FSM ADC
Finite State Machine Analog-To-Digital Converter

Team 1:
202200284 - Danni Raetzel
202304936 - Lucie Mandøe
202200298 - Martin Elken
202200296 - Mikkel Amstrup Brandt Neiiendam
202307576 - Kasper David Arberg

Indholdsfortegnelse

Indledning

I denne laboratorieøvelse vil vi udvikle og implementere en analog-til-digital omformer (ADC) med VHDL, som kan konvertere en analog spænding til en digital værdi, der vises på et 7-segment display. Vi vil anvende Analog Discovery til at generere de analoge spændinger, der skal måles. I denne forbindelse har vi valgt at benytte en counting_ADC til at udføre konverteringen, der kan måle analoge spændingsniveauer og præsentere dem digitalt på et 7-segment display ved hjælp af en multiplexer, der styrer displayets segmenter.

Design / Teori

Dette projekt omfatter design og implementering af en ADC (Analog-To-Digital Converter). En ADC omdanner et analogt input signal til en tilsvarende digital repræsentation. Denne type ADC bruger en ekstern clock til at styre tælle-operationerne. Når en konvertering startes, begynder ADC'en at tælle fra en startværdi, normalt 0 eller en anden forudbestemt værdi. Det analoge inputsignal, f.eks. spændingen, der skal konverteres, er forbundet til en reference, og tælleren inkrementeres, indtil den nåede værdi svarer til det analoge input. Hver inkrementering repræsenterer et trin i ADC'ens opløsning. Når konverteringen er færdig, gemmes den endelige værdi i tælleren. Denne værdi repræsenterer det digitale output fra ADC'en og giver en digital repræsentation af det analoge inputsignal. Opløsningen af ADC'en bestemmes af antallet af bit i tælleren.

FSM-enhed

FSM'en fungerer som hjernen i vores ADC-controller og styrer processen med at konvertere det analoge inputsignal til en digital værdi. Dens formål er at organisere og styre forskellige tilstande og overgange i konverteringsprocessen.

Tilstande

1. **Idle (Ventetilstand):** I denne tilstand er ADC'en inaktiv og venter på at starte en konverteringscyklus.
2. **Start Conversion (Start konvertering):** Denne tilstand aktiveres, når en konvertering skal startes. ADC'en begynder at tælle eller udføre andre nødvendige operationer for at konvertere det analoge input til en digital værdi.
3. **Conversion in Progress (Konvertering pågår):** ADC'en er i gang med at udføre konverteringen. Den overvåger det analoge inputsignal og producerer det tilsvarende digitale output.

4. **Conversion Complete (Konvertering fuldført):** Når konverteringen er afsluttet, går ADC'en ind i denne tilstand. Den sender det digitale output til scaling unit'en og er klar til at starte en ny konvertering, hvis nødvendigt.

Overgangslogik

- **Idle til Start Conversion:** Denne overgang udløses, når startsignalet modtages fra kontrolenheden. ADC'en skifter fra ventetilstand til starttilstand for at initiere konverteringen.
- **Start Conversion til Conversion in Progress:** Når konverteringen er startet, skifter ADC'en til konverteringstilstanden og begynder at udføre konverteringsoperationerne.
- **Conversion in Progress til Conversion Complete:** Når konverteringen er afsluttet, skifter ADC'en til fuldførelsestilstand og er klar til at sende det digitale output.

ADC-enhed

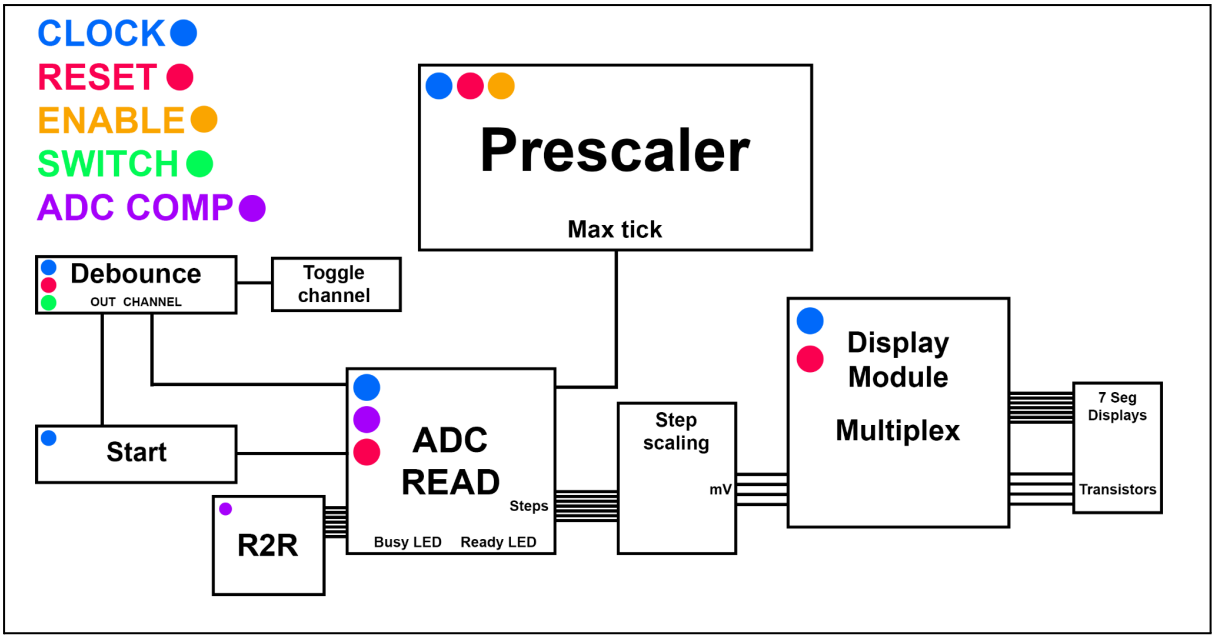
ADC-enheden er designet til at konvertere et analogt input-signal til en tilsvarende digital repræsentation. Den modtager input-signaler som **'clk'** (ur), **'reset'** (nulstilling), **'start'** (start konvertering), **'comp'** (komparationsinput fra R2R), og **'en'** (aktivere ADC'en). Ud over dette giver den også outputsignaler for **'ready'** (klar til ny konvertering) og **'busy'** (beskæftiget med konvertering).

Scaling_unit

Scaling-unit'en konverterer det digitale output fra ADC'en til en repræsentation, der kan vises på 7-segment displayet. Den modtager det digitale output fra ADC'en
Ved hjælp af en skaleringfaktor på 13 mV/bit omdanner scaling-unit'en det digitale output fra ADC'en til en millivolt værdi, der er proportionel med det oprindelige analoge inputsignal. Dette sikrer, at det digitale output på displayet afspejler den faktiske spænding, der blev målt.

Efter skaleringen konverteres den resulterende millivolt værdi til et 4-cifret BCD-format (Binary Coded Decimal). Dette gøres ved at opdele den skalerede værdi i fire separate nibbler, hver repræsenterer et decimalciffer fra 0 til 9. Hvert cifre er derefter klar til at blive vist på 7-segment displayet.

Blokdiagrammer



Figur 1 - Blokdiagram

Implementering

I det implementerede system bruger vi tre hovedblokke til at opbygge det fulde kredsløb: ADC'en (Analog-To-Digital Converter), Scaling Unit og Display Multiplexer med 7-segment dekoder. Da display multiplexer og 7-segment dekoder allerede er beskrevet i tidligere rapporter, vil vi fokusere på ADC'en og scaling unit'en.

FSM (Finite State Machine)

Interne tællere og signaler

Vores FSM benytter interne tællere og signaler til at styre konverteringsprocessen. Disse omfatter **state_reg** og **state_next**, der styrer tilstandsovergange, og en intern 8-bit tæller til at spore antallet af clock-cyklusser under konvertering.

Tilstandsovergange

Ved start af konverteringsprocessen aktiveres vores FSM, og den går i gang med at overvåge tilstandsovergange. Vi bruger en tilstandsregister (**state_reg**) til at opretholde den aktuelle tilstand og en næste tilstandssignal (**state_next**) til at bestemme den næste tilstand baseret på aktuelle betingelser og kontrolsignaler.

Overvågning af komparator

Under konverteringsprocessen overvåger vores FSM også komparatoren fra R2R-netværket. Når komparatoren registrerer, at det analoge input-signal har nået eller overskredet den korrekte værdi, aktiveres comp-signalet, hvilket indikerer, at konverteringen skal afsluttes. Dette udløser en tilstands-overgang i FSM'en, hvor antallet af trin sendes til vores step scaling blok til yderligere behandling.

Kontrol af konverterings-cyklus

Under hele konverterings-cyklussen overvåger FSM'en clock-signalet (**clk**) og styrer, hvornår konverteringen starter, stopper og rapporterer om status gennem relevante kontrolsignaler som **'start'** og **'comp'**.

ADC (Analog-To-Digital Converter) blok

Vores ADC Counter benytter sig af både interne tællinger, inputs '**clk**' '**reset**' '**enable**', samt nogle logiske inputs '**start**' og '**comp**'.

Når ADC'en er aktiveret og konverteringsprocessen er startet, begynder tælleren at tælle opad med hver cyklus af prescaleren. Tælleren benytter sig af '**state_reg**' og '**state_next**' til dette, dens tælling udføres internt i enheden og anvender en 8-bit tæller, hvilket betyder, at den kan repræsentere værdier fra 0 til 255, hvilket er passende til vores 8bit R2R ADC.

Under konverteringsprocessen overvåger tælleren også komparatoren fra R2R-netværket. Når R2R komparator signalet, '**comp**', går lav, indikerer det, at tællingen skal stoppes, og herfra vil vores antal af steps blive videresendt til vores step scaling blok.

For at forbedre brugeroplevelsen er der også tilføjet indikatorlys til vores ADC. Når ADC'en er i gang med at konvertere, lyser "busy"-lyset, og når den er klar til at starte en ny konvertering, lyser "ready"-lyset.

```
25      -- Sequential state assignment
26      process (clk, reset)
27      begin
28          if (reset = '1') then
29              state_reg <= S_IDLE;
30              counter <= (others => '0');
31          elsif rising_edge(clk) then
32              if (en = '1') then            -- state and so on only shifted when en is true, to slow down state machine.
33                  state_reg <= state_next;  -- data-path for state variable
34                  counter <= counter_next;  -- data-path for counter
35              end if;
36          end if;
37      end process;
38      -- Next state logic with Moore and Mealy output logic
39      process(state_reg, start, comp, counter)
40      begin
41          -- Default
42          state_next <= state_reg;
43          counter_next <= counter;
44          r2r <= (counter);
45          value <= (counter);
46          -- Finite State Machine with Data path
47          case state_reg is
48              when S_IDLE =>
49                  ready <= '1';                    -- Ready to start
50                  busy <= '0';
51                  if start = '1' then              -- Waiting for Start command
52                      state_next <= S_COUNT;       -- Next state
53                      counter_next <= (others => '0'); -- Reset counter
54                  end if;
55              when S_COUNT =>
56                  ready <= '0';
57                  busy <= '1';                     -- Busy
58                  if comp = '0' then               -- Check compare input
59                      state_next <= S_DONE;        -- Conversion done if comparator output is high
60                  else
61                      counter_next <= std_logic_vector(unsigned(counter) + 1);  -- Increment counter
62                  end if;
63              when S_DONE =>
64                  ready <= '1';                    -- Ready again
65                  busy <= '0';
66                  state_next <= S_IDLE;            -- Go to first state again
67          end case;
68      end process;
```

Figur 2 - FSM og ADC logik

Step scaling blok

Vi benytter os af en step scaling blok til at få et output vi kan vise på vores display, vi tager antallet af steps ganget med vores R2R resolution, 13 mV, og derefter deler dem op i 4 outputs ved at tage modulus, for at få cifrene der hver skal bruges på de enkelte displays.

```
15  architecture Behavioral of scaling_unit is
16    signal bin_int : integer := 0;
17  begin
18    process(scale_in)
19    begin
20      bin_int <= to_integer((unsigned(scale_in) * 13)); -- multiply by 13mV/bit
21
22      -- Extract BCD digits
23      bcd3 <= std_logic_vector(to_unsigned((bin_int / 1000) mod 10, 4));
24      bcd2 <= std_logic_vector(to_unsigned((bin_int / 100) mod 10, 4));
25      bcd1 <= std_logic_vector(to_unsigned((bin_int / 10) mod 10, 4));
26      bcd0 <= std_logic_vector(to_unsigned(bin_int mod 10, 4));
27    end process;
28  end Behavioral;
```

Figur 3 - Scaling_unit logik

Simulering

For at validere funktionaliteten af vores design blev der udført omfattende simuleringer ved hjælp af en testbench. Formålet med disse simuleringer var at verificere, at ADC'en fungerede som forventet i forskellige scenarier og ved forskellige inputsignaler.

Forventede resultater:

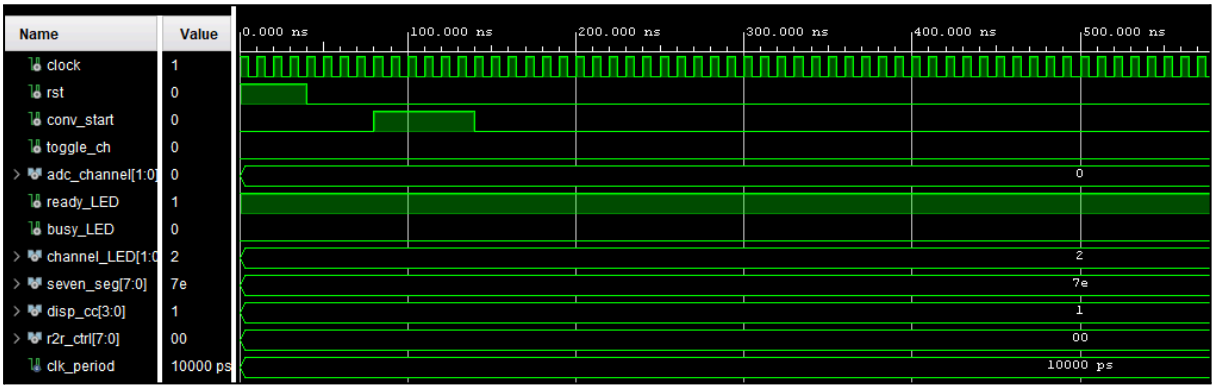
I simuleringen forventede vi, at når **'conv_Start'**-signalet blev aktiveret, ville ADC-enheden begynde en konverteringscyklus, hvilket ville resultere i aktivering af **'busy_LED'**-signalet. Derefter skulle ADC'en korrekt konvertere det analoge inputsignal og levere den tilsvarende digitale repræsentation på **'seven_seg'**-outputtet. Vi forventede også at observere, at når den aktuelle konvertering var afsluttet, ville **'ready_LED'**-signalet blive aktiveret for at indikere, at ADC'en var klar til en ny konvertering.

Aktuelle resultater:

Desværre opnåede vi ikke de forventede resultater i vores simuleringer. Vi observerede, at **'conv_start'** blev aktiveret korrekt en gang, men derefter blev værdien på **'seven_seg'**-displayet konstant og uændret, og viste blot 0. Dette var ikke den forventede adfærd, da vi forventede, at **'seven_seg'**-displayet skulle vise en korrekt digital repræsentation af det analoge inputsignal efter en vellykket konvertering i ADC'en.

```
51      -- Stimulus process
52      stim_proc: process
53      begin
54          -- Reset the system
55          rst <= '1';
56          wait for 40 ns;
57          rst <= '0';
58          wait for 40 ns;
59
60          -- Start a conversion
61          conv_start <= '1';
62          wait for 60 ns;
63          conv_start <= '0';
64
65          -- Simulate the comparator response with fixed value for testing
66          for i in 0 to 7 loop
67              wait until (r2r_ctrl /= "00000000" and busy_LED = '1');
68              wait for 100 ns;
69
70              -- Simulate comparator signal based on expected output value (e.g., 128)
71              if unsigned(r2r_ctrl) >= 128 then
72                  adc_channel <= "10";
73              else
74                  adc_channel <= "00";
75              end if;
76
77              wait for 100 ns;
78          end loop;
79
80          -- Wait for the conversion to complete
81          wait until ready_LED = '1';
82
83          -- Check the result in `seven_seg` output
84          wait for 50 ns;
85          assert seven_seg = "0111111";  -- Expected 7-segment code for '1' (LSB)
86
87          wait;
88      end process;
89  end Behavioral;
```

Figur 4 - Testbench process



Figur 5 - Simulering

Under simuleringen initialiseres en sekvens af handlinger, der simulerer ADC'ens drift. Først udløses en nulstilling for at sikre, at systemet starter i en kendt tilstand. Derefter aktiveres en konverteringscyklus ved at sætte '**conv_start**' signalet højt i en bestemt periode, hvilket simulerer starten af en analog-til-digital konvertering. Efter denne periode deaktiveres '**conv_start**'-signalet igen.

Herefter simuleres ADC'ens respons på det analoge inputsignal. En løkke genererer en række simuleringer af ADC'ens konvertering. I hver iteration af løkken simuleres ADC'ens adfærd, herunder vurdering af den analoge inputværdi, konvertering af denne værdi til en digital repræsentation og angivelse af, om ADC'en er klar til en ny konvertering.

Desværre lykkedes det ikke under simuleringen at opnå de forventede resultater. På trods af gentagne forsøg blev det observeret, at '**ready_LED**'-signalet forblev aktivt hele tiden, hvilket indikerede, at ADC'en ikke registrerede, at konverteringscyklussen var fuldført. Desuden blev det bemærket, at outputtet på '**seven_seg**' displayet forblev uændret og viste en konstant værdi, hvilket indikerer, at ADC'en muligvis ikke korrekt håndterede det analoge inputsignal.

Realisering

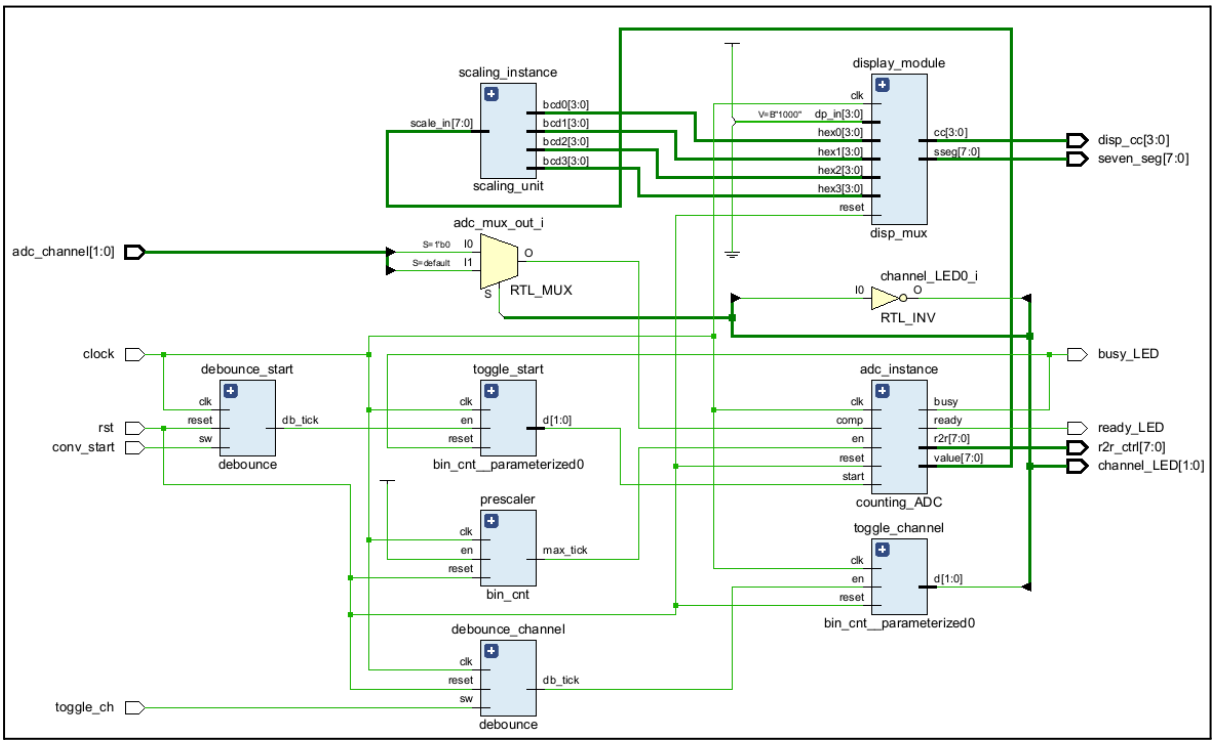
Systemet er opbygget omkring to centrale komponenter: ADC-enheden og scaling unit'en.

ADC-enheden fungerer som indgangsporten til digitalisering af det analoge inputsignal. Den styrer modtagelsen af input signaler som '**clk**' (ur), '**reset**' (nulstilling), '**start**' (start konvertering), '**comp**' (komparationsinput fra R2R) og '**en**' (aktivering af ADC'en). Derudover genererer den output signaler som '**ready**' (klar til ny konvertering) og '**busy**' (beskæftiget med konvertering). ADC'en spiller en afgørende rolle i at omsætte det kontinuerlige analoge input til en diskret digital repræsentation.

I vores system, hvor ADC'en opererer med et spændingsområde fra 0 til 3.3 volt og en opløsning på 8 bit (256 mulige trin), er det vigtigt at forstå, hvordan hver digital enhed, der læses fra ADC'en, korrelerer med ændringer i det analoge inputsignal. Med et spændingsområde fra 0 til 3.3 volt repræsenterer hver step i ADC'ens output en ændring på ca. $\frac{3.3\ volt}{256} \approx 0.01289\ volt/step$ hvilket svarer til cirka 13 millivolt (mV) pr. trin.

Dette betyder, at systemet er i stand til at måle ændringer i det analoge input signal med en opløsning på omkring 13 mV pr. trin, hvilket giver en måling inden for det specificerede spændingsområde. Denne evne til at konvertere spændingsændringer til digitale værdier med en sådan præcision er afgørende for mange applikationer, hvor nøjagtig og pålidelig dataopsamling er nødvendig.

Schematic



Figur 6 - Schematic

Demo

Til vores rapport inkluderer vi en demonstrationsvideo af vores projekt "Finite State Machine Analog-To-Digital Converter". I videoen demonstreres kredsløbets funktionalitet i en praktisk kontekst ved at visualisere, hvordan tælleren øges, og derefter stoppes, når '**comp**'-signalet går lavt.

Demonstration på artyboardet:
https://www.youtube.com/watch?v=uHrZtQXIYug&ab_channel=Danni

Constraints

I vores constraint-fil definerer vi de nødvendige hardwarekrav til FPGA'en, herunder pin-tildelinger og timing-specifikationer. Ved hjælp af '**set_property**'-kommandoer tildeler vi specifikke funktioner til FPGA-pins, såsom clock, reset og diverse input/output signaler.

For clock-signalet opretter vi en clock-kilde med en fastsat periode på 10 ns, navngivet '**sysClk**'. Dette sikrer en driftsfrekvens på 100 MHz, som er i overensstemmelse med vores designkrav.

Herefter specificerer vi individuelle pins for hver komponent, herunder 7-segment displays, displaydrivere, LEDs, R2R-ladningen, R2R-komparatoren og knapperne. Hver pin konfigureres med I/O-standarden LVCMOS33 for at sikre korrekt signalrouting og funktionalitet mellem FPGA'en og de tilsluttede enheder.

For eksempel tildeler vi pins til hvert segment af 7-segment displays samt displaydrivere, LED'er og knapper ved hjælp af '**set_property**'-kommandoer. Disse tildelinger sikrer korrekt funktionalitet og interaktion med det omgivende kredsløb.

Samlet set giver constraint-filen en klar definition af FPGA'ens hardwarekrav og sikrer, at designet opfylder de nødvendige hastigheds- og funktionalitets-specifikationer.

```
## clock
create_clock -period 10.000 -name sysClk -waveform {0.000 5.000} [get_ports {clk}]
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { clk }]; #100 MHZ clock

## buttons
set_property -dict { PACKAGE_PIN D9    IOSTANDARD LVCMOS33 } [get_ports { btn_r }]; #btn0 read
set_property -dict { PACKAGE_PIN C9    IOSTANDARD LVCMOS33 } [get_ports { btn_w }]; #btn1 write
set_property -dict { PACKAGE_PIN B8    IOSTANDARD LVCMOS33 } [get_ports { reset }]; #btn3 reset

## Switches
set_property -dict { PACKAGE_PIN A8    IOSTANDARD LVCMOS33 } [get_ports { val_in[0] }]; # Writedata
set_property -dict { PACKAGE_PIN C11   IOSTANDARD LVCMOS33 } [get_ports { val_in[1] }]; # Writedata
set_property -dict { PACKAGE_PIN C10   IOSTANDARD LVCMOS33 } [get_ports { val_in[2] }]; # Writedata
set_property -dict { PACKAGE_PIN A10   IOSTANDARD LVCMOS33 } [get_ports { val_in[3] }]; # Writedata

## LEDs
set_property -dict { PACKAGE_PIN F6    IOSTANDARD LVCMOS33 } [get_ports { buf_e }]; # fifo empty
set_property -dict { PACKAGE_PIN G6    IOSTANDARD LVCMOS33 } [get_ports { buf_f }]; # fifo full

## 7 segments drivers
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports { an[0] }]; # driver 0
set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { an[1] }]; # driver 1
set_property -dict { PACKAGE_PIN D15   IOSTANDARD LVCMOS33 } [get_ports { an[2] }]; # driver 2
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { an[3] }]; # driver 3

## 7 segment
set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports { sseg[0] }]; #Assign G
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports { sseg[1] }]; #Assign F
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { sseg[2] }]; #Assign E
set_property -dict { PACKAGE_PIN C15   IOSTANDARD LVCMOS33 } [get_ports { sseg[3] }]; #Assign D
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports { sseg[4] }]; #Assign C
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { sseg[5] }]; #Assign B
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports { sseg[6] }]; #Assign A
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { sseg[7] }]; #Assign dp
```

Figur 7 - Constraints fil

Konklusion

I vores laboratorieøvelse har vi udviklet og implementeret en analog-til-digital omformer FSM (Finite State Machine) med en ADC ved hjælp af VHDL, som kan konvertere et analogt input signal til en digital værdi, der vises på et 7-segment display. Ved at bruge Analog Discovery til at generere analoge spændinger har vi skabt en realistisk testplatform for vores ADC-system.

Vores design inkluderer centrale blokke som ADC-enheden, scaling unit'en og display multiplexer med 7-segment dekoder. Desværre opnåede vi ikke de forventede resultater under simuleringen, da displayet viste en konstant værdi i stedet for at afspejle det analoge input signal korrekt.

En central del af vores design var implementeringen af FSM, der styrer konverteringsprocessen og organiserer forskellige tilstande og overgange. FSM'en

fungerede som hjernen i vores ADC-controller og sikrede en struktureret og pålidelig konverteringsproces