INF3430 Høsten 2008

ChipScope PRO - En kort innføring

Innhold

Innledning

ChipScope PRO er et verktøy for å måle og monitorere interne signaler i en Xilinx FPGA for å kunne gjøre effektiv feilsøking (debugging) for å finne feil som simuleringer av en eller annen grunn ikke avdekker. Funksjonen er meget lik en tradisjonell logikkanalysator, der man kan studere mange signaler samtidig enten i form av timingdiagrammer eller lister (jfr. waveforms og lists i Modelsim). Hovedforskjellen er at en logikkanalysator benyttes for å måle og monitorere eksterne signaler. Man styrer innsamling av data ved å velge ut en samplingsklokke. En hovedutfordring man generelt har i elektronikk er å kunne effektivt fange øyeblikket der problemene viser seg. Å fange et slikt øyeblikk kalles trigging. Så det er viktig å kunne lage seg effektive triggebetingelser. Når triggebetingelsen(e) inntreffer blir data lagret og vi kan observere og feilsøke i timingdiagrammet som Chipscope lager. Triggebetingelsene kan variere fra enkle betingelser der man søker etter et bestemt mønster i dataene eller man lager mer avanserte betingelser f.eks. vi kan ønske at en bestemt mønster/begivenhet skal skje N antall ganger før datainnsamling starter eller man kan lage triggebetingelser som består av at en sekvens av begivenheter skal skje før data lagres. Vi kan også bestemme om og hvor mange sampel som skal lagres før triggebetingelsen(e) skal slå til. Dette ønsker man ofte fordi på den måten får man med seg historikken i signalene. Man kan tenke seg at man trigger på en kjent feil og at man kan da spore tilbake i tid fra tidspunktet feilen meldte seg og forhåpentligvis finne opphavet til feilen.

Før vi ser nærmere på bruk av ChipScope (Analyzer) må vi først klargjøre FPGA'en for bruk av ChipScope. Vi trenger å få laget to moduler som kalles ICON og ILA. Til å lage disse modulene benytter vi verktøyet ChipSCope Core Generator. Etter at modulene er generert kan vi instantiere dem i toppnivåarkitekturen vår eller vi kan benytte programmet ChipScope Core inserter. I dette eksemplet skal vi benytte instantiering i toppnivåarkitekturen.

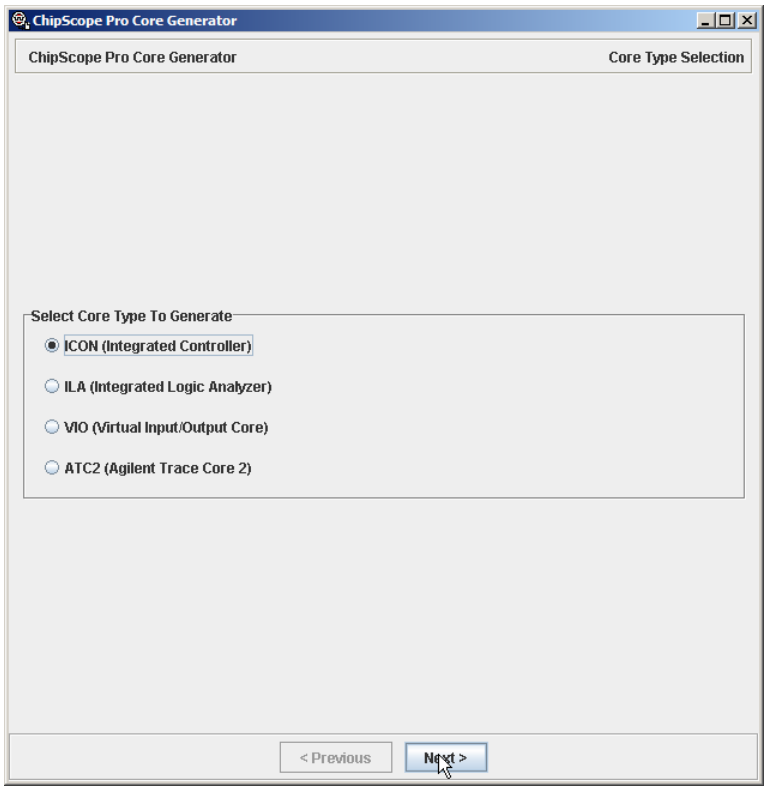
Verktøyflyten for vår bruk av Chipscope kan summeres opp slik:

1. Oppretting av et ISE prosjekt
2. Generering av ICON modul (Integrated Controller)
3. Generering av ILA modul (Integrated Logic Analyzer)
4. Instantiering av ICON og ILA entiteter i toppnivåarkitekturen.
5. Syntese, place and route, Bitstream generering
6. Oppstart av ChipScope analyzer og oppsett av et analyzer prosjekt.

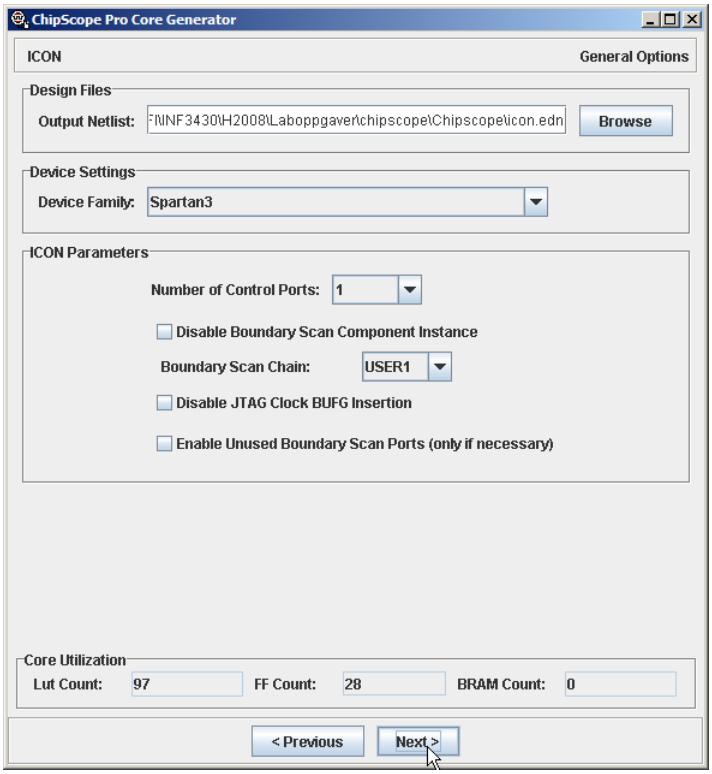
Generering av Chipscope kjerner

Generering av ICON (Integrated Controller) modul

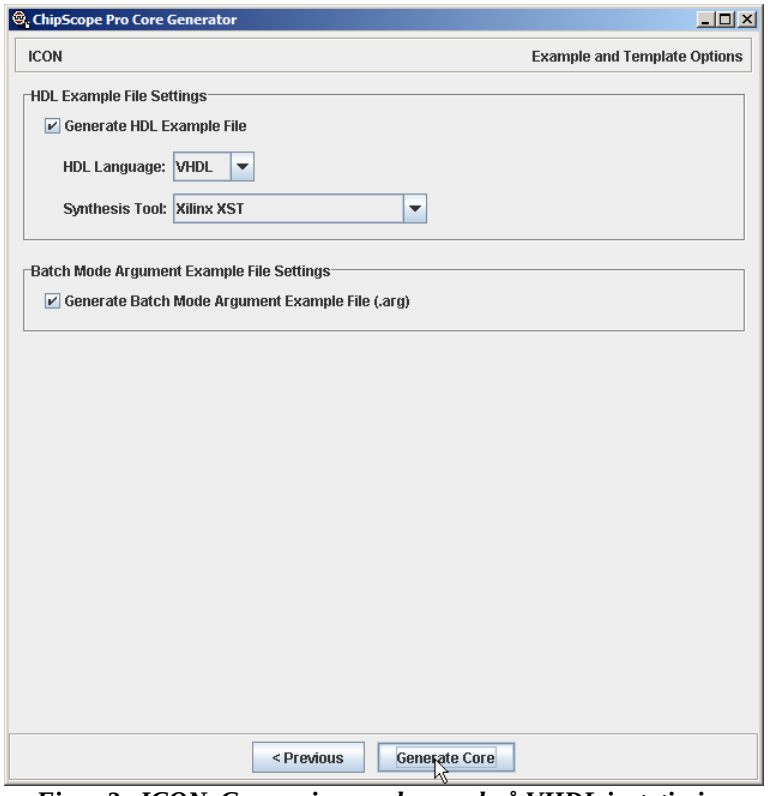
Vi starter Chipscope Core generator ved Start=>Programs=>ChipScope Pro 9.1i => Xilinx ChipScope Pro Core Generator og får fram følgende bilde. Man går igjennom "Wizard'en" for å generere ICON og ILA modulene som vist i de påfølgende figurene.



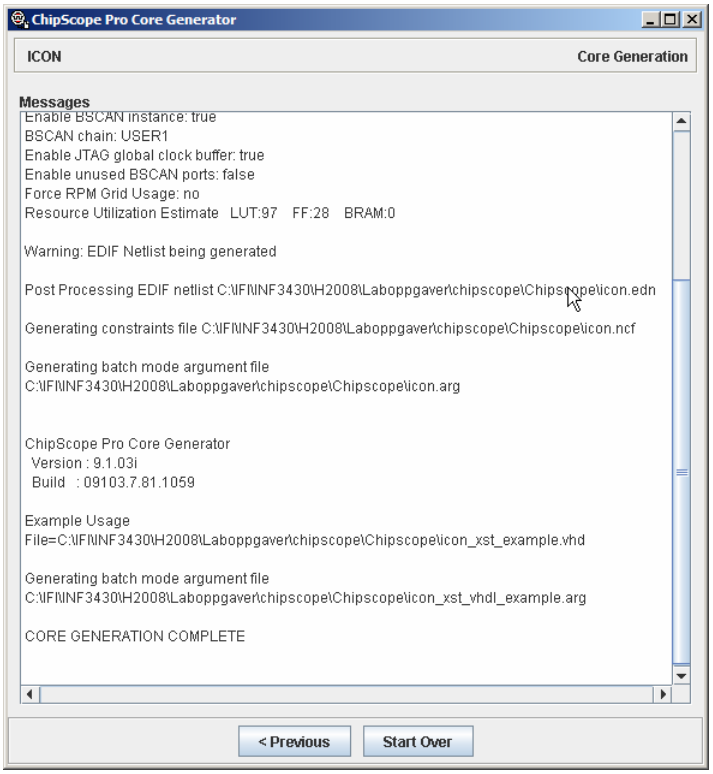
Figur 1. Hovedvindu i ChipScope Core generator. Valg av ICON kjerne



Figur 2. ICON. Valg av plassering og teknologi

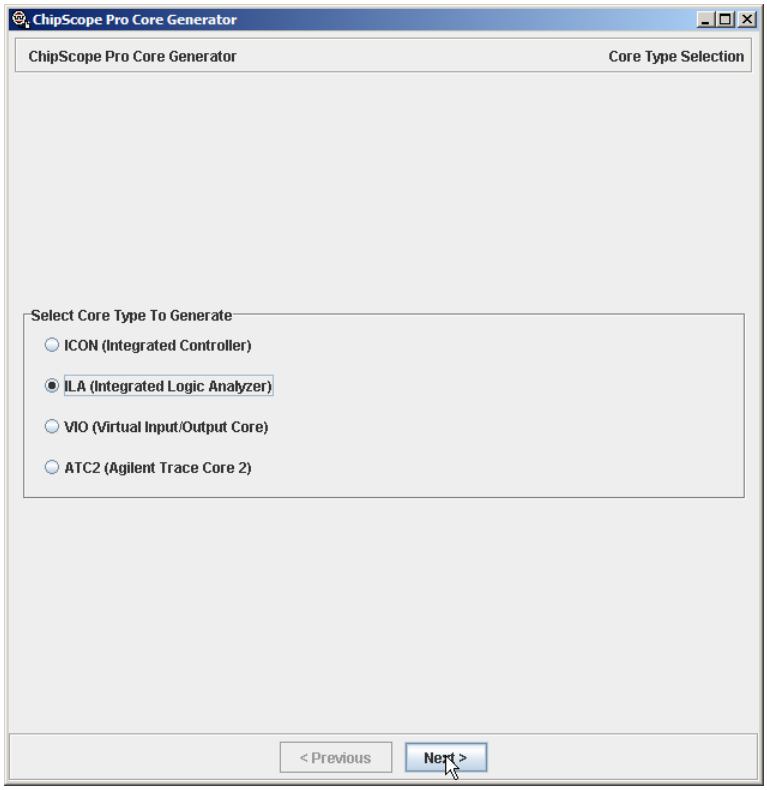


Figur 3. ICON. Generering av eksempel på VHDL instatiering

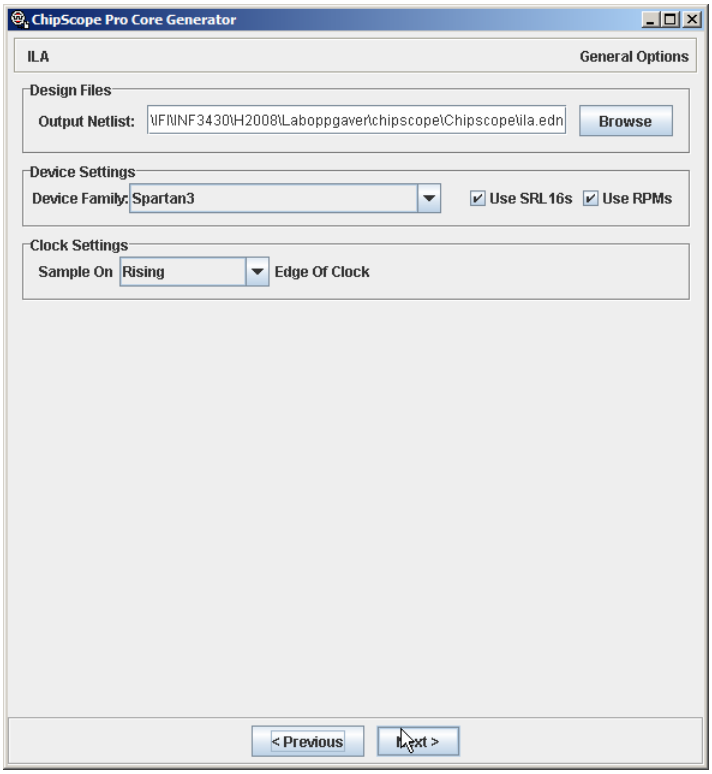


Figur 4. ICON. Oppsummeringsvindu. Sjekk at path'er stemmer

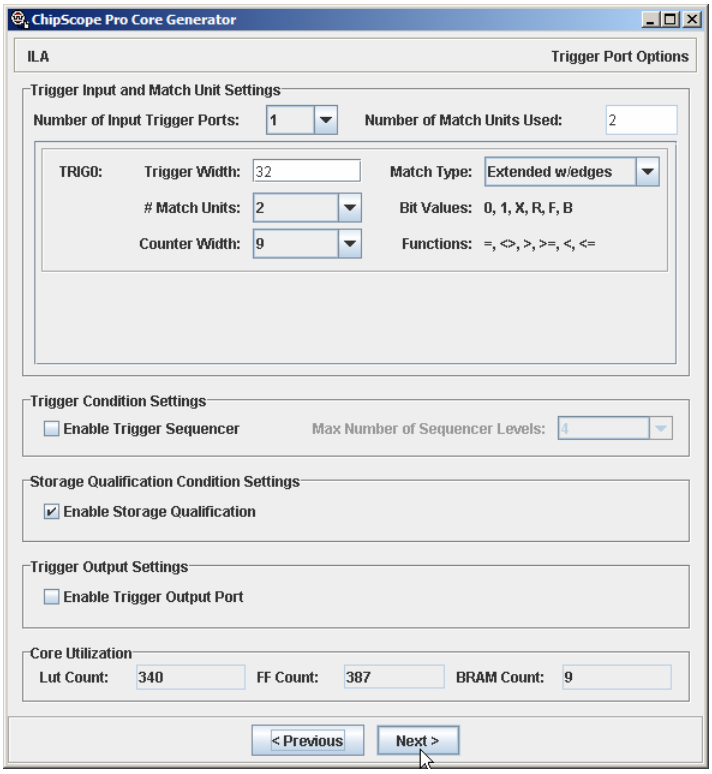
Generering av ILA (Integrated Logic Analyzer) modul



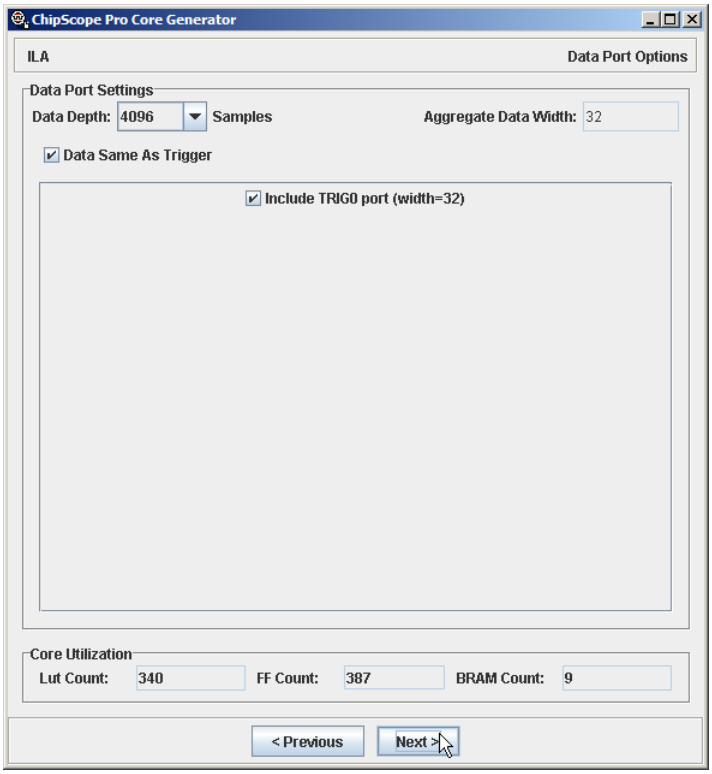
Figur 5. Valg av ILA (Integrated Logic Analyzer)



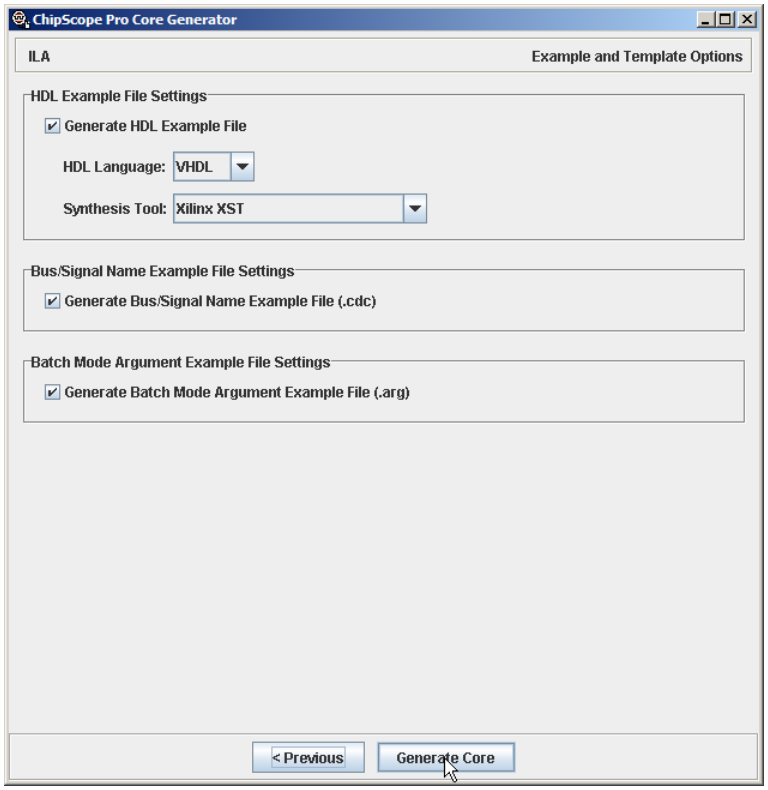
Figur 6. ILA. Valg av filplassering., teknologi og flanke på samplingsklokke



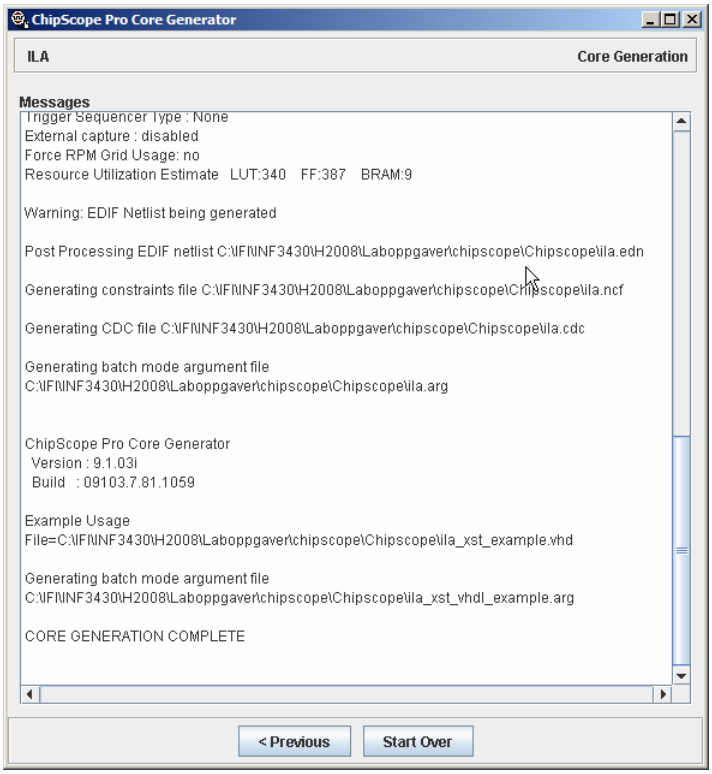
Figur 7. ILA. Valg av triggerport og triggeregenskaper



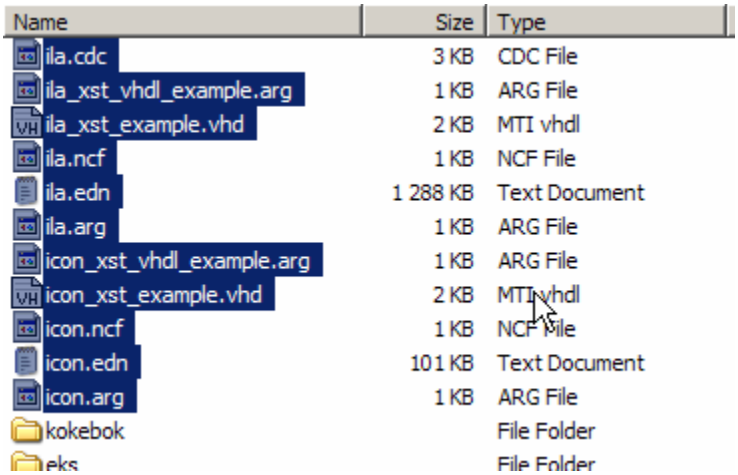
Figur 8. ILA. Valg av tracedybde og velger data samme som trigger



Figur 9. ILA. Generering av eksempel på instatiering



Figur 10. ILA. Oppsummeringsvindu



Figur 11. Genererte ICON og ILA filer

Eksempeldesign

Instantiering av ChipScope modulene ICON og ILA i toppnivåarkitekturen

Vi skal benytte ChipScope til å se på signalene i et enkelt eksempel. Eksempeldesignet består av en teller (first) som er koblet sammen med en sjusegmentdekoder funksjon. Telleren er fire bit og verdien av telleren vises på det ene sjusegmentdisplayet på testkortet.

Legg merke til at hver av entitetene first og bin2seg7 er utstyrt med et sett ekstra signaler, `chip_scope_out` kan kobles til ILA modulen. Hvilke signaler man tar ut på `chip_scope_out` avhenger av hva man er interessert i å se på og hva slags problem man står ovenfor. Ved å ha disse signalene tilgjengelige har man laget en struktur for testing og debugging. I vårt eksempel er det så små moduler at vi har tatt ut alt av signaler, mens i et virkelig design vil den normale situasjonen være at man må begrense seg. Man skal være klar over at bruk av Chipscope spiser opp betydelige mengder av ressursene man har i FPGA'en.

Instantieringen av ICON og ILA er et eksempel på en "Black box" instatiering der det ikke ligger noe VHDL kode under modulene, men foreligger som edn-filer (nettlister på edif-format.)[1] generert av Chipscope core generator og vil være en del av kildefilene i designet.

Scopetop.vhd

scopetop.vhd

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

entity scopetop is
  port
  (
    clk             : in  std_logic; -- Klokke fra bryter CLK1/INP1
    reset           : in  std_logic; -- Global Asynkron Reset
    load            : in  std_logic; -- Synkron reset
    inp             : in  std_logic_vector(3 downto 0); -- Startverdi
    max_count       : out std_logic;  -- Viser telleverdi
    seg7_en         : out std_logic_vector(3 downto 0);
    gfedcba         : out std_logic_vector(6 downto 0)
  );
end scopetop;

architecture rtl of scopetop is

  --  Område for deklarasjoner
  signal count      : std_logic_vector(3 downto 0);

  component first is
    port
    (
      clk             : in  std_logic; -- Klokke fra bryter CLK1/INP1
      reset           : in  std_logic; -- Global Asynkron Reset
```

[1] EDIF = Electronic Design Interchange Format

scopetop.vhd

```
      load            : in  std_logic; -- Synkron reset
      inp             : in  std_logic_vector(3 downto 0); -- Startverdi
      count           : out std_logic_vector(3 downto 0); -- Telleverdi
      max_count       : out std_logic;  -- Viser telleverdi
      chip_scope_out  : out std_logic_vector(7 downto 0) --Chipscope outputs
    );
  end component first;

  component bin2seg7 is
    port
    (
      bin             : in  std_logic_vector(3 downto 0);
      gfedcba         : out std_logic_vector(6 downto 0);
      chip_scope_out  : out std_logic_vector(6 downto 0)
    );
  end component bin2seg7;

  --Chipscope spesifikk seksjon
  signal control0   : std_logic_vector(35 downto 0);
  signal trig0      : std_logic_vector(31 downto 0);

  component icon
    port
    (
      control0    :   out std_logic_vector(35 downto 0)
    );
  end component;

  component ila
    port
    (
      control     : in    std_logic_vector(35 downto 0);
      clk         : in    std_logic;
      trig0       : in    std_logic_vector(31 downto 0)
    );
  end component;

begin

seg7_en <= "1110";

counter: first
port map
(
  clk            => clk,
  reset          => reset,
  load           => load,
  inp            => inp,
  count          => count,
  max_count      => max_count,
  chip_scope_out => trig0(13 downto 6)
);

seg7dekoder: bin2seg7
port map
(
  bin            => count,
  gfedcba        => gfedcba,
  chip_scope_out => trig0(20 downto 14)
);
```

scopetop.vhd

```
--Chipscope spesifikk seksjon
trig0(0)            <= reset;
trig0(1)            <= load;
trig0(5 downto 2)   <= inp;

i_icon : icon
port map
(
  control0    => control0
);

i_ila : ila
port map
(
  control   => control0,
  clk       => clk,
  trig0     => trig0
);


end rtl;
```

First.vhd

first.vhd

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

entity first is
  port
  (
    clk             : in  std_logic; -- Klokke fra bryter CLK1/INP1
    reset           : in  std_logic; -- Global Asynkron Reset
    load            : in  std_logic; -- Synkron reset
    inp             : in  std_logic_vector(3 downto 0); -- Startverdi
    count           : out std_logic_vector(3 downto 0); -- Telleverdi
    max_count       : out std_logic;  -- Viser telleverdi
    chip_scope_out  : out std_logic_vector(7 downto 0) --Chipscope outputs
  );
end first;

--  Arkitekturen under beskriver en 4-bits opp-teller.  Når telleren når
--  maksimal verdi går signalet MAX_COUNT aktivt.

architecture rtl of first is

  --  Område for deklarasjoner
  signal count_i      : unsigned(3 downto 0);
  signal max_count_i  : std_logic;

begin

  --  Her starter beskrivelsen
```

first.vhd

```
  counter :
  process (reset,clk)
  begin
    if(reset  = '1') then
      count_i <= (others => '0');
    elsif rising_edge(clk) then
      -- synkron reset
      if load = '1' then
        count_i <= unsigned(inp);
      else
        count_i <= count_i + 1;
      end if;
    end if;
  end process counter;

  count <= std_logic_vector(count_i);

  -- concurrent signal assignment
  max_count_i <= '1' when count_i = "1111" else '0';
  max_count <= max_count_i;

  --Signaler som kan vises i ChipScope hvis ønskelig
  chip_scope_out(3 downto 0)  <= std_logic_vector(count_i);
  chip_scope_out(4)           <= max_count_i;

end rtl;
```

Bin2seg7.vhd

bin2seg7.vhd

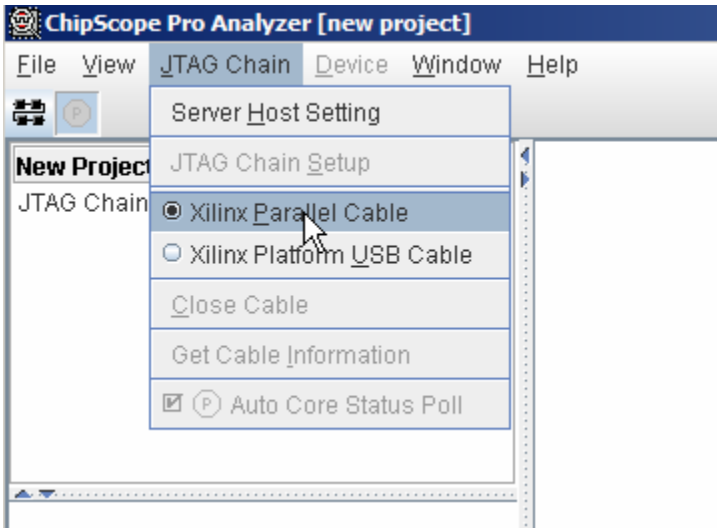
```
Library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.numeric_std.all;

entity bin2seg7 is
port
  (
    bin             : in  std_logic_vector(3 downto 0);
    gfedcba         : out std_logic_vector(6 downto 0);
    chip_scope_out  : out std_logic_vector(6 downto 0)
  );
end bin2seg7;

architecture rtl of bin2seg7 is

  signal gfedcba_i : std_logic_vector(6 downto 0);

begin

  bin2seg7process:
  process(bin)
  begin
    case bin is
        --  segments: gfedcba
      when  "0000" => gfedcba_i <=  "1000000";
```

bin2seg7.vhd

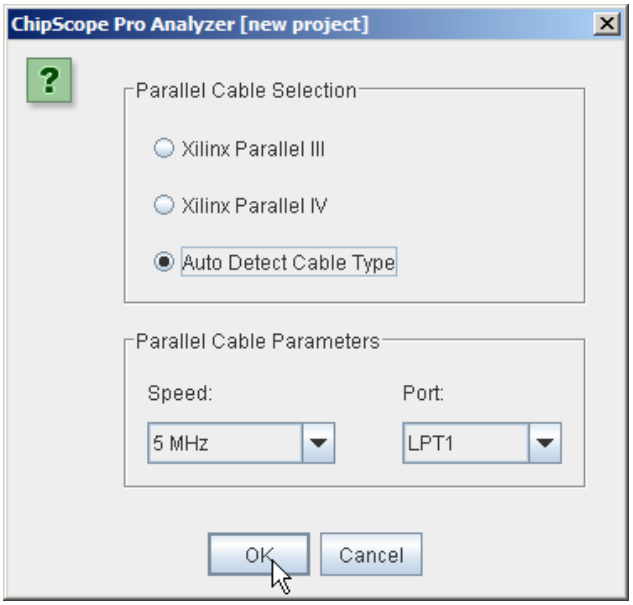
```
      when  "0001" => gfedcba_i <=  "1111001";
      when  "0010" => gfedcba_i <=  "0100100";
      when  "0011" => gfedcba_i <=  "0110000";
      when  "0100" => gfedcba_i <=  "0011001";
      when  "0101" => gfedcba_i <=  "0010010";
      when  "0110" => gfedcba_i <=  "0000010";
      when  "0111" => gfedcba_i <=  "1011000";
      when  "1000" => gfedcba_i <=  "0000000";
      when  "1001" => gfedcba_i <=  "0010000";
      when  "1010" => gfedcba_i <=  "0001000";
      when  "1011" => gfedcba_i <=  "0000011";
      when  "1100" => gfedcba_i <=  "1000110";
      when  "1101" => gfedcba_i <=  "0100001";
      when  "1110" => gfedcba_i <=  "0000110";
      when  others => gfedcba_i <=  "0001110";
    end case;
  end process;

  gfedcba         <= gfedcba_i;

  --Signaler som kan vises i ChipScope hvis ønskelig
  chip_scope_out  <= gfedcba_i;
end rtl;
```

ChipScope Pro Analyze

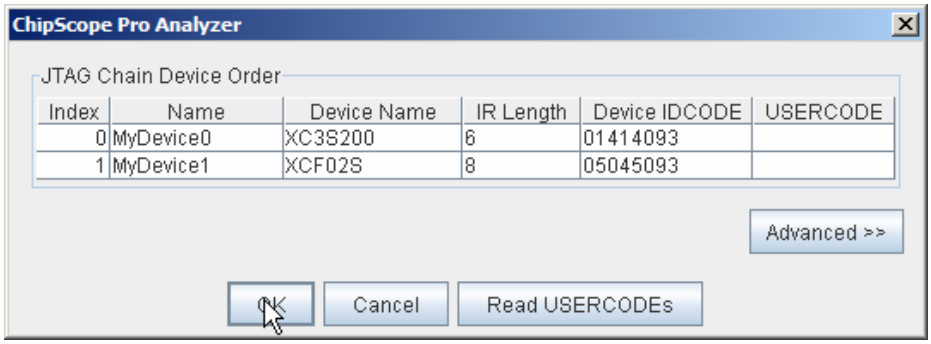
Vi kan starte ChipScope Pro analyze (fra nå av omtalt som ChipScope) enten innenfra ISE eller utenfra på samme måte som Core generator. Bruk av Chipscope bygger på at man har fått generert en bistreamfil som inneholder ICON og ILA modulene, og der ILA modulen er koblet til signaler fra designet vårt.
Når man har startet ChipScope kan man enten starte et nytt ChipScope prosjekt eller åpne et eksisterende. Alle innstillinger man har gjort blir lagret i prosjektet.
Det første man må gjøre er å konfigurere "download"-kabelen man benytter og detektere kretsene i JTAG-kjeden på kortet. Vi konfigurer denne ved å velge JTAG Chain i menyen. Vi velger Xilinx Parallel Cable og Autodetect. Dersom kabelen er koblet til og strøm er satt på kortet vil alle kretser som er koblet i JTAG-kjeden bli automatisk detektert.



Figur 12. Kabeloppsett. Valg av download kabel

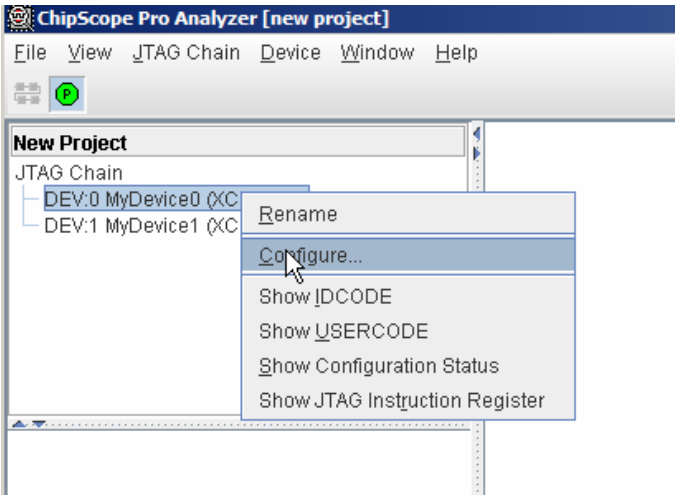


Figur 13. Kabeloppsett. Velg autodetect (eller Parallel III)

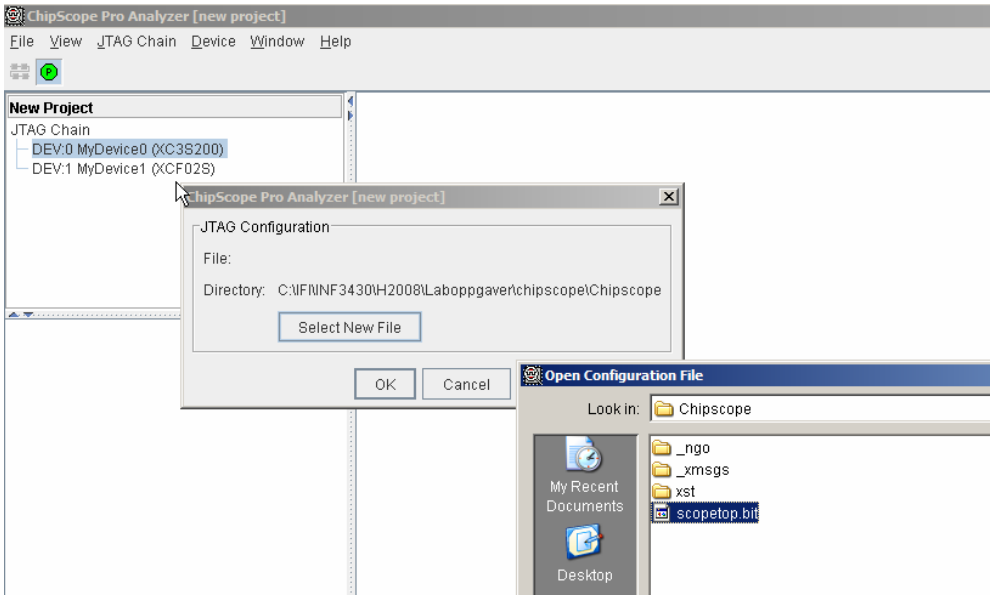


Figur 14. Kabeloppsett. Detekterte kretser i JTAG kjeden.

Etter FPGA'en vår er detektert skal vi konfigurere den ved å laste ned bitstreamfilen vi på forhånd har laget. Vi velger ut FPGA'en i JTAG chain (XC3S200), høyreklikk og velger Configure. Vi "browser" oss fram til ønsket bitstreamfil og velger Ok. Da vil FPGA'en bli konfigurert med innholdet i denne.

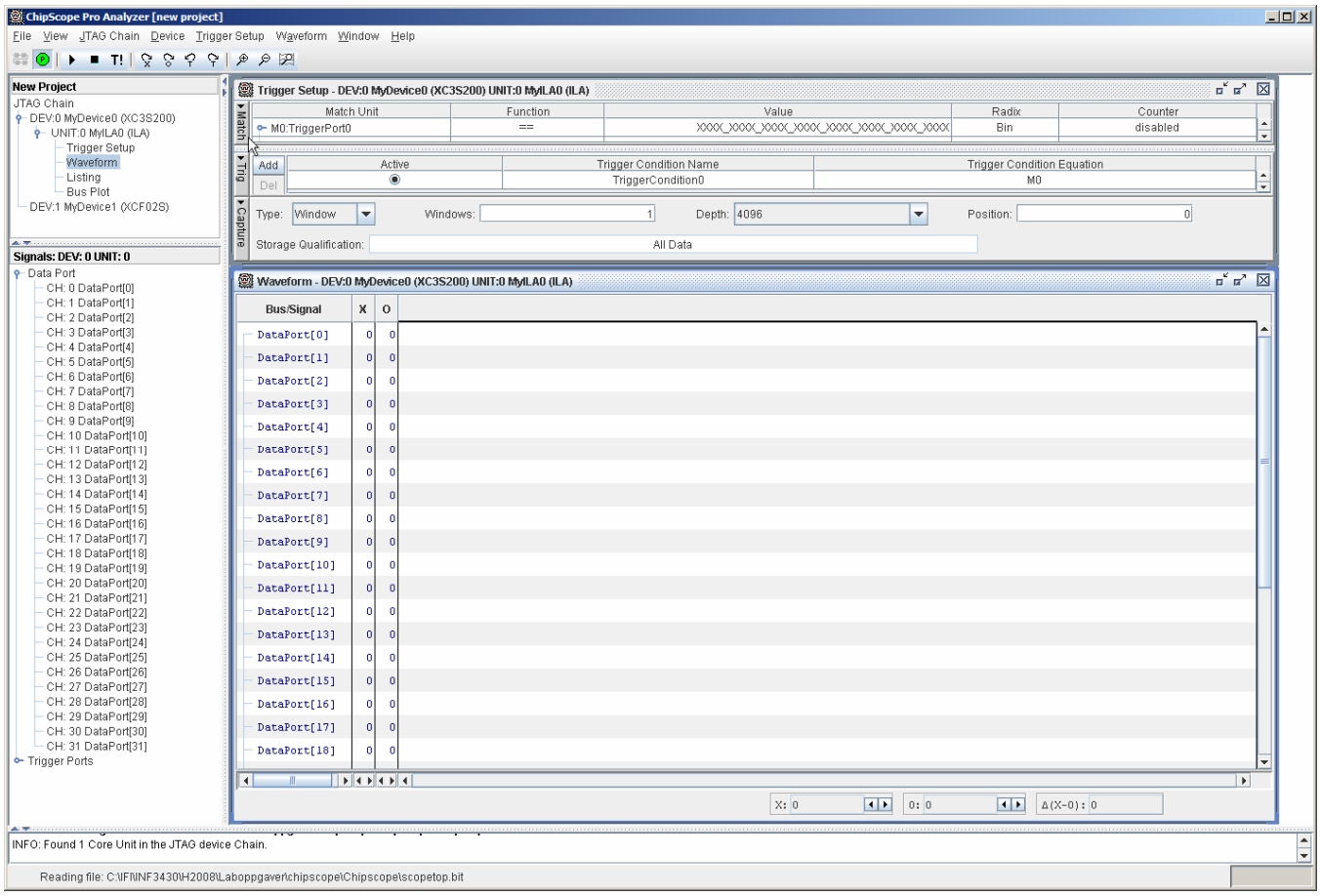


Figur 15. Konfigurere FPGA'en

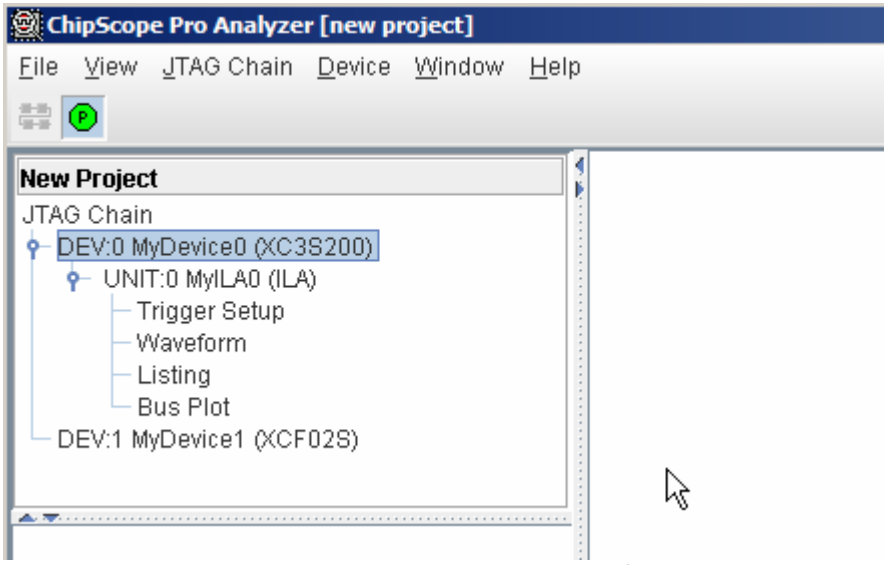


Figur 16. Valg av bitstreamfil

Etter at konfigureringen er ferdig og bitstreamfilen inneholder ICON og ILA modulene vil følgende bilde dukke opp.

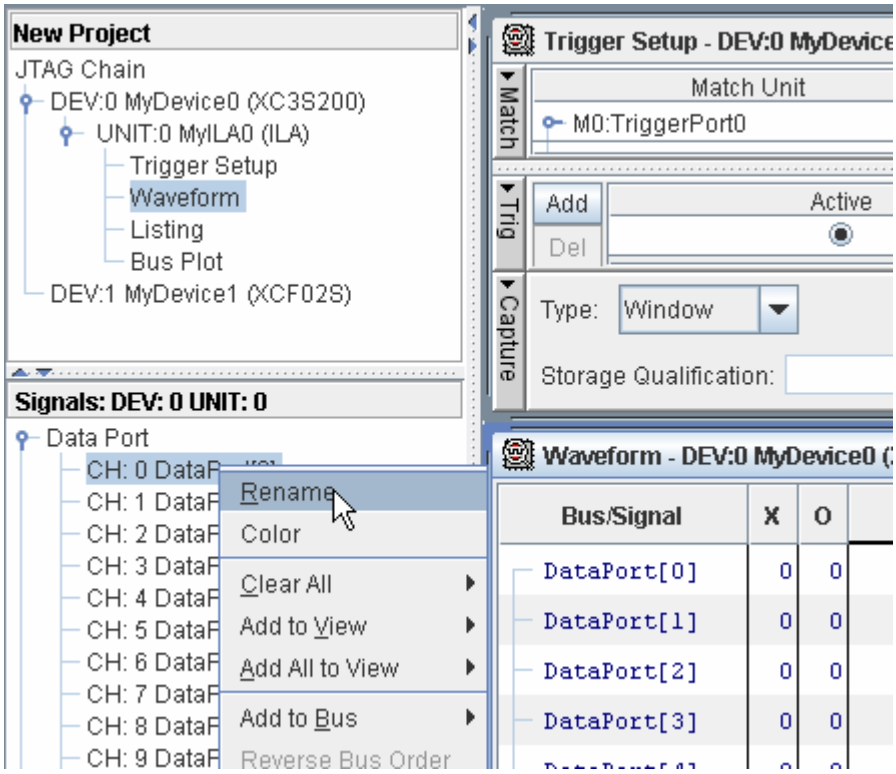


Figur 17. ChipScope etter at konfigurasjon er lastet opp

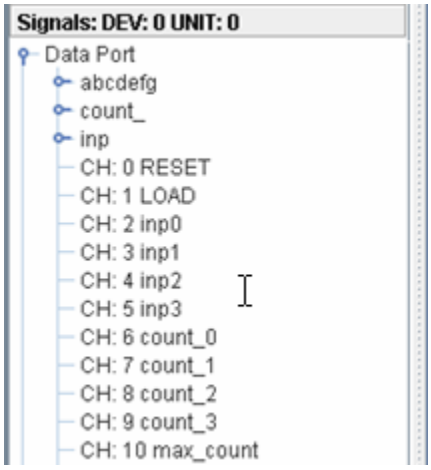


Figur 18. Detaljert utsnitt over ILA til vår krets

Signalene i ILA modulen vil på dette tidspunktet inneholde kryptiske navn. Så det første vi bør gjøre er å gi enkeltsignalene nytt navn i forhold til hvilke signaler de faktisk er koblet til inne i FPGA'en. Dette er en engangsjobb og dette oppsettet vil bli lagret i ChipScope prosjektfilen.

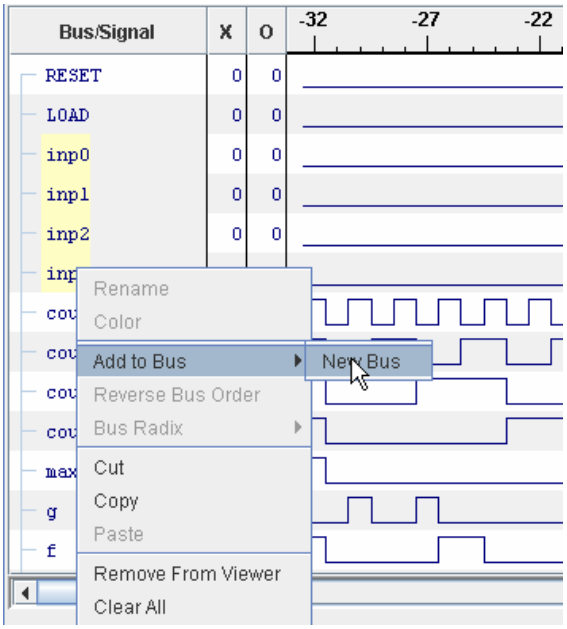


Figur 19. Navneendring til faktiske signalnavn



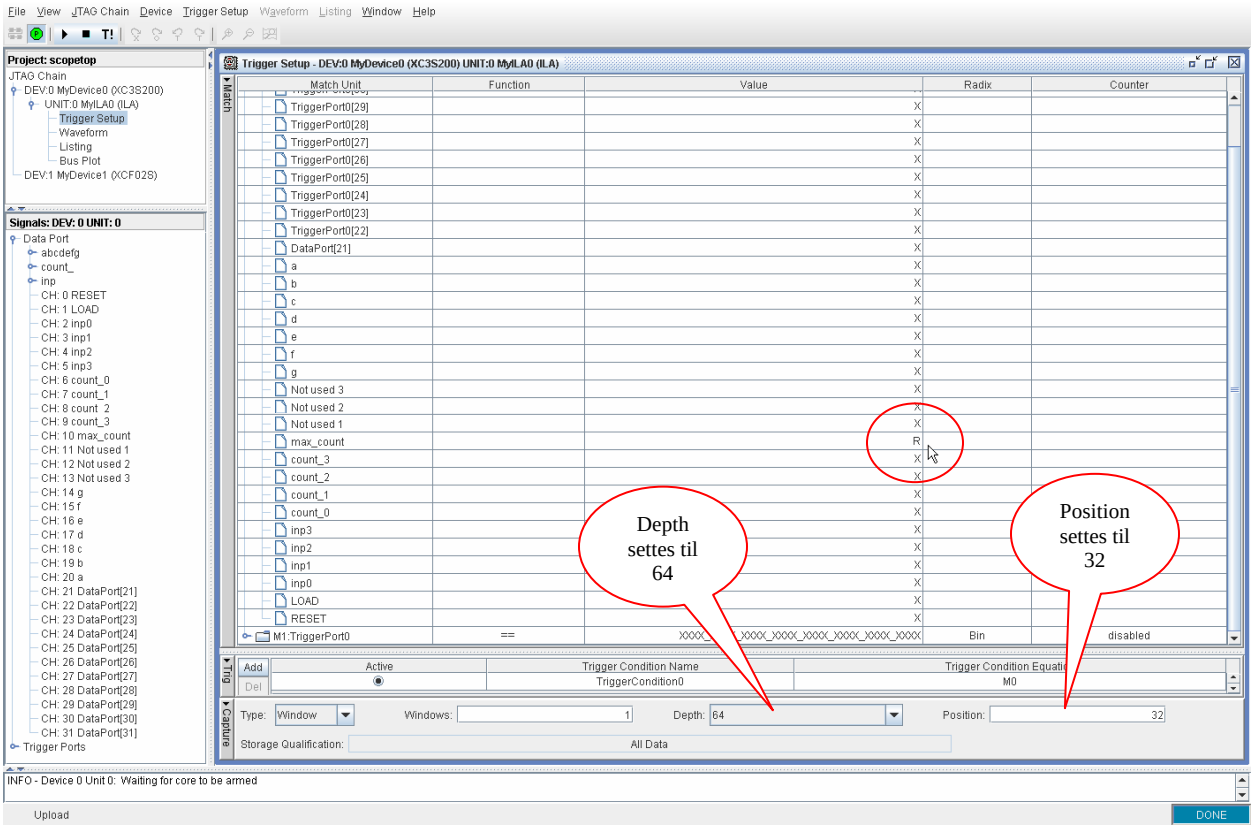
Figur 20. Etter navneendring

Det neste vi nå gjør er å eventuelt gruppere sammen signaler for å definere busser. F.eks. så passer det å gruppere sammen inp-signalene til bussen inp_(navn foreslått av verktøy) og count signalene til bussen count_. Det er lettere å studere hex-verdier enn enkeltbit.



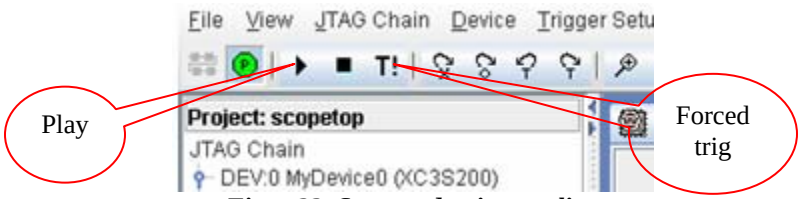
Figur 21. Definering av bus

Etter at vi har satt opp alle ønskede signalnavn og definert ønskede busser setter vi opp triggebetingelser. Da velger vi trigger setup i JTAG Chain vinduet og gjerne forstørrer trigger setup vinduet som i figuren under. I vårt eksempel velger vi å trigge på stigende flanke av signalet max_count. Videre ønsker vi å ikke benytte alle 4096 sampler vi har tilgjengelige i ILA, men avgrenser "tracen" vår til 64 sampel. Videre ønsker vi å lagre 32 sampel før (pre-store) triggebetingelsen inntreffer. Dette gjøres ved å sette "position" til 32.



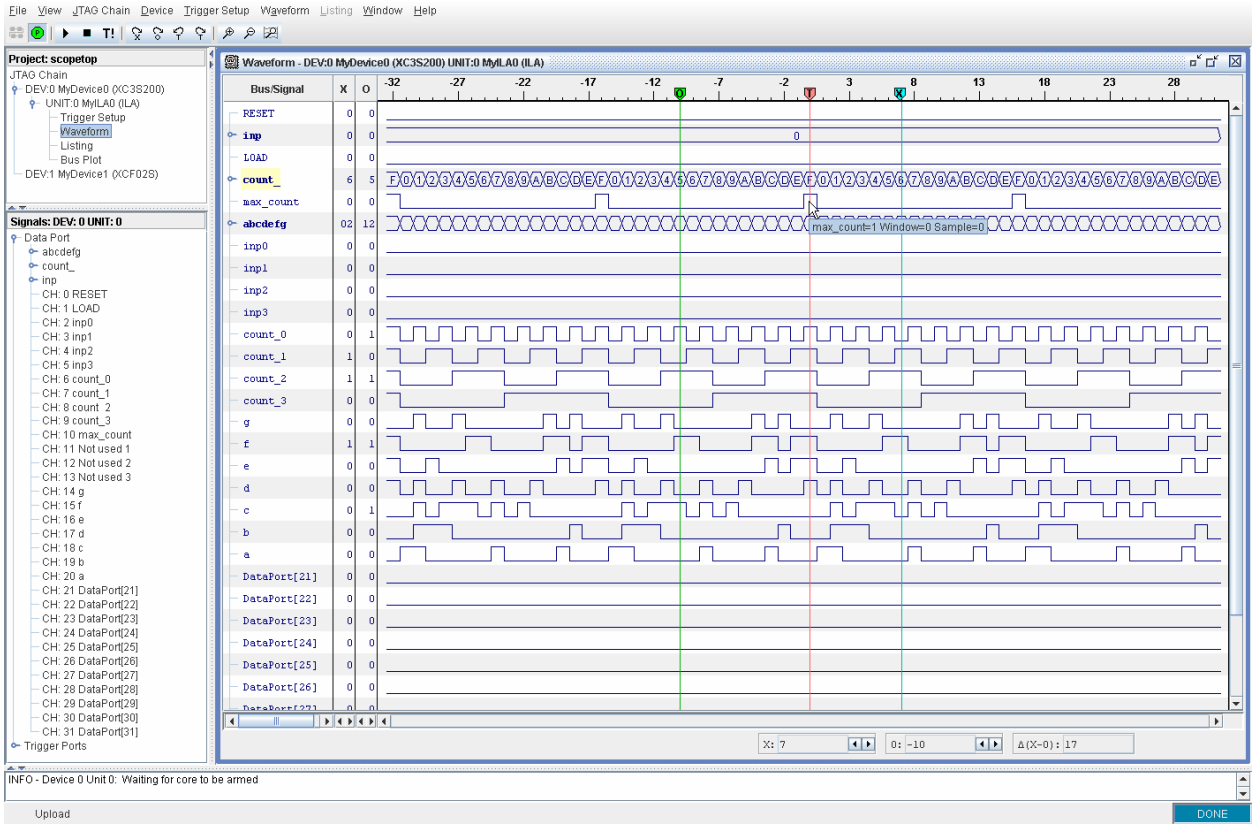
Figur 22. Trigger setup

Man starter datainnsamling ved å trykke på "Play" knappen som vist på neste figur:

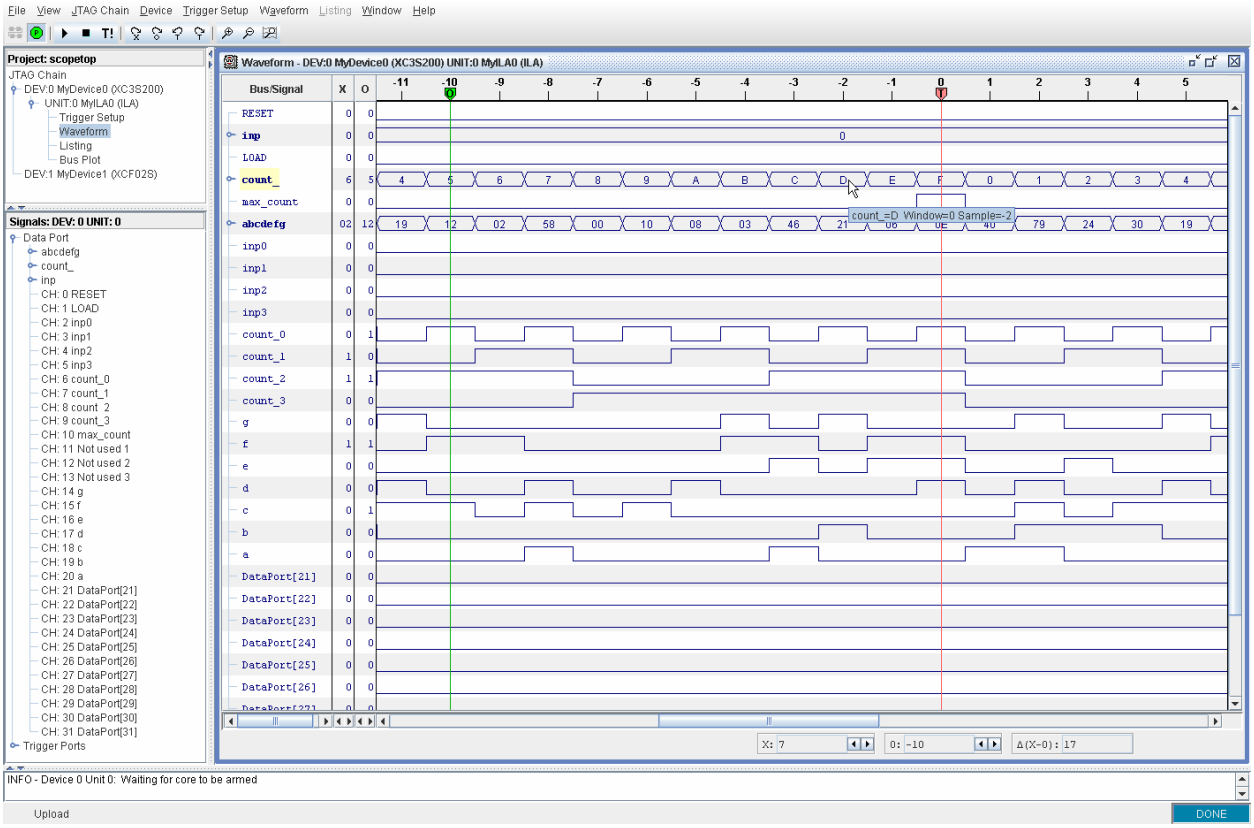


Figur 23. Start av datainnsamling

Når triggebetingelsen er sann vil datainnsamlingen stoppe og vi vil få vist et bilde tilsvarende figuren under. Legg merke til at vi har tre kursorer i bildet. En T-kursor som viser triggepunktet og en O- og X-kursor. Vi kan endre posisjonene på O- og X-kursorene og vi kan zoome inn området dekket av intervallet mellom O- og X-kursorene.
Man ønsker å avslutte med forced trig trykk "T!" når man tror at triggebetingelsen burde vårt oppfylt eller man bare vil sample for eksempel etter reset for å se på initialverdiene.

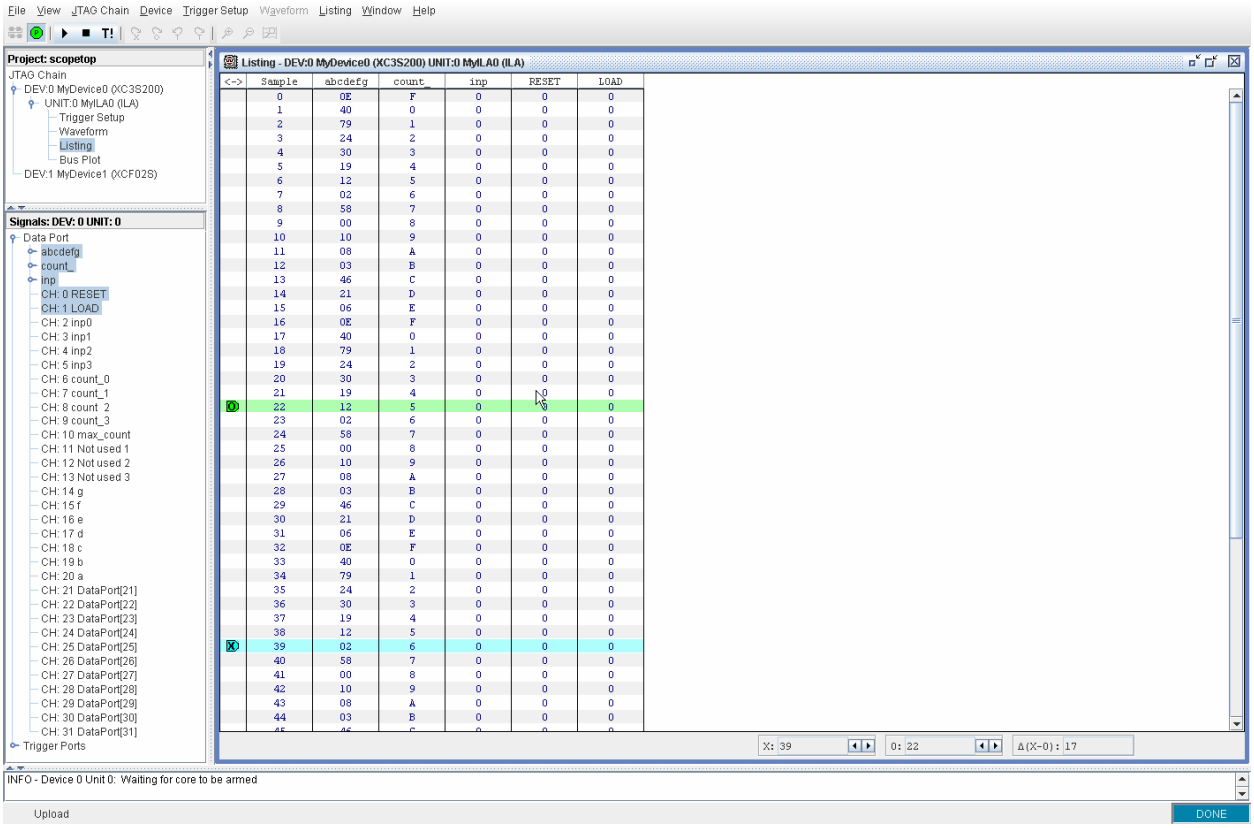


Figur 24. Waveformvindu med T-, O- og X-kursorer



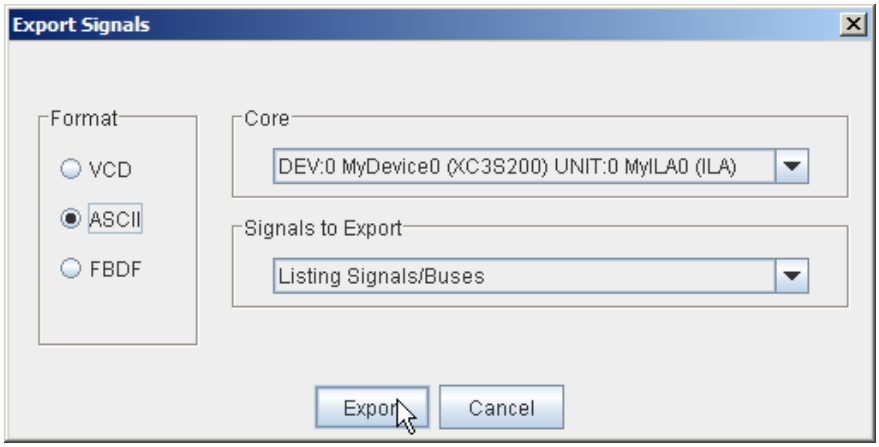
Figur 25. Waveformvindu zoomet til området dekket av O- og X-kursor

Man kan også velge å presentere dataene som en liste av sampler. Vi kan velge ut enkeltsignaler, høyreklikke og velge add to listing. Da får man et bilde tilsvarende figuren under.



Figur 26. Listing vindu

Vi kan eksportere inneholde av waveform og listing vinduene til forskjellige formater. F.eks. kan man ønske å benytte et trace til å analysere dataene i et annet program, som input Modelsim eller annet. Eller man kan ønske å benytte waveforms som dokumentasjon. Figuren under viser eksport av listing til en ascii-fil.



Figur 27. Eksport av listing til ASCII-fil

For ytterligere informasjon henvises til Xilinx Chipscope Pro Users manual som kan åpnes i programgruppen for ChipScope Pro eller fra Xilinx sin hjemmeside:
http://www.xilinx.com/support/documentation/sw_manuals/chipscope_pro_sw_cores_9_1i_ug029.pdf