpbWcQAziECCMQJzICCAAyAggAMgIIADICCAAyAggAMgIIADIECAAQHjIECAAQHjIECAAQHIDX-QxY3f8MYMmADWgAcAB4AIABPYgBIQOSAQE3mAEAoAEBqgELZ3dzLXdpei1pbWc&sclient=img&ei=uuG4Xo7MClx6QTC7aaoCA&bih=754&biw=1536#imgrc=g4MtfxgOVhbh5M](https://www.google.com/search?q=oslomet&tbm=isch&ved=2ahUKEwiOkcqSiavpAhWlWJoKHcK2CYUQ2-cCegQIABAA&oq=oslomet&gs_lcp=CgNpbWcQAziECCMQJzICCAAyAggAMgIIADICCAAyAggAMgIIADIECAAQHjIECAAQHjIECAAQHIDX-QxY3f8MYMmADWgAcAB4AIABPYgBIQOSAQE3mAEAoAEBqgELZ3dzLXdpei1pbWc&sclient=img&ei=uuG4Xo7MClx6QTC7aaoCA&bih=754&biw=1536#imgrc=g4MtfxgOVhbh5M)

### Fig. 1.0 bilde:

- [https://www.google.com/search?q=laboratorie+data&tbm=isch&ved=2ahUKEwjSjrvShKjpAhUXwaYKHTZ6CsQQ2-cCegQIABAA&oq=laboratorie+data&gs\\_lcp=CgNpbWcQAzoECCMQJzoECAAQHjoGCAAQBRAeOgQIABAYOgYIABAIEB5Q18QJWK\\_SCWD\\_OgIoAHAAeACAAbkBiAH-BZIBBDExLjGYAQGgAQGgAQtnD3Mtd2l6LWltZw&sclient=img&ei=Wkq3XtL5K5eCmwW29KmgDA&bih=754&biw=1536#imgrc=TyhMaTmeCZgyRM](https://www.google.com/search?q=laboratorie+data&tbm=isch&ved=2ahUKEwjSjrvShKjpAhUXwaYKHTZ6CsQQ2-cCegQIABAA&oq=laboratorie+data&gs_lcp=CgNpbWcQAzoECCMQJzoECAAQHjoGCAAQBRAeOgQIABAYOgYIABAIEB5Q18QJWK_SCWD_OgIoAHAAeACAAbkBiAH-BZIBBDExLjGYAQGgAQGgAQtnD3Mtd2l6LWltZw&sclient=img&ei=Wkq3XtL5K5eCmwW29KmgDA&bih=754&biw=1536#imgrc=TyhMaTmeCZgyRM)

### Prescaler:

- <https://en.wikipedia.org/wiki/Prescaler>

### Oppgaveheftet:

- <https://oslomet.instructure.com/courses/17789/files/folder/Lab.%20oppgaver%20og%20prosjekt?preview=1246021>

### Læreboka:

- Digital Elektronikk 2. Utgave – Knut Harald Nygaard.

