E2DSD
Øvelse # - Navn på øvelse

Gruppe #
Navn1 - Studienummer
Navn2 - Studienummer

21 januar 2018

1 Øvelsestemplate

1.1 Introduktion

Her besvares spørgsmålet: ”Hvorfor?”. Baseret på din forståelse af øvelsesvejledningen, hvad ser du som formålet med øvelsen? 3-5 linjer. Ligesom det indledende i en avisartikel, skal den være kort, præcist og give læseren lyst til at læse videre.

1.2 Design og Implementering

Her besvares spørgsmålet: ”Hvordan?”. Du beskriver her hvordan du har løst opgaven. Har du skrevet kode, indsættes den her. Koden skal indeholde passende kommentarer for det som ikke er trivielt eller ikke er set før. Indeholder din løsning hardware, beskrives denne også her.

1.3 Resultater

Her besvares spørgsmålet: ”Hvad?”. Resultater i form af simuleringwaveforms, RTL- og Technology Map Views og fotos fra tests medtages her. Vigtige observationer fremhæves nøgternt, f.eks. med cirkle og pile på figurer og med tilhørende tekst. Du skal ikke tolke dine resultater her, blot præsentere dem.

1.4 Diskussion

Dette afsnit diskuterer og tolker på resultaterne og uddyber besvarelsen af spørgsmålet: ”Hvad?” Fremhæv og forklar sammenhænge fundet i resultaterne (Eller ikke fundet!).

F.eks. hvordan afspejles implementeringen i resultaterne? Hvordan mapper kode til RTL-Views? Hvordan fremgår det af simuleringen at koden implementerer det ønskede? Hvordan illustrerer fotoet det? Hvad er sammenhængen imellem emne, implementering og testresultater? Hvis en opgave har flere underopgaver og dermed resultater, kan man, hvis det giver mening, lægge et diskussionsafsnit ved hvert resultat i stedet for et samlet afsnit.

1.5 Konklusion

Samlet, kort konlusion på øvelsen. Hvilke mål blev opfyldt, hvilke mål kom i ikke i mål med og hvorfor.

2 Øvelse: Half-, full- og ripple-adders

Dette er et eksempel på en journal lavet via templaten.

2.1 Introduktion

Denne øvelse undersøges implementeringen af half- / full- og ripple-carry adders. Formålet er at lære de forskellige kodningsformer i VHDL, samt at arbejde med aritmetik i VHDL. Design og Implementering I øvelse 1 implementeres en 4-bit adder. Den kommenterede source kode er vist i snippet herunder. Kommentarer benytter Doxygen (doxygen.org) syntax.

```
1  ----------------------------------------------------------
2  --! @file exercise1.vhd
3  --! @brief Top-level design to map four bit adder to
4  --! @brief DE-2 board
5  ----------------------------------------------------------
6  library ieee;
7  use ieee.std_logic_1164.all;
8
9  --! Exercise 1 entity declaration
10 entity exercise1 is
11     port(
12           sw : in  std_logic_vector(15 downto 0);   --! Input
       Switches
13           ledr : out std_logic_vector(15 downto 0)
14     ); --! Outpt Red Leds
15 end exercise1;
16
17 --! @brief Architecture Structural
18 --! @details This implements mapping of 4-bit adder to the DE-2
       board pins
19 architecture strutural of exercise1 is
20 begin
21     DUT : entity work.four_bit_adder port map(
22               a => sw(3 downto 0),
23               b => sw(7 downto 4),
24               cin => sw(8),
25               Sum => ledr(3 downto 0),
```

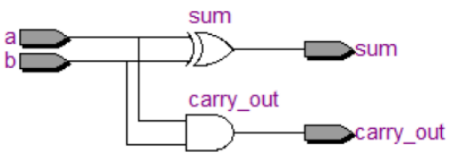

Fig. 1: *Kredsløb programmeret med dataflow strukturen.*

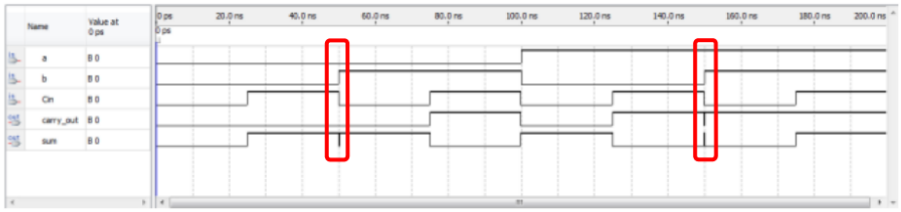

Fig. 2: *Timing simulering for adder. (Bemærk at opløsningen for figuren ikke er god nok her!)*

```
26                 cout => ledr(4)
27             );
28 end;
```

2.2 Resultater

I øvelsen undersøges outputtet af implementeringerne: Hvad bliver koden til rent skematisk? Hvad er det simulerede output? Og hvad sker der når vi tester på DE2-boardet?

Af RTL-view'et illustreret i figure 1 fremgår det at det at half-adder koden skrevet i dataflow style syntetiseres til hhv. en xor- og en and- gate.

2.3 Diskussion

Dataflow er den mest low-level kodningsform i VHDL. De to dataflow udtryk fra koden i afsnit 2 oversættes direkte til to gates som det ses af figure 1. Det vil sige at der er tæt sammenhæng mellem det skrevne i dataflow og implementeringen, hvilket vi også forventede.

2.3.1 Timing simulering

I simuleringen er alle kombinationsmuligheder for A og B anvendt, og det fremgår at outputtene følger funktionen for en full-adder. Hvor inputtene skifter samtidigt, forekommer der spikes (eller glitches) på outputtet, se figure 2.

2.4 Diskussion

Implementeringen af full-adderen lever funktionelt op til hensigten. Der er midlertidig spikes på outputtene når inputtene skifter samtidigt. Dette sås ikke for den funktionelle simulering og årsagen hertil skal findes i routing delay fra FPGA pins til.... Statisk-0/1 Hazard... Karnough...

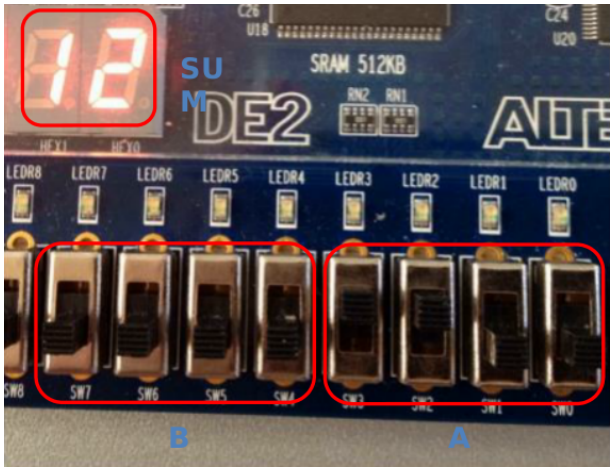

Fig. 3: *Foto af DE2-board, for test af adder.*

2.4.1 Test på DE2-Board

Adderen blev mappet til switches og 7-segment displays, som illustreret i figure 3, bygget og programmeret ned på DE2-boardet. Ved at sætte switches, skiftede display til den korrekte sum.

2.5 Konklusion

Adderen fungerer funktionelt som den er designet i VHDL. Adderen kan lave overflow, men implementeringen tager heller ikke højde for dette. De spikes som sås i simuleringen, figure 2, ses ikke i testen. De er for kortvarige til at kunne ses med det blotte øje på displayene.