INF 3430/4430

Simuleringsmetodikk

Innhold

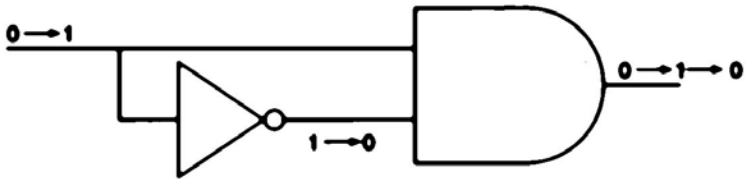
- Event driven simulation
- Simulering av VHDL-modeller
- Selvtestende testbenker
- Fil-operasjoner
- Eksempel på SRAM modell og simulering av lesing fra denne

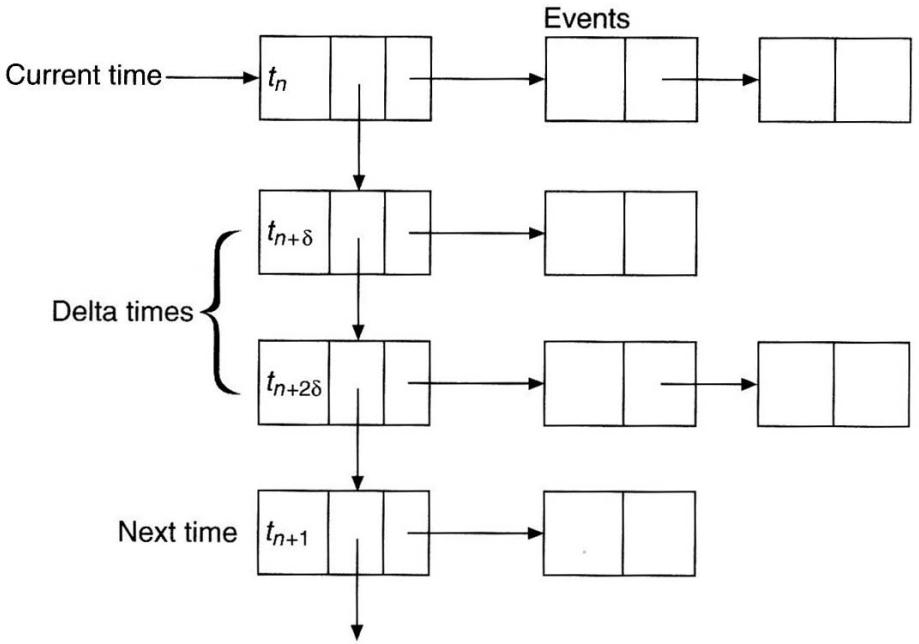
”Event driven” simulering

- Konseptet ”event driven” betyr at simuleringen drives framover i tid av endringer (event’er)
  - Endringer på innganger (stimuli)
  - Endringer på utganger som igjen trigger nye endringer
- Alle signaldrivere modelleres med en forsinkelse vi kaller ”delta-delay”.
- Et ”delta-delay” tilsvarer en forsinkelse på 0 fs (femto) og gir en mekanisme for et sette ”eventer” korrekt inn i en eventkø.

Eventkøer og delta-delay

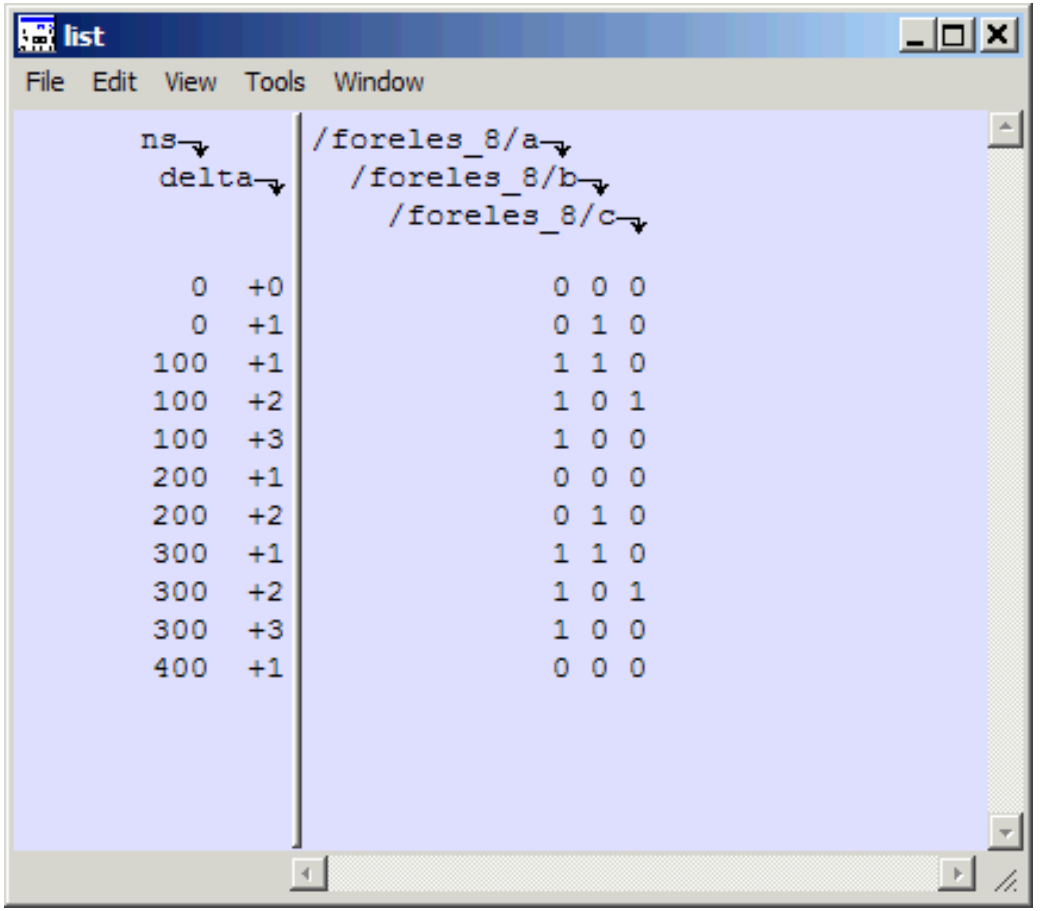
- I kretsen til høyre har alle elementer null forsinkelse.
- Null forsinkelse modelleres som delta-delay
- Det betyr at alle eventer generert ved ”current simulation time” blir ”schedulert” til å hende et delta-delay senere.
- Bruk av delta-delay sikrer at forskjellige simulatorer gir samme resultat.
- Delta-delay medfører stadige oppdatering og innsetting av nye eventer i eventkø.





Eventkøer og delta-delay

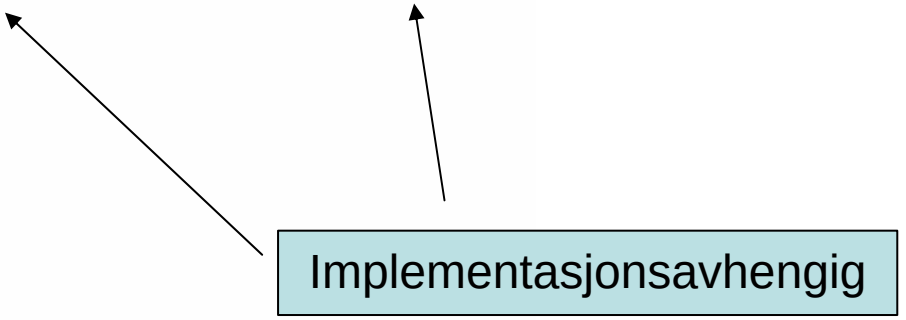
```
15:    STIMULI:
16:    process
17:    begin
18:      loop
19:        a <= '0';
20:        wait for 100 ns;
21:        a <= '1';
22:        wait for 100 ns;
23:      end loop;
24:    end process;
25:
26:    EKS1:
27:    process(a,b)
28:    begin
29:      b <= not a;
30:      c <= b and a;
31:    end process;
```



Simulering av VHDL-modeller

- Datatypen ***time*** er forhåndsdefinert i std.vhd
- Funksjonen ***now*** returnerer nåværende simuleringstid

```
type time is range -2147483647 to 2147483647
  units
    fs;
    ps = 1000 fs;
    ns = 1000 ps;
    us = 1000 ns;
    ms = 1000 us;
    sec = 1000 ms;
    min = 60 sec;
    hr = 60 min;
  end units;
```



Attributter (1)

These attributes are predefined for any signal X:

Name	Definition
X'event	True when X changes (boolean)
X'active	True when X assigned to (boolean)
X'last_event	When X last changed (time)
X'last_active	When X was last assigned to (time)
X'last_value	Previous value of X (same type as X)

Attributter (2)

These attributes create a **new signal**, based on signal X:

Name	**Definition**
X'delayed(T)	X, delayed by T (same type as X)
X'stable(T)	True if X unaltered for time T (boolean)
X'quiet(T)	True if X unassigned for time T (boolean)
X'transaction	"Toggles" when X is assigned (bit)

VHDL simuleringssyklus

- VHDL simulering består av to faser:
  - Initialiseringsfase
  - Repeterte eksekveringer av simuleringssyklusen
- Simulering starter ved 0 ns
- Nåværende simuleringstid, Tc
- Neste simuleringstid, Tn
- Tn beregnes ut fra den tidligste forekomst av:
  - TIME'high (max. Simuleringstid)
  - Neste gang en driver blir aktivert
  - Neste gang en process starter igjen (fortsetter)

VHDL simuleringssyklus

- Ved initialisering antar man at alle signaler har hatt sin verdi uendelig lenge
- Hver simuleringssyklus eksekveres ved at
  - Tc settes lik Tn
  - Alle eksplisitte tilordninger (input stimuli) og alle implisitte signaler oppdateres. Begge disse kan forårsake nye eventer, enten et delta delay framover eller til en annen tid. F.eks. A<='1' after 1 ns;
  - Hvis Tn = Tc er neste cycel en deltacycel

VHDL modellering

- Husk å inkludere alle relevante signaler på sensitivitetslisten
- For å gjøre simulering raskere:
  - Bruk et lite antall prosesser
  - Dette kan være i motsetning til godt lesbar kode
- Bruk av variabler simulerer raskere fordi disse er historieløse og enkle datastrukturer
- Bruk ’event istedenfor ’stable. ’stable lager et signal.
- Datatyper som *integer* simulerer raskere en f.eks. *std_logic* (mapper direkte til standard datatyper i språket simulatoren er kodet i).

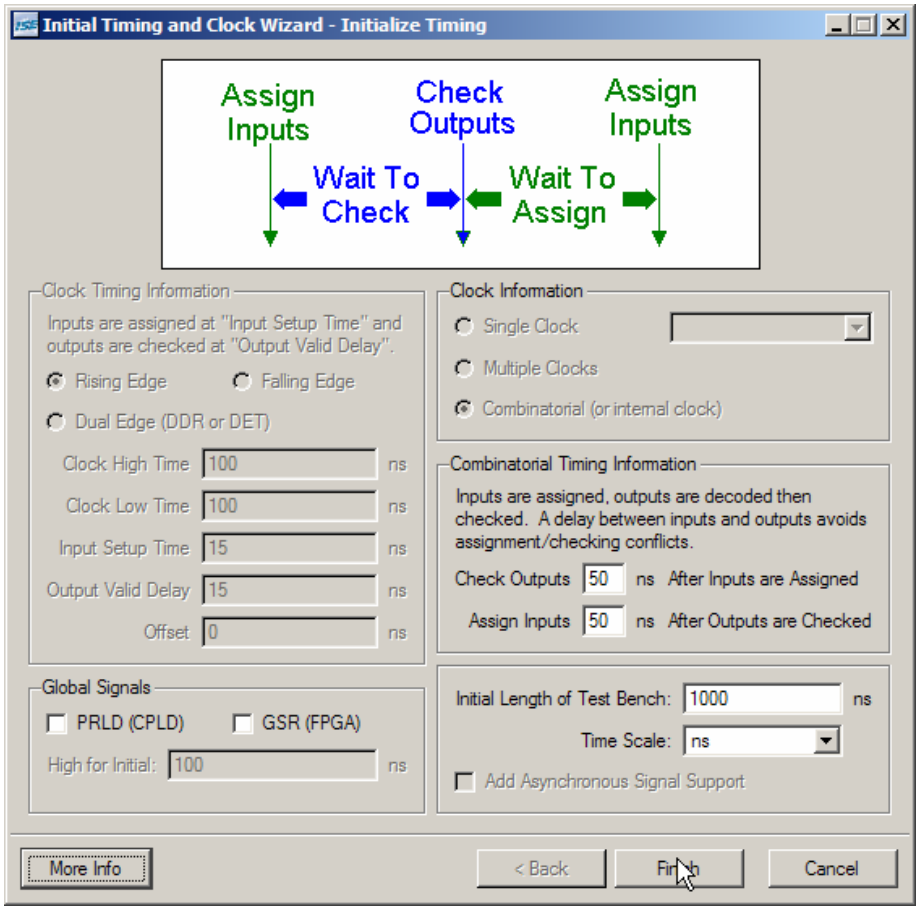
Selvtestende testbenker

- I en selvtestende testbenk sjekkes alle utganger opp mot en fasit, og resultat av simulering rapporteres som ”Ok” eller ”Not Ok”.
- Fordelen med dette er at man sparer mye tid til gransking av timingdiagrammer.
- Andre kan lettere vedlikeholde kode fordi man har en entydig virkemåte i fasiten.
- Krevende oppgave å lage selvtestende testbenker.

Selvtestende testbenker

- Xilinx ISE har verktøyet ”HDL bencher ” for å lage selvtestende testbenker. Grafisk input/redigering, VHDL output.
- Man tegner alle inngangsstimuli grafisk.
- Simulerer på behavorial nivå (kan være en ikke-syntetiserbar modell) i Modelsim til man er fornøyd med resultater
- Genererer deretter en selvtestende testbenk vhdl(verilog)-fil
- Man får generert en testbenk der alle utgangsverdier blir sjekket opp mot forventede verdier.
- Feil rapporteres
- Man starter HDL Bencher i processvinduet ved Create New source og så velge Test Bench Waveform

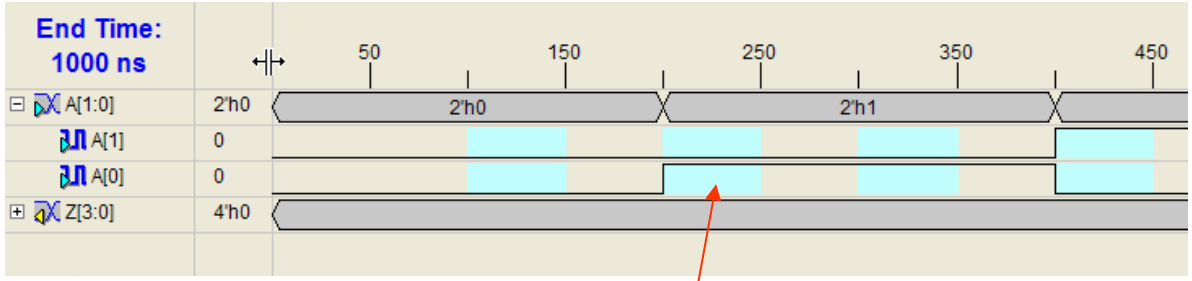
HDL Bencher (1)



HDL Bencher (2)

- Testfil: Decoder.vhd

```
23:   PROC_DECODER:
24:   process(A)
25:   begin
26:     if A = "00" then
27:       Z <= "0001";
28:     elsif A = "01" then
29:       Z <= "0010";
30:     elsif A = "10" then
31:       Z <= "0100";
32:     else
33:       Z <= "1000";
34:     end if;
35:   end process PROC_DECODER;
```



Redigering av input

Generert VHDL testbench

- Instantiering av UUT
- Generering av procedure som sjekker output
- Stimuli process med sjekking av outputs
- Den selvtestende testbenken ”husker” virkemåten og det er lett å repetere simuleringen senere på RTL-koden både for en selv og andre.
  - Meget viktig etter vedlikehold av design å sørge for at endringer i koden ikke får uønskede sideeffekter => regresjonstesting

Prosedyre for sjekking av utganger

```
PROCEDURE CHECK_Z(
    next_Z : std_logic_vector (3 DownTo 0);
    TX_TIME : INTEGER
) IS
    VARIABLE TX_STR : String(1 to 4096);
    VARIABLE TX_LOC : LINE;
    BEGIN
    IF (Z /= next_Z) THEN
        STD.TEXTIO.write(TX_LOC, string'("Error at time="));
        STD.TEXTIO.write(TX_LOC, TX_TIME);
        STD.TEXTIO.write(TX_LOC, string'("ns Z="));
        IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, Z);
        STD.TEXTIO.write(TX_LOC, string'(", Expected = "));
        IEEE.STD_LOGIC_TEXTIO.write(TX_LOC, next_Z);
        STD.TEXTIO.write(TX_LOC, string'(" "));
        TX_STR(TX_LOC.all'range) := TX_LOC.all;
        STD.TEXTIO.writeline(RESULTS, TX_LOC);
        STD.TEXTIO.Deallocate(TX_LOC);
        ASSERT (FALSE) REPORT TX_STR SEVERITY ERROR;
        TX_ERROR := TX_ERROR + 1;
    END IF;
END;
```

Stimuli og sjekk av utganger

```
BEGIN
    -- -------------  Current Time:  100ns
    WAIT FOR 100 ns;
    A <= "01";
    -- -------------------------------------
    -- -------------  Current Time:  150ns
    WAIT FOR 50 ns;
    CHECK_Z("0010", 150);
    -- -------------------------------------
    -- -------------  Current Time:  200ns
    WAIT FOR 50 ns;
    A <= "10";
    -- -------------------------------------
    -- -------------  Current Time:  250ns
    WAIT FOR 50 ns;
    CHECK_Z("0100", 250);
    -- -------------------------------------
    -- -------------  Current Time:  300ns
    WAIT FOR 50 ns;
    A <= "11";
    -- -------------------------------------
    -- -------------  Current Time:  350ns
    WAIT FOR 50 ns;
    CHECK_Z("1000", 350);
    -- -------------------------------------
```

Sluttrapport

```
IF (TX_ERROR = 0) THEN
    STD.TEXTIO.write(TX_OUT, string'("No errors or warnings"));
    STD.TEXTIO.writeline(RESULTS, TX_OUT);
    ASSERT (FALSE) REPORT
      "Simulation successful (not a failure).  No problems detected."
      SEVERITY FAILURE;
ELSE
    STD.TEXTIO.write(TX_OUT, TX_ERROR);
    STD.TEXTIO.write(TX_OUT,
        string'(" errors found in simulation"));
    STD.TEXTIO.writeline(RESULTS, TX_OUT);
    ASSERT (FALSE) REPORT "Errors found during simulation"
         SEVERITY FAILURE;
END IF;
```

Filoperasjoner

- Tekstfiler benyttes ofte til å lagre input stimuli og lagre resultater.
- Resultat-fil kan benyttes for å verifisere f.eks. syntesemodeller opp mot RTL-kode.
- Resultat-fil kan også benyttes som input stimuli til etterfølgende krets.

Eksempel: Testing av N-bit adder

```
36: begin
37: a1: entity work.NBitAdder port map (X, Y, ci, Z, co);
38:
39: p1: process is
40:       file vectors : text open read_mode is "vectors.txt";
41:       file results : text open write_mode is "result_alt.txt";
42:       variable ILine, OLine : Line;
43:       variable X_in, Y_in, Z_out : bit_vector((N-1) downto 0);
44:       variable ci_in, co_out : bit;
45:       variable ch : character;
46:     begin
47:       while not endfile(vectors) loop
48:         readline(vectors, ILine);
49:         read(ILine, X_in);
50:         read(ILine, ch);
51:         read(ILine, Y_in);
52:         read(ILine, ch);
53:         read(ILine, ci_in);
```

Testing av N-bit adder (forts)

```
54:
55:           X <= to_stdlogicvector(X_in);
56:           Y <= to_stdlogicvector(Y_in);
57:           ci <= to_stdulogic(ci_in);
58:           wait for 60 ns;
59:
60:           Z_out := to_bitvector(Z);
61:           co_out := to_bit(co);
62:           write(OLine, Z_out, right, 5);
63:           write(OLine, co_out, right, 2);
64:           writeline(results, OLine);
65:         end loop;
66:       end process p1;
67: end architecture fileio;
```

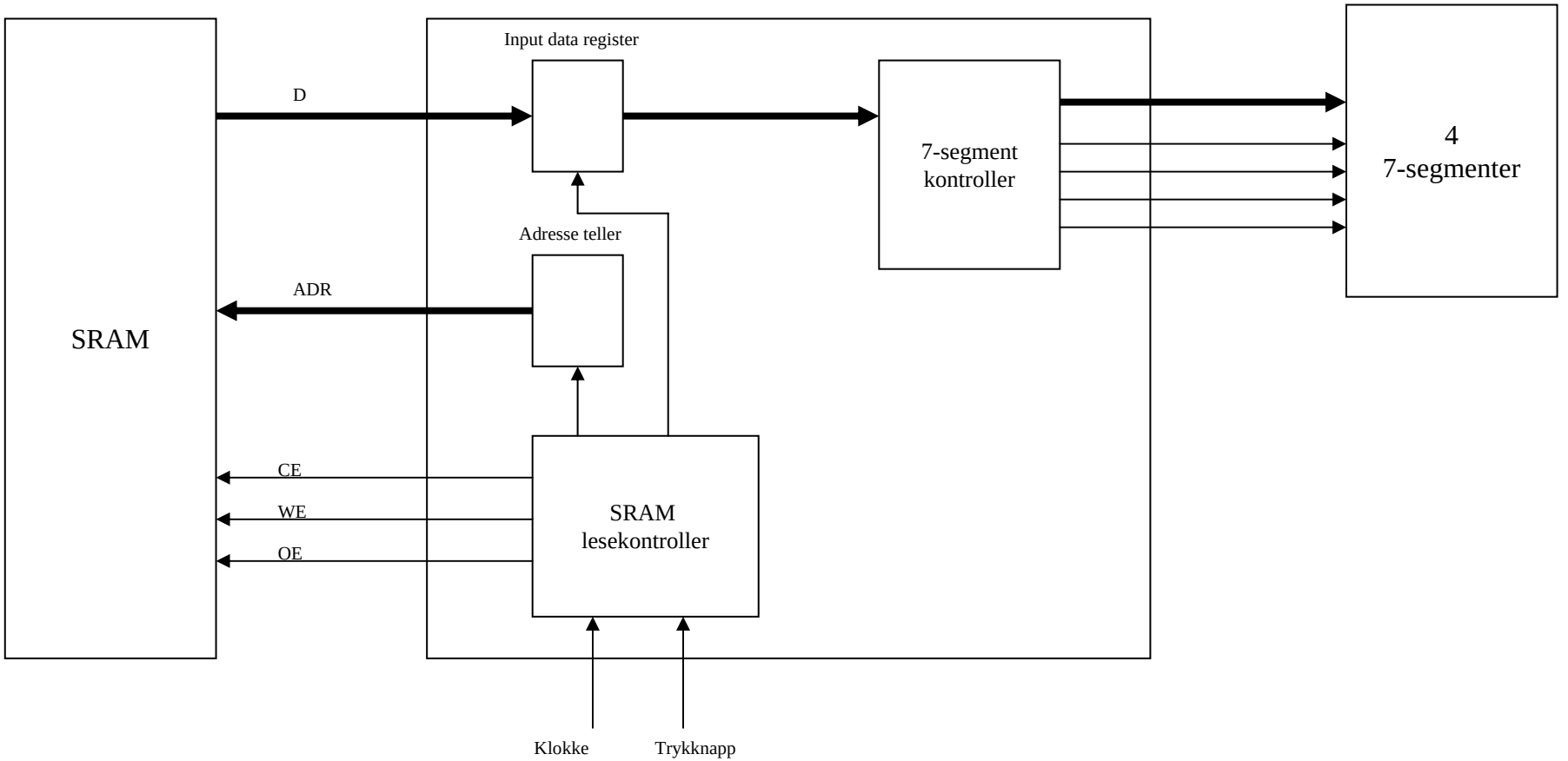
Utskrift av data til consol (transcript) vindu

```
48:       --Procedure for echoing the data to the consol window
49:       --Writing to std_out (output) must be performed inside
50:       --a procedure
51:       procedure MONITOR ( Z_out  : bit_vector;
52:                           co_out : bit) is
53:
54:       variable outline : line;
55:       begin
56:         write(outline, string'("Z_out: "));
57:         write(outline, Z_out, right, 5);
58:         write(outline, string'("\"));
59:         HWRITE(outline, to_stdlogicvector(Z_out));
60:         write(outline, string'("h"));
61:         write(outline, string'("  co_out: "));
62:         write(outline, co_out, right, 2);
63:         write(outline,string'("  Current time: "));
64:         write(outline,now,LEFT,12,ns);
65:         writeline(output,outline);
66:       end MONITOR;
```

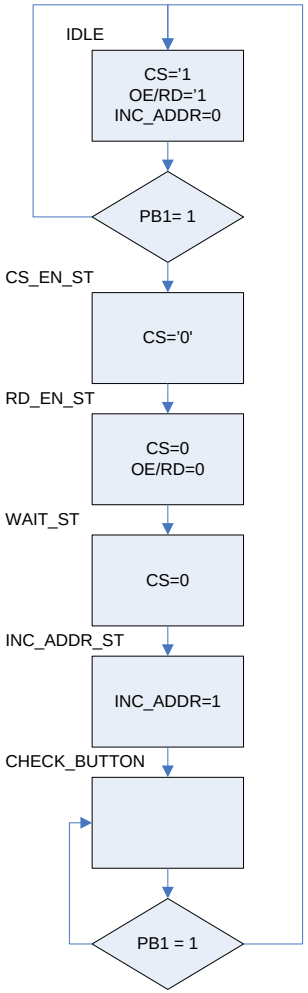
Utskrift av data til consol (transcript) vindu

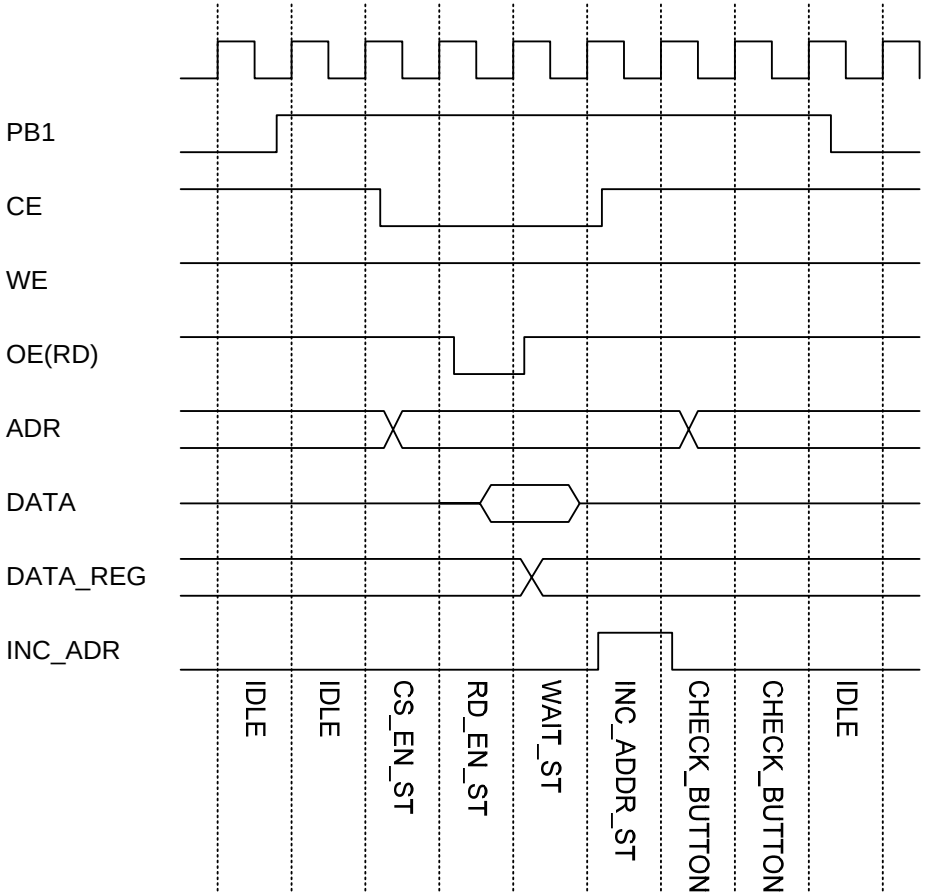
```
VSIM 30> run 1us
# Z_out:  0000\0h  co_out:  0  Current time: 60 ns
# Z_out:  0001\1h  co_out:  0  Current time: 120 ns
# Z_out:  1110\Eh  co_out:  1  Current time: 180 ns
# Z_out:  1111\Fh  co_out:  1  Current time: 240 ns
# Break in Process p1 at C:/IFI/INF3430/H2007/Foreles/f8/vhdl_src/tb_alt.vhd line 91
```

Lesing fra SRAM-Arkitektur

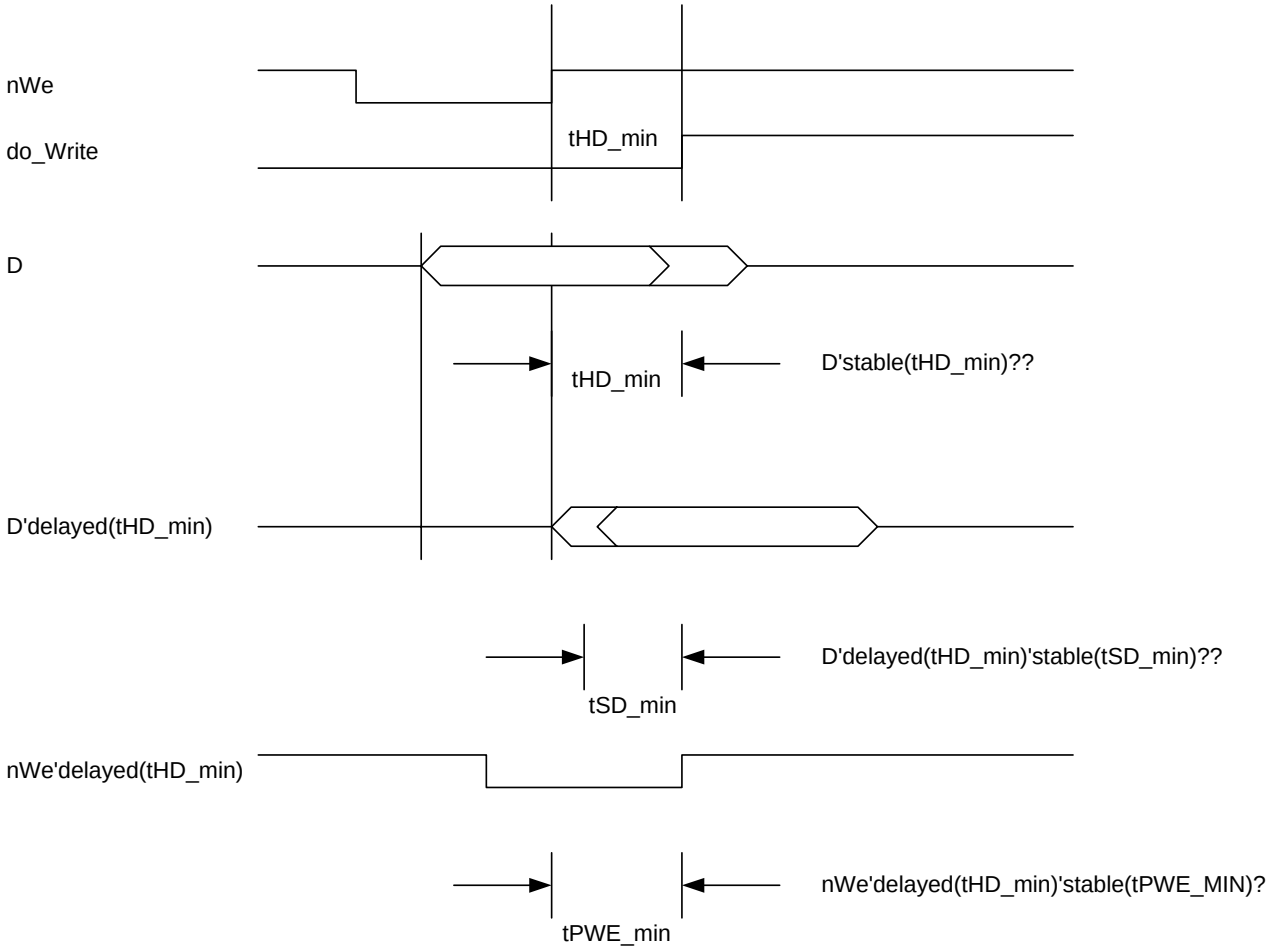


Lesing fra RAM, ASM og timing





Sjekk av noen timingparametre i SRAM-modellen



Oppsummering

- Event driven simulation
  - Delta delay
  - Eventer og vedlikehold av eventkøer
- Simulering av VHDL-modeller
- Selvtestende testbenker
  - Sjekker utganger opp mot en kjent fasit
  - Finnes egne verktøy for å lage fasit, f.eks HDL bencher
- Fil-operasjoner
  - Stimuli fra input-fil
  - Output til resultat-fil. Kan benyttes i testbenk for å sammenligne resultat mellom forskjellige simuleringer, f.eks. før og etter syntese.
- Eksempel på SRAM modell og simulering av lesing fra denne
  - Fylle ram med kjente data
  - Timing sjekk i modell