Mads Friis Bornebusch, s123627
Kristian Sloth Lauszus, s123808

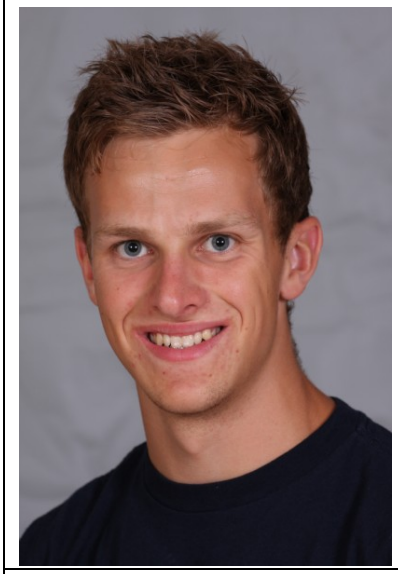
22. maj 2013

02139 Digital Elektronik 2

Sodavandsautomat projekt

Gruppe H02

Mads Bornebusch
s1233627



Kristian Sloth Lauszus
s123808

Rapporten og øvelsen har været et samarbejde mellem Mads Friis Bornebusch og Kristian Sloth Lauszus hvor de begge har været med i hele processen. Mads fokuserede mest på analysen af dermed på tilstandsdiagrammerne og datapathen for CPU'en, men var også med til programmering af Basys 2 boardet. Dette stod Kristian dog primært for, da han før har arbejdet med programmering af elektronik deriblandt af FPGA'er. Både Kristian og Mads har derfor bidraget til alle dele af rapporten, mens det samtidig er forsøgt at udnytte deres respektive styrker.

Abstract

The goal of this project was to design and implement a control circuit for a vending machine. The vending machine should take 2kr and 5kr coins emulated by two pushbuttons. Six switches can set the price. When the user presses the buy button the vending machine will assert the signal release_can if the sum of coins inserted is greater than or equal to the price or the signal alarm will be asserted if the sum is less than then price. The machine will subtract the price from the sum when a can is being released. The price and the sum of coins inserted should be displayed on a multiplexed seven segment display.
We designed a datapath for the CPU of the circuit and a FSM to control the datapath. Furthermore we implemented some extensions to our design including displaying sum and price as decimal numbers on the seven segment display, scrolling the text COLA or PEPSI on the display when a beverage is bought and a serial interface to send data to a PC. The serial interface sends the sum whenever it is changed. We implemented this design in VHDL and simulated the CPU and the serial interface in Modelsim using a test bench written in VHDL. Furthermore the design was programmed to a Basys2 FPGA board and tested thoroughly. The circuit with all its extensions worked as intended and we can therefore conclude that the project was a success.

Indholdsfortegnelse

Indledning og problemformulering

Formålet med denne opgave var at designe og implementerer en sodavandsautomat på et FPGA udviklingsboard af typen Basys2[1] fra Digilent.

Ifølge opgavespecifikationerne skulle man kunne bruge to forskellige mønter ind i automaten hhv. en 2 og 5 krone. Disse skulle dog blot emuleres vha. af to knapper på boardet. Derudover skulle man kunne sætte prisen på en sodavand vha. en række skydeknapper. De bestemte således hvor meget den pågældende sodavand skulle koste. I tilfælde af at der blev trykket på knappen buy skulle vores system være i stand til at checke om summen af mønternes værdi i maskinen var større eller lig med prisen.

På den baggrund skulle den enten frigive en sodavand eller sætte en alarm der fortalte brugeren at der ikke var nok mønter i automaten. Derudover skulle vi også kunne generer en manuel clock, så vi kunne se outputtet for hver eneste clock.

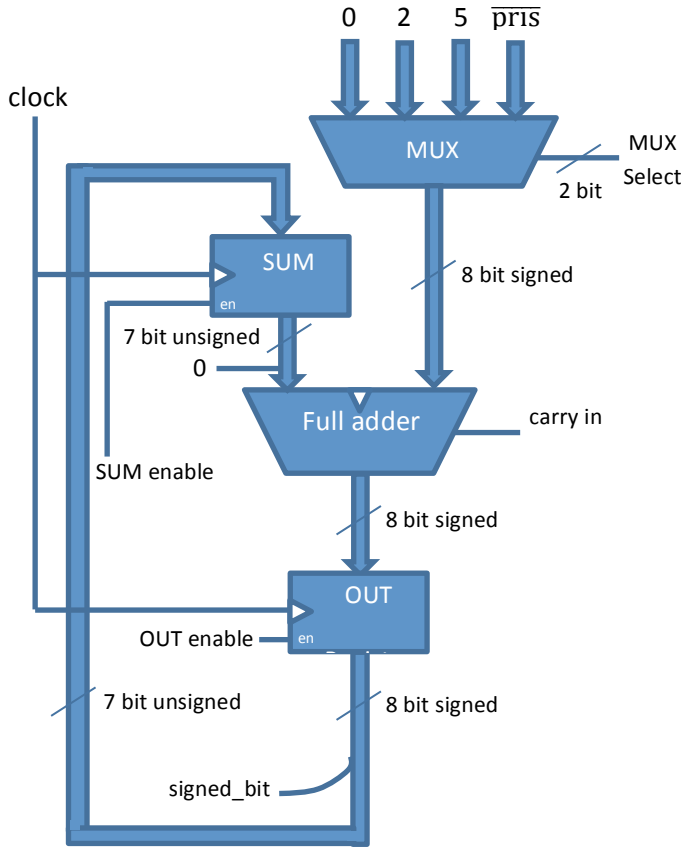
Den samlede sum af mønterne samt prisen skulle derudover vises på fire såkaldt seven segment displays.

Udover at vise dette brugte vi også displayet til at vise en række beskeder alt efter hvad personen købte og hvis der ikke var nok penge i automaten til at foretage købet. Mere om dette i afsnittet Udvidelser.

CPU'en

De funktioner CPU'en skal varetage er at den skal kunne lægge 2 kr eller 5 kr til en sum og at den skal kunne trække prisen fra summen når brugeren køber en sodavand.

Dette kan realiseres med en datapath og en tilstandsmaskine til at styre hvad der sker i datapathen. For at kunne addere og subtrahere tal skal vi bruge en adder. Derudover er der brug for et register til at holde værdien fra adderens output og et register til at holde summen, som også er den ene operand for adderen. Til sidst er der brug for en multiplexer til at vælge den anden operand i adderen. Datapath'en er vist på Figur 1.



Figur 1, CPU datapath'en

Med to registre, en adder og en MUX er det muligt at udføre de operationer CPU'en skal. MUX'en vælger mellem værdierne til adderen som afhængig af

[1] http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS2

styresignalet add/sub enten lægges til eller trækkes fra summen. Outputtet fra adderen læses ud i Out-registeret og læses derfra ind i Sum-registeret hvis det er positivt.

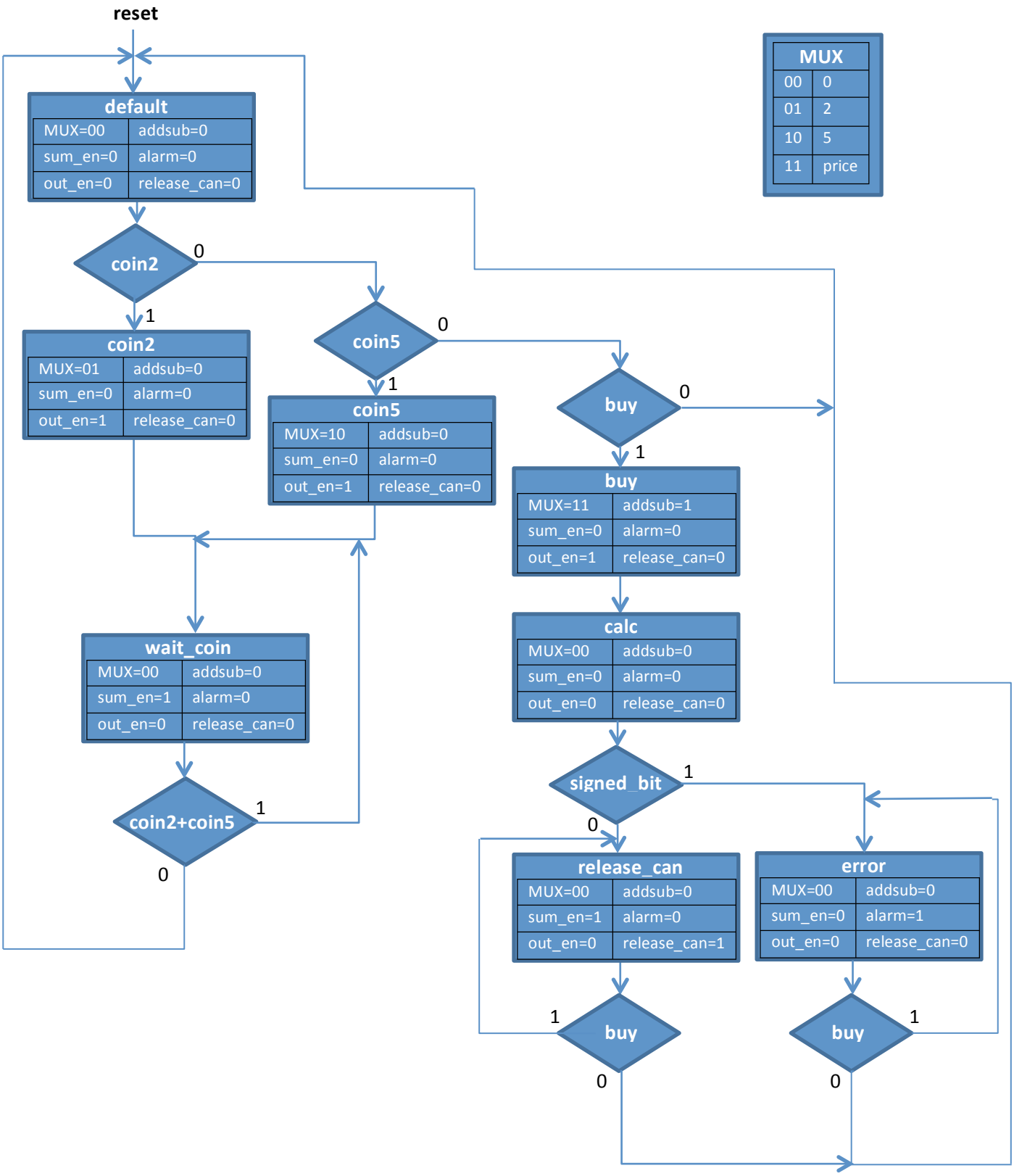
Addition af binære tal

Der findes forskellige repræsentationer af binære tal når man også skal kunne repræsentere negative tal. Tre af repræsentationerne er sign and magnitude, 1's compliment og 2's compliment[2]. Hvis man bruger 2's compliment kan man bruge en almindelig adder til både at lægge tal sammen og trække dem fra hinanden. Med 2's compliment fås den negative værdi af et tal ved at invertere alle bitsene og derefter lægge 1 til. Den mest betydende bit er så fortegnet således at et 0 betyder at tallet er positivt og et 1-tal betyder at tallet er negativt. Når man ved dette, kan man ved hjælp af xor-gates let lave en adder der kan lægge til eller trække fra[3]. Dette har vi dog ikke brug for her. Prisen skal altid trækkes fra og værdierne 2 og 5 skal altid lægges til. Man kan derfor invertere prisen og hvis carry in på adderen så sættes til 1, samtidig med at MUX'ens output er pris, vil prisen blive trukket fra. Efter out-registeret kan man så se på den mest betydende bit om resultatet er positivt eller negativt.

FSM til styring af CPU'ens datapath

Datapathen har styresignalerne MUX select, SUM enable, OUT enable og add/sub. Disse signaler skal drives af en FSM der således styrer datapathen. FSM'en har inputs'ene coin2, coin5, buy og signed_bit. Outputtet fra FSM'en er styresignalerne til datapathen og de to signaler alarm og release_can. Et ASM-chart for Moore-type FSM'en er vist på Figur 2. Figuren viser også hvad de forskellige signaler til MUX'en vælger. Ved reset er starttilstanden default state. Så længe signalerne coin2, coin5 eller buy ikke sættes høj bliver FSM'en i denne tilstand. Når FSM'en får signalet coin2 eller coin5 går den over i den tilsvarende tilstand hvor summen adderes med 2 eller 5 og dette lægge i output-registeret. Efter dette følger tilstanden wait_coin hvor resultatet lægges over i sum-registeret. Denne tilstand går maskinen først ud af, når signalerne coin2 og coin5 begge går lav igen. Ved at gøre det på denne måde, undgår man at der f.eks. lægges 2 til for hver clocksyklus signalet coin2 er høj. Der lægges således kun 2 til summen én gang når coin2 går høj. Når signalet buy går høj og brugeren forsøger at købe noget, går tilstandsmaskinen i tilstanden buy hvor prisen trækkes fra summen. Efter dette har vi tilstanden calc. Denne tilstand skal sikre at resultatet af beregningen er udført og gemt i output-registeret før vi tjekker værdien af signed_bit. Dette er sikret når klokken slår og tilstanden er der således bare for at lade klokken slå før vi lader signed_bit bestemme næste tilstand. Hvis signed_bit er 0 er tallet positivt og vi går så over i tilstanden release_can hvor resultatet gemmes i sum-registeret og release_can sættes høj. Brugeren har således købt en sodavand. Denne tilstand bliver FSM'en i indtil buy går lav igen. Hvis signed_bit derimod er 1 og tallet er negativt, går tilstandsmaskinen i tilstanden error. Her sættes alarm høj og resultatet gemmes ikke i sum-registeret.

[2] Brown & Vranesic: Digital Logic, side 262
[3] Brown & Vranesic: Digital Logic, side 266



Figur 2, ASM-chart for FSM'en til styring af datapathen i CPU'en

Test af CPU'en

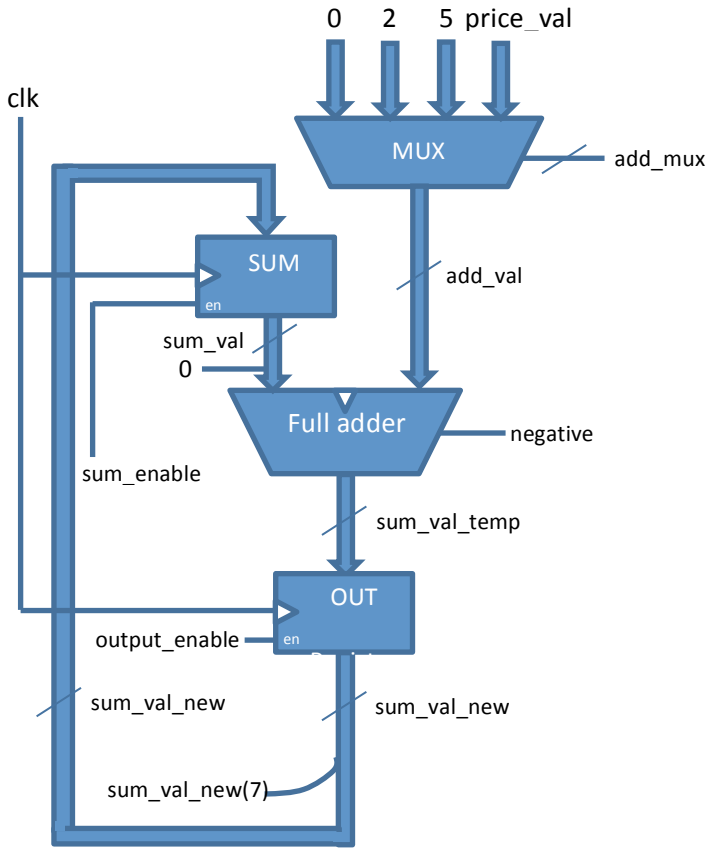
Outputtet fra simuleringen af CPU'en er vist i "Appendix C – Simulering af CPU'en".

Datapathen med signalnavnene fra VHDL-koden er vist på Figur 3. VHDL-koden for test bench'en kan ses i Appendix B – et udsnit af denne kan ses i Figur 4.

Det første der sker i testsekvensen efter kredsløbet er blevet nulstillet, er at prisen sættes til 7.

Prisen er nu 7 og sum er 0. Nu sættes buy høj. Ud fra simuleringen i Appendix C kan ses at FSM'en nu går i buy-state og calc state og at sum subtraheret prisen udregnes da styresignalet negative til muxen er 1.

Resultatet af dette (som går ud på sum_val_temp og derefter sum_val_new) er hexadecimalt 0xF9 hvilket svarer til 11111001 i binær 2's compliment eller -7 i decimal. Tallet er altså negativt og næste tilstand bliver da alarm_state hvor alarm er høj så længe buy er høj.



Figur 3, Signalnavne i datapath'en

Nu sættes signalet coin2 høj og FSM'en går i coin2_state og wait_coin_state mens add_val sættes til 2 således at der lægges 2 til summen. I wait_coin_state går sum_enable høj således at resultatet læses ind i sum-registeret og bliver den nye sum_val. Signalet coin2 sættes nu lavt igen. På tilsvarende måde sættes coin5 høj så der lægges 5 til summen. Det er her værd at bemærke at signalet add_mux her er 10 (2 i decimal og 0x02 i hex) således at værdien af add_val bliver 5.

Nu sættes coin5 lav og buy sættes høj. Prisen trækkes nu igen fra summen. Her giver resultatet 0 og tilstandsmaskinen går derfor i tilstanden release_can_state hvor signalet release_can sættes høj.

```
-- Stimulus process
stim_proc: process
begin
   -- insert stimulus here

   reset <= '1';
   wait for clk_50_period*5;
   reset <= '0';
   wait for clk_50_period*5;

   price <= "000111";
   wait for clk_50_period*5;
   buy <= '1';
   wait for clk_50_period*5;
   assert alarm = '0' report "error: alarm should be 1";
   buy <= '0';

   coin2 <= '1';
   wait for clk_50_period*5;
   coin2 <= '0';
   wait for clk_50_period*5;
   coin5 <= '1';
   wait for clk_50_period*5;
   coin5 <= '0';
   buy <= '1';
   wait for clk_50_period*5;
   assert release_can = '0' report "error: release_can should be 1";

   wait;
end process;
```

Figur 4 – Udsnit af koden til simuleringen af CPU'en.

Simuleringen viser altså at CPU'en virker som det var tiltænkt. Desuden er det her tydeligt at se hvorfor der er brug for tilstanden calc_state til at læse sum_val_temp ind i output-registeret. Klokken skal slå før det læses ind og dermed kommer på signalet sum_val_new.

Udvidelser

Vi valgte at lave en række udvidelser, da vi havde ekstra tid og lyst. I dette afsnit vil vi gennemgå de forskellige udvidelser og forklare hvordan vi implementerede dem.

Display

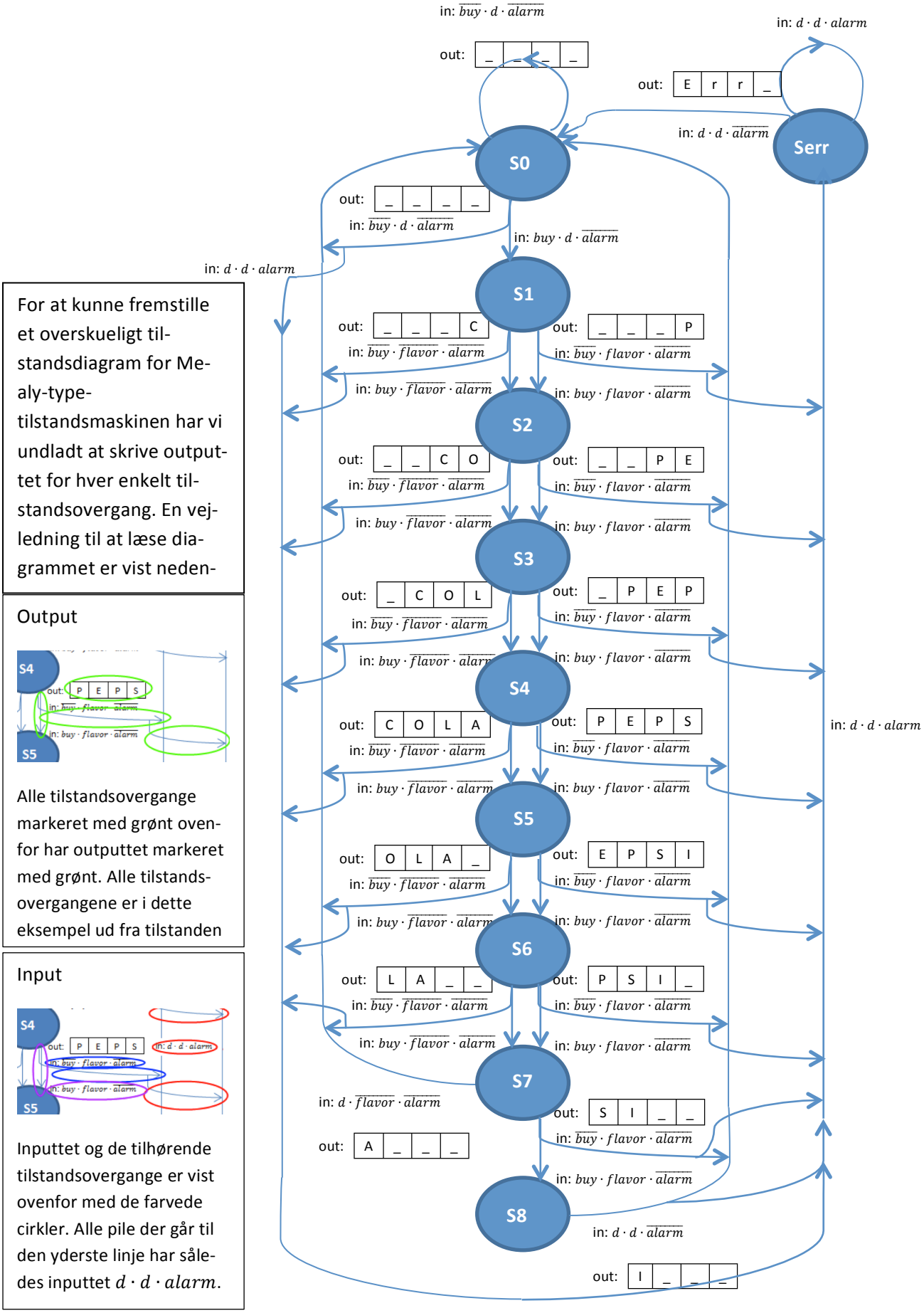
Den første udvidelse vi lavede var at printe tallene som base 10 i stedet for i base 16 (hexadecimal). For at konverterer det binære tal til et decimal tal lavede vi en tabel vha. Filen "GenBcdConv.java" skrevet af Martin Schoeberl. Vi modificerede den dog lidt, så vi kunne konvertere et 7-bit binært tal til et 8-bit decimal tal hvor det højeste var 99. Den færdige tabel kan ses i Appendix A "bcdtab.vhd".

Dette muliggør at man nemt kan konvertere de binære tal "price" og "sum" til fire decimal tal, så vi let kan vise dem på displayet. Hvor "price" er et 6-bit nummer bestemt af 6 swithces på boardet, mens "sum" bestemt vha. knapperne: buy, coin2 og coin5, som specificeret i projektspecifikationerne. Sum kan dog maksimum være 99. I øjeblikket vil den blot holde den maksimale værdi 99, hvis der trykkes på coin2 eller coin5 igen – det kunne tænkes at mønten ville blive returneret i en virkelig implementering.

Den næste udvidelse vi lavede, var noget mere kompliceret da vi ønskede at kunne scrolle en tekst fra højre mod venstre. Vi valgte således at vi ville skrive "COLA" og "PEPSI". Disse to blev valgt vha. bit'en flavor, som i den færdige implementering blev bestemt af om prisen var lige eller ulige – se Appendix A "display_manager.vhd". Derudover valgte vi også at den skulle skrive "Err" hvis "alarm" var høj. Tilstandsdiagrammet for dette er vist på Figur 5. Tilstandsdiagrammet er for en Mealy-type FSM der afhængigt af inputtet flavor scroller tekst over displayet når buy er høj og alarm er lav. Der skal bruges 9 tilstande til at scrolle PEPSI over displayet og tilstandsmaskinen har derfor 9 tilstande, S0 til S8. Ud over dette var der tilstanden Err hvor displayet skulle vise Err. Da COLA kan scrolles over displayet ved brug af 8 tilstande bliver maskinen en Mealy-type FSM. Outputtet på overgangen fra hver tilstand er afhængigt af inputtet flavor og da der bruges en tilstand mindre til at skrive COLA er næste tilstand også afhængig af flavor. Som en hjælp til at forstå tilstandsdiagrammet i Figur 5 er den tilhørende tilstandstabel vist på Figur 6. Ud fra denne kan det ses at hvis flavor ændres mens tilstandsmaskinen er i andre tiltande en S0 kan det give et utilsigtet output. Flavor er dog den mindst betydende bit fra prisen som vi antager ikke ændres. Hvis den alligevel ændres vil det aldrig kunne føre til at tilstandsmaskinen "fanges" i en tilstand.

Koden for den komponent der skriver tekst i displayet kan ses i Appendix A "display_text.vhd".

For at kunne vælge imellem om displayet skulle vise hhv. summen og prisen eller teksten, implementerede vi et topmodul kaldet "display_manager.vhd". Dette gør den ved at styre hvilke af de to der skal "forbindes" til displayet – i dette tilfælde afhang det af om brugeren trykkede på knappen "buy" eller ej.



For at kunne fremstille et overskueligt tilstandsdiagram for Mealy-type-tilstandsmaskinen har vi undladt at skrive outputtet for hver enkelt tilstandsovergang. En vejledning til at læse diagrammet er vist neden-

Output

Alle tilstandsovergange markeret med grønt ovenfor har outputtet markeret med grønt. Alle tilstandsovergangene er i dette eksempel ud fra tilstanden

Input

Inputtet og de tilhørende tilstandsovergange er vist ovenfor med de farvede cirkler. Alle pile der går til den yderste linje har således inputtet $d \cdot d \cdot alarm$.

Figur 5 – Tilstandsdiagram for "display_text.vhd".

Nuvæ-rende tilstand	Næste tilstand				Output	
	$\overline{buy} \cdot d \cdot \overline{alarm}$	$buy \cdot flavor \cdot \overline{alarm}$	$buy \cdot \overline{flavor} \cdot \overline{alarm}$	$d \cdot d \cdot alarm$	$d \cdot flavor \cdot d$	$d \cdot \overline{flavor} \cdot d$
S0	S0	S1	S1	Serr	_ _ _ _	_ _ _ _
S1	S0	S2	S2	Serr	_ _ _ C	_ _ _ P
S2	S0	S3	S3	Serr	_ _ C O	_ _ P E
S3	S0	S4	S4	Serr	_ C O L	_ P E P
S4	S0	S5	S5	Serr	C O L A	P E P S
S5	S0	S6	S6	Serr	O L A _	E P S I
S6	S0	S7	S7	Serr	L A _ _	P S I _
S7	S0	S8	S0	Serr	A _ _ _	S I _ _
S8	S0	S0	S0	Serr	I _ _ _	I _ _ _
Serr	S0	S0	S0	Serr	E r r _	E r r _

Figur 6 – Tilstandstabel for "display_text.vhd".

Serial interface – UART

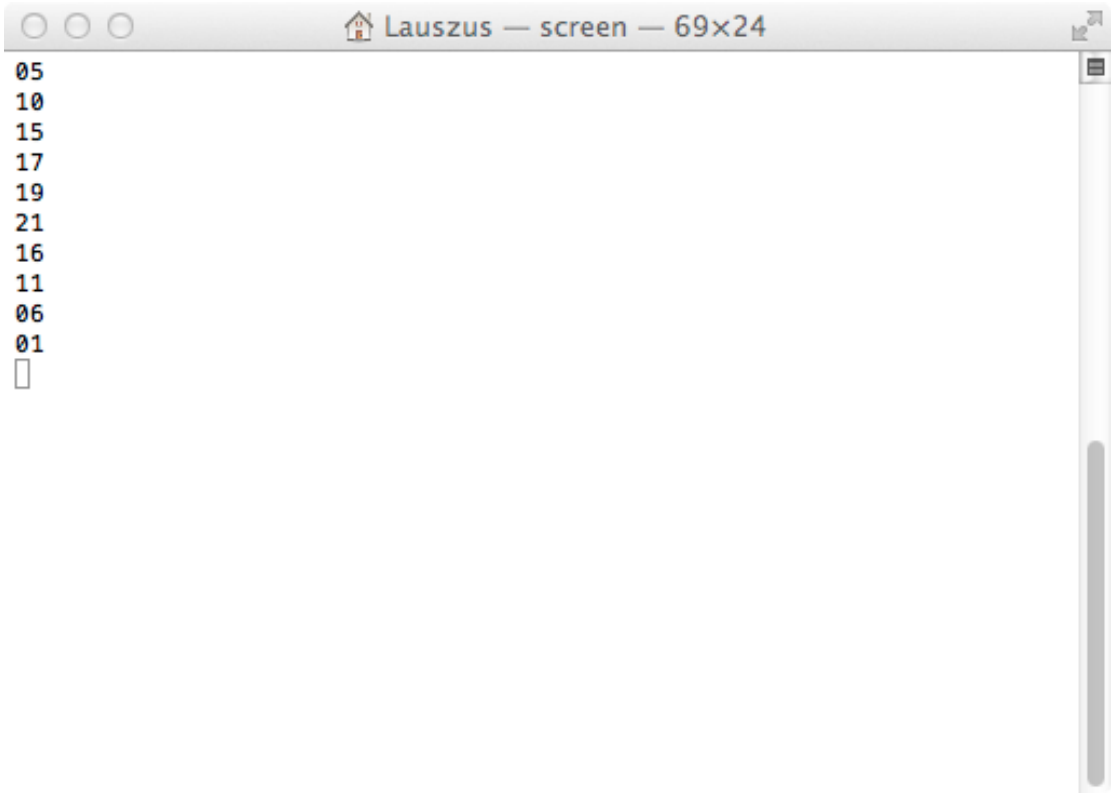
Den næste udvidelse vi lavede var at lave en serielt interface. I øjeblikket kan den dog kun sende data. Dette gør den med en baudrate på 115200. En pakke består af 8 bits med ingen parity bit og én stop bit også forkortet: 8N1. En forklaring på hvordan serielkommunikation virker kan ses i følgende kilder[4] og [5].

For at sende dette til en computer brugte vi en såkaldt USB til seriel konverter af typen FT230X[6], vha. af et breakout board[7]. Man behøver blot at forbinde D12 fra Basys2 boardet til RX på FT230X chippen samt forbinde GND mellem de to enheder. Derefter kan man åbne en valgfri seriel terminal for at se det sendte data.

FPGA boardet sender værdien af "sum" hver gang den ændrer sig efterfulgt af et såkaldt carrige return der sætter markøren helt til venstre og et line feed der sætter markøren ned på næste linje. Figur 7 viser et eksempel på dette – først er der trykket 3 gang på coin5, så 3 gang på coin2 derefter er der trykket 4 gange på buy hvor prisen var sat til 5.

Den færdige kode kan ses i Appendix A"serial_interface.vhd".

[4] http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter#Character_framing
[5] Martin Schoeberl, Kursus 02139 Digitalelektronik 2, Forelæsning 12: Interface and Interface Protocols, 2013
[6] http://www.ftdichip.com/Products/ICs/FT230X.html
[7] http://jim.sh/ftx/



Figur 7 – Eksempel på serielt output fra FPGA boardet.

Simulering af serial interface

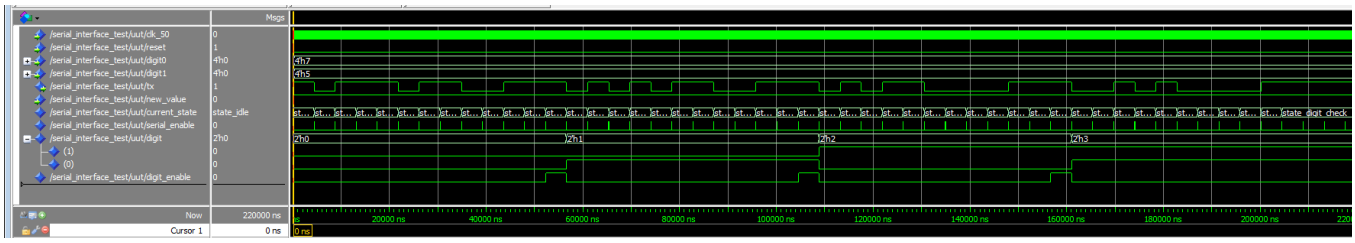
For at tjekke om vores serielle interface virkede som det skulle, valgte vi at simulere filen ”serial_interface.vhd” i Modelsim vha. testbench’en ”serial_interface_test.vhd”. Det udsnit af koden der sætter de forskellige inputs kan ses i Figur 8. Det ønskede output er således ASCII[8] karaktererne ’7’ og ’5’ efterfulgt af carriage return og line feed, der har ASCII værdierne hhv. 0x0D og 0x0A.

[8] http://www.asciitable.com/

```
stim_proc: process
begin
    reset <= '1';
    wait for clk_50_period*10;
    reset <= '0';
    wait for clk_50_period*10;

    digit0 <= "0111";
    digit1 <= "0101";
    new_value <= '1';

    wait;
end process;
```

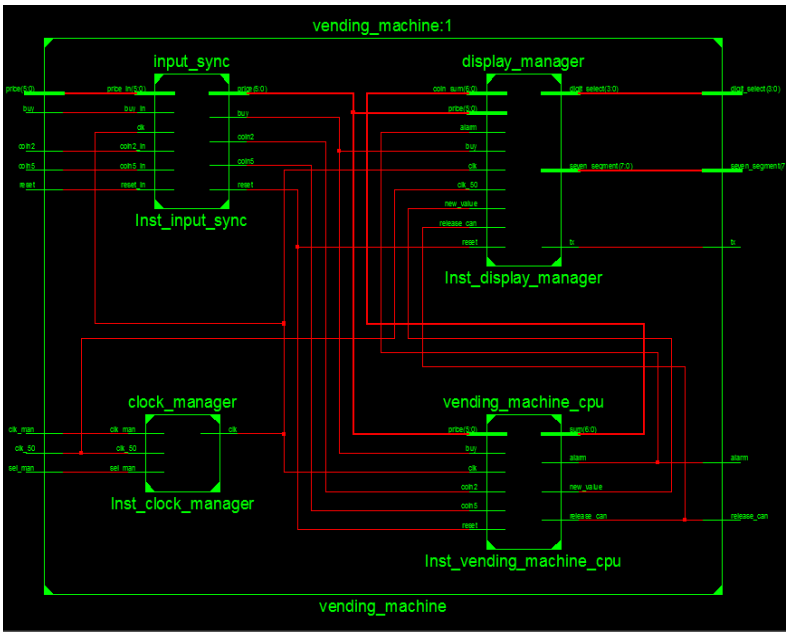
Figur 8 – Udsnit af simulerings koden

Det fulde output kan ses i Figur 9 – et større billede kan ses i ”Appendix D – Simulering af serielkommunikation”. Bemærk at for at konverterer et tal mellem 0-9 fra decimal til ASCII sættes bit 4 og 5 til 1. Som man kan se sættes TX først lav for at indikere at der bliver sendt en ny pakke. Derefter er den høj i tre clock cyklusser og lav i én, da det er den binære værdi for 7, ”0111” – bemærk at tallet sendes LSB først. Derefter går den høj i to clock cyklusser for at konverterer tallet til ASCII. De næste to bits sættes blot lav. Derefter sættes TX høj i tre clock-cyklusser og næste pakkes sendes. Denne er den samme bortset fra bit1 der sættes lav, da 5 der har den binære værdi ”0101”. Derefter sendes et carrige return der har den binære værdi ”1101” og derefter et line feed med den binære værdi ”1010”.



Figur 9 – Simulering af ”serial_interface.vhd”.

RTL-schematics af implementeringen



Figur 10, RTL-schematic fra Xilinx af top-modulet i sodavandsautomaten

På Figur 10 er RTL-diagrammet vist for topmodulet i sodavandsautomaten. Det kan ses at den består af fire dele nemlig input synchronizeren, clock manageren, display manageren og CPU'en.

Syntetiseringsrapporten for display_text er vist på Figur 11.

Af figuren kan det ses at Xiinx har fundet en FSM med 10 states. Det svarer til de states vi havde designet på tilstandsdiagrammet for display_text. Der er dog også fundet en adder og et 8 bit register. Dette er ikke en del af desginet fra tilstandsdiagrammet men er kommet fordi vi har lavet et Clock enable signal (display_enable som Xilinx også har fundet). Dette har vi gjort for at displayet skal scrolle langsom og ikke med clockfrekvensen på 763 Hz. Vi har således en tæller der lægger 1 til for hver clockpuls og giver en puls ud for hver 200 tællinger. Dette er lavet med en adder, med registre og med logik.

```
Synthesizing Unit <display_text>.
    Related source file is "C:/Users/Mads Borne-
busch/Documents/GitHub/Basys2/VendingMachine/src/display_text.vhd".
    Found finite state machine <FSM_1> for signal <current_state>.
    -----------------------------------------------------------------------
    | States             | 10                                             |
    | Transitions        | 29                                             |
    | Inputs             | 3                                              |
    | Outputs            | 10                                             |
    | Clock              | clk                          (rising_edge)     |
    | Clock enable       | display_enable               (positive)        |
    | Reset              | reset                        (positive)        |
    | Reset type         | asynchronous                                   |
    | Reset State        | state0                                         |
    | Power Up State     | state0                                         |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Found 1-of-4 decoder for signal <digit_select>.
    Found 8-bit 4-to-1 multiplexer for signal <seven_segment>.
    Found 8-bit adder for signal <cnt$addsub0000> created at line 85.
    Found 8-bit register for signal <cnt_reg>.
    Found 2-bit up counter for signal <segment_selector>.
    Summary:
         inferred   1 Finite State Machine(s).
         inferred   1 Counter(s).
         inferred   8 D-type flip-flop(s).
         inferred   1 Adder/Subtractor(s).
         inferred   8 Multiplexer(s).
         inferred   1 Decoder(s).
Unit <display_text> synthesized.
```

Figur 11, Syntetiseringsrapporten for display_tekst.vhd

Syntetiseringsrapporten for CPU'en er vist på Figur 12. Her har Xilinx fundet en FSM med 8 tilstande og 16 overgange. Det er det samme antal tilstande og overgange der er i ASM-chartet for CPU'ens FSM. De fire inputs er coin2, coin5, buy og signed_bit.

I datapathen er der fundet to registre for sum_val og sum_val_new hvilket svarer til de to registre for sum og output fra adderen. Derudover er der fundet to greater/lesser-comparatorer. Den ene er kommet fra at vi sammenligner summen med 99 da vi ikke lægger noget til når summen bliver 99. Den anden er sammen med en ekstra adder kommet af den lidt klodsede måde vi har skrevet VHDL-koden for adderen. Vi kunne ikke få adderen til at regne med 2's compliment negative tal og vi valgte derfor at lave en if-statement der bestemte om adderen trak fra eller lagde til. Implementeringen på FPGA er altså blevet dyrere end den implementering vi havde planlagt.

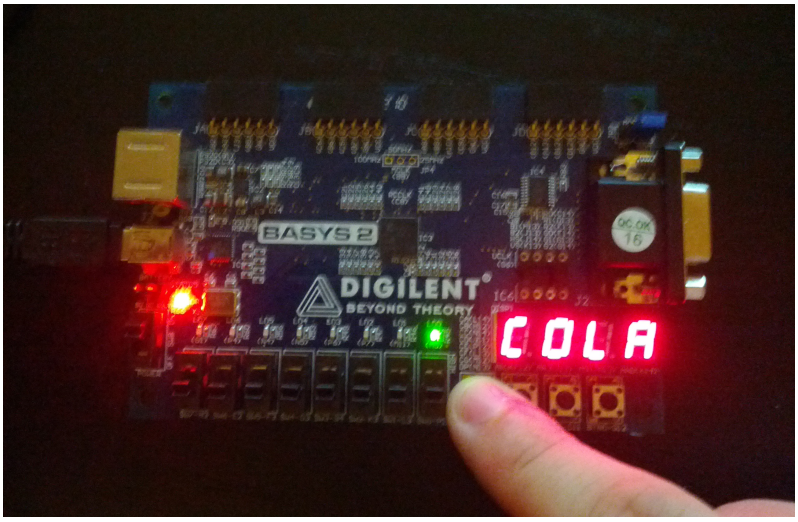
```
Synthesizing Unit <vending_machine_cpu>.
    Related source file is "C:/Users/Mads Borne-
busch/Documents/GitHub/Basys2/VendingMachine/src/vending_machine_cpu.vhd".
    Found finite state machine <FSM_0> for signal <current_state>.
    -----------------------------------------------------------------------
    | States             | 8                                              |
    | Transitions        | 16                                             |
    | Inputs             | 4                                              |
    | Outputs            | 8                                              |
    | Clock              | clk                          (rising_edge)     |
    | Reset              | reset                        (positive)        |
    | Reset type         | asynchronous                                   |
    | Reset State        | default_state                                  |
    | Power Up State     | default_state                                  |
    | Encoding           | automatic                                      |
    | Implementation     | LUT                                            |
    -----------------------------------------------------------------------
    Using one-hot encoding for signal <add_mux>.
    Found 1-bit register for signal <new_value>.
    Found 8-bit comparator less for signal <current_state$cmp_lt0000> creat-
ed at line 52.
    Found 7-bit register for signal <sum_val>.
    Found 8-bit register for signal <sum_val_new>.
    Found 7-bit adder for signal <sum_val_temp$addsub0000> created at line
125.
    Found 7-bit comparator greater for signal <sum_val_temp$cmp_gt0000> cre-
ated at line 125.
    Found 8-bit addsub for signal <sum_val_temp$share0000> created at line
122.
    Summary:
        inferred   1 Finite State Machine(s).
        inferred  16 D-type flip-flop(s).
        inferred   2 Adder/Subtractor(s).
        inferred   2 Comparator(s).
Unit <vending_machine_cpu> synthesized.
```

Figur 12, Syntetiseringsrapporten for vending_machine_cpu.vhd

Diskussion

Syntetiseringsrapporten fra Xilinx viste at vores implementering på FPGA'en ikke helt svarede til den tænkte implementering. Dette skyldtes at vi ikke kunne få det ti at virke at trække 2's-compliment tal fra hinanden med + opratoren. Selvom det ene tal var negativt ville de altid blive lagt sammen. Vi valgte derfor at skrive koden på en måde der gav os en ekstra adder og en "mindre end"-comparator. Realiseringen på FPGA'en blev derfor en smule dyrere end hvad vi havde tænkt.

Vi fik implementeret en del udvidelser til sodavandsautomaten blandt andet en række udvidelser til displayet, bl.a. så den kunne skrive summen og prisen som et normalt base 10 decimal tal, men også så den kunne skrive en besked på displayet. Figur 13 viser et billede taget lige da "COLA" blev vist på displayet.



Figur 13 – Billede taget lige da "COLA" blev vist på displayet.

Derudover lavede vi også begyndelse på et serielt interface idet vi implementerede at summen blev printet serielt hver gang den ændrede sig. Det ville derudover også være interessant at implementerer et modtager modul, så man også kunne sende data til automaten. Dette kunne fx være brugbart i en rigtig automat, da den dermed kunne udveksle data med en tekniker. Derudover ville man også kunne bruge dette til at koble FPGA'en til en anden FPGA eller fx en microcontroller.

Seriel-interfacet giver derfor mange muligheder for at lave yderligere udvidelser der kunne have nyttige funktioner i en rigtig colaautomat.

I dette projekt har vi fokuseret meget på at udvide designet med yderligere funktionalitet. Der er flere steder hvor vi kunne have lavet nogle ting på en måske simplere måde. Blandt andet kunne man have lavet både scroll funktionaliteten i displayet og serielkommunikationen med shiftregistre. Vi har dog lagt mere vægt på at udvide funktionaliteten end at finde den absolut billigste løsning.

Konklusion

I denne opgave skulle vi designe og implementer en sodavandsautomat på et FPGA board.

Vi designede en datapath og FSM for sodavandsautomaten der opfyldte kravene om funktionalitet givet i projektspecifikationerne til fulde. Derudover implementerede og dokumenterede vi også en række udvidelser til sodavandsautomaten. Vi lavede et scrollende display der kunne vise teksten COLA eller PEPSI til brugeren. Derudover lavede vi også et serielt interface der tillod at sende data fra FPGA'en til fx en computer. Vi fik dermed givet vores design nogle meget spændende muligheder for at kommunikere med andre enheder. Vi simulerede, ved hjælp af en test-bench skrevet i VHDL, CPU'en og seriel-interfacet i modelsim

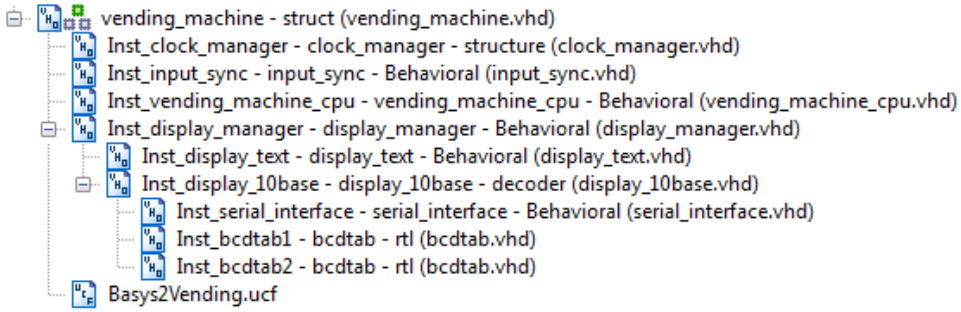
med et tilfredsstillende resultat. Til slut fik vi programmeret vores kredsløb til et FPGA-board og testet det grundigt. Projektet har derfor været succesfuldt.

Kilder

1. Stephen Brown, Zvonko Vranesic. (2009). Digital Logic with VHDL Design. Third Edition. International Edition 2009. McGraw-Hill.
2. Martin Schoeberl, Kursus 02139 Digitalelektronik 2, Forelæsning 12: Interface and Interface Protocols, 2013.
3. Wikipedia: Universal asynchronous receiver/transmitter. Internetadresse: http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter#Character_framing - Besøgt d. 21.5.2013.
4. FTDI: FT230X. Internetadresse: http://www.ftdichip.com/Products/ICs/FT230X.html - Besøgt d. 21.5.2013.
5. Jim.sh: MicroFTX. Internetadresse: http://jim.sh/ftx/ - Besøgt d. 21.5.2013.
6. ASCII Table. Internetadresse: http://www.asciitable.com/ - Besøgt d. 21.5.2013.
7. Digilent: Bays2. Internetadresse: http://www.digilentinc.com/Products/Detail.cfm?Prod=BASYS2 - Besøgt d. 21.5.2013.

Appendix A – VHDL code

Strukturen af VHDL-koden i xilinx er vist på figuren nedenfor. Den fulde kildekode kan også ses på følgende side: https://github.com/Lauszus/Basys2/tree/master/VendingMachine.



vending_machine.vhd

```
----------------------------------------------------------------------------
-- Top component of the vending machine for the course:
-- 02139 Digital electronics 2 at the Technical University of Denmark
-- This component declares and instantiates all the components of the
-- vending machine.
--
-- Created by Mads Bornebusch and Kristian Lauszus
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity vending_machine is
	port (
		clk_50		: in  std_logic; -- 50MHz clock
		clk_man		: in  std_logic; -- Button used for manual clock
		sel_man		: in  std_logic; -- Switch used to select between manual and
onboard clock
		reset		: in  std_logic; -- Switch used to reset
		coin2		: in  std_logic; -- Button used to indicate that a 2 coins has been
inserted
		coin5		: in  std_logic; -- Button used to indicate that a 5 coins has been
inserted
		buy		: in  std_logic; -- Button used to buy a can
		price		: in  std_logic_vector(5 downto 0); -- Switches used to set the
price of the can
		release_can	: out std_logic; -- Led used to indicate that a can is now released
		alarm		: out std_logic; -- Alarm used to indicate that there was not
enough money in order to buy the can
		seven_segment	: out std_logic_vector(7 downto 0); -- Cathodes for the segments
		digit_select	: out std_logic_vector(3 downto 0); -- Anodes for the segments
		tx		: out  std_logic -- TX for the UART
	);
end vending_machine;

architecture struct of vending_machine is
	signal clk		: std_logic; -- 763 Hz clock output from clock manager
	signal sync_reset	: std_logic; -- Reset switch after input synchronizer
	signal sync_coin2	: std_logic; -- 2 coin button after input synchronizer
	signal sync_coin5	: std_logic; -- 5 coin button after input synchronizer
	signal sync_buy		: std_logic; -- Buy button after input synchronizer
	signal sync_price : std_logic_vector(5 downto 0); -- Price switches after input synchro-
nizer
	signal sum		: std_logic_vector(6 downto 0); -- Coin sum calculated in vending
machine cpu

	signal release_can_var, alarm_var : std_logic; -- Internal release can and alarm signals
	signal new_value : std_logic; -- Used to indicate when a new value has been set, either by
buying a can or by inserting a coin

----------------------------------------------------------------------------
-- Clock divider component declaration
```

```
-- Used to generate the clock
--------------------------------------------------------------------------------

	component clock_manager
	port(
		clk_50   :  in  std_logic;
		clk_man  :  in  std_logic;
		sel_man  :  in  std_logic;
		clk      :  out std_logic
		);
	end component;

--------------------------------------------------------------------------------
-- Input synchronizer
-- Synchronize external inputs with the clock in order to avoid metastability
--------------------------------------------------------------------------------

	COMPONENT input_sync
	PORT(
		clk : IN std_logic;
		reset_in : IN std_logic;
		coin2_in : IN std_logic;
		coin5_in : IN std_logic;
		buy_in : IN std_logic;
		price_in : IN std_logic_vector(5 downto 0);
		reset : OUT std_logic;
		coin2 : OUT std_logic;
		coin5 : OUT std_logic;
		buy : OUT std_logic;
		price : OUT std_logic_vector(5 downto 0)
		);
	END COMPONENT;

--------------------------------------------------------------------------------
-- Vending machine cpu
--------------------------------------------------------------------------------

	COMPONENT vending_machine_cpu
	PORT(
		clk : in std_logic;
		reset : IN std_logic;
		coin2 : IN std_logic;
		coin5 : IN std_logic;
		buy : IN std_logic;
		price : IN std_logic_vector(5 downto 0);
		sum : OUT std_logic_vector(6 downto 0);
		release_can : OUT std_logic;
		alarm : OUT std_logic;
		new_value : OUT std_logic
		);
	END COMPONENT;

--------------------------------------------------------------------------------
-- Display manager
-- Controls the display
-- Writes the current price and coin sum
-- Will scroll COLA or PEPSI and a new can is bought or Err if alarm is high
--------------------------------------------------------------------------------

	COMPONENT display_manager
	PORT(
		clk_50 : IN std_logic;
		clk : IN std_logic;
		reset : IN std_logic;
		price : IN std_logic_vector(5 downto 0);
		coin_sum : IN std_logic_vector(6 downto 0);
		buy : IN std_logic;
		release_can : IN std_logic;
		alarm : IN std_logic;
		tx : OUT std_logic;
		seven_segment : OUT std_logic_vector(7 downto 0);
		digit_select : OUT std_logic_vector(3 downto 0);
		new_value : IN std_logic
		);
	END COMPONENT;
```

```
--------------------------------------------------------------------------------

begin
        -- The clock manager instance
        Inst_clock_manager : clock_manager port map (
                clk_50  => clk_50,
                clk_man => clk_man,
                sel_man => sel_man,
                clk => clk
        );

        -- Input synchronizer instance
        Inst_input_sync: input_sync PORT MAP(
                clk => clk,
                reset_in => reset,
                coin2_in => coin2,
                coin5_in => coin5,
                buy_in => buy,
                price_in => price,
                reset => sync_reset,
                coin2 => sync_coin2,
                coin5 => sync_coin5,
                buy => sync_buy,
                price => sync_price
        );

        -- Vending machine cpu instance
        Inst_vending_machine_cpu: vending_machine_cpu PORT MAP(
                clk => clk,
                reset => sync_reset,
                coin2 => sync_coin2,
                coin5 => sync_coin5,
                buy => sync_buy,
                price => sync_price,
                sum => sum,
                release_can => release_can_var,
                alarm => alarm_var,
                new_value => new_value
        );

        -- Display manager instance
        Inst_display_manager: display_manager PORT MAP(
                clk_50 => clk_50,
                clk => clk,
                reset => sync_reset,
                price => sync_price,
                buy => sync_buy,
                seven_segment => seven_segment,
                digit_select => digit_select,
                coin_sum => sum,
                release_can => release_can_var,
                alarm => alarm_var,
                new_value => new_value,
                tx => tx
        );

        alarm <= alarm_var;
        release_can <= release_can_var;

--------------------------------------------------------------------------------
end struct;
```

clock_manager.vhd

```
--------------------------------------------------------------------------------
-- This component divides the 50 MHz clock signal down to a 762 Hz signal
-- In addition the user can select between a manual clock
-- and the 763 Hz clock.
--------------------------------------------------------------------------------

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

```
entity clock_manager is
	port(
		clk_50	: in  std_logic; -- 50Mhz clock signal from board
		clk_man	: in  std_logic; -- Manual clock signal
		sel_man	: in  std_logic; -- Select signal between 763 Hz clock and manual clock
		clk	: out std_logic -- Output signal from clock, 763 Hz
	);
end clock_manager;

architecture structure of clock_manager is
signal count_present, count_next : unsigned(15 downto 0):=(others => '0');
attribute clock_signal : string;
attribute clock_signal of clk : signal is "yes";

begin
	count_next <= count_present + 1;

	process(sel_man, clk_man, count_present)
	begin
		if sel_man = '1' then
			clk <= clk_man;
		else
			clk <= count_present(15);
		end if;
	end process;

	process(clk_50, count_present)
	begin
		if rising_edge(clk_50) then
			count_present <= count_next;
		end if;
	end process;

end structure;
```

inut_sync.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity input_sync is
	port(
	clk : in std_logic;
	reset_in : in std_logic;
	coin2_in : in std_logic;
	coin5_in : in std_logic;
	buy_in : in std_logic;
	price_in : in std_logic_vector(5 downto 0);

	reset : out std_logic;
	coin2 : out std_logic;
	coin5 : out std_logic;
	buy : out std_logic;
	price : out std_logic_vector(5 downto 0));
end input_sync;

architecture Behavioral of input_sync is

begin
	process(clk)
	begin
		if rising_edge(clk) then
			reset <= reset_in;
			coin2 <= coin2_in;
			coin5 <= coin5_in;
			buy <= buy_in;
			price <= price_in;
		end if;
	end process;
end Behavioral;
```

vending_machine_cpu.vhd

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.numeric_std.all;

entity vending_machine_cpu is
	port(
		clk : in std_logic;
		reset : in std_logic;
		coin2 : in std_logic;
		coin5 : in std_logic;
		buy : in std_logic;
		price : in std_logic_vector(5 downto 0);
		sum : out std_logic_vector(6 downto 0);
		release_can : out std_logic;
		alarm : out std_logic;
		new_value : out std_logic
	);
end vending_machine_cpu;

architecture Behavioral of vending_machine_cpu is
	type state_type is (default_state, coin2_state, coin5_state, wait_coin_state, re-
lease_can_state, calc_state, buy_state, alarm_state);
	signal current_state, next_state : state_type;

	signal add_mux : std_logic_vector(1 downto 0);
	signal sum_enable, output_enable, negative : std_logic;
	signal price_val : unsigned(5 downto 0);
	signal sum_val, add_val : unsigned(6 downto 0);
	signal sum_val_temp, sum_val_new : signed(7 downto 0);

begin
	price_val <= unsigned(price);
	sum <= std_logic_vector(sum_val);

	-- Next state logic for the FSM
	process(current_state,coin2,coin5,buy,sum_val_new)
	begin
		next_state <= current_state;

		case current_state is
			when default_state =>
				if coin2 = '1' then
					next_state <= coin2_state;
				elsif coin5 = '1' then
					next_state <= coin5_state;
				elsif buy = '1' then
					next_state <= buy_state;
				end if;
			when coin2_state => next_state <= wait_coin_state;
			when coin5_state => next_state <= wait_coin_state;
			when buy_state => next_state <= calc_state;
			when calc_state =>
				if sum_val_new < 0 then -- Check if the result is negative
					next_state <= alarm_state; -- If it is negative then
it means that there is not enough money in order to buy the can
				else
					next_state <= release_can_state;
				end if;
			when wait_coin_state =>
				if coin2 = '0' and coin5 = '0' then -- Wait until button is
released
					next_state <= default_state;
				end if;
			when alarm_state =>
				if buy = '0' then -- Wait until button is released
					next_state <= default_state;
				end if;
			when release_can_state =>
				if buy = '0' then -- Wait until button is released
					next_state <= default_state;
				end if;
			when others => next_state <= default_state;
		end case;
	end process;

	-- State logic for the FSM
	process(current_state,sum_val,price_val)
	begin
```

```
                add_mux <= "00";
                alarm <= '0';
                release_can <= '0';
                output_enable <= '0';
                sum_enable <= '0';
                negative <= '0';

                case current_state is
                        when coin2_state =>
                                add_mux <= "01";
                                output_enable <= '1';
                        when coin5_state =>
                                add_mux <= "10";
                                output_enable <= '1';
                        when wait_coin_state => sum_enable <= '1';

                        when buy_state =>
                                add_mux <= "11";
                                output_enable <= '1';
                                negative <= '1';
                        when alarm_state => alarm <= '1';
                        when release_can_state =>
                                sum_enable <= '1';
                                release_can <= '1';

                        when others =>
                end case;
        end process;

        -- We use a mux in order to select which value to add or subtract
        process(add_mux,price_val)
        begin
                if add_mux = "01" then
                        add_val <= "0000010"; -- 2
                elsif add_mux = "10" then
                        add_val <= "0000101"; -- 5
                elsif add_mux = "11" then
                        add_val <= '0' & price_val;
                else
                        add_val <= "0000000";
                end if;
        end process;

        -- This adder will take the value selected by the mux and add or subtract in from the cur-
rent coin sum
        process(add_val,sum_val,negative)
        begin
                if negative = '1' then
                        sum_val_temp <= signed('0' & sum_val) - signed('0' & add_val);
                else
                        if sum_val + add_val > 99 then
                                sum_val_temp <= "01100011"; -- 99
                        else
                                sum_val_temp <= signed('0' & sum_val) + signed('0' & add_val);
                        end if;
                end if;
        end process;

        process(clk, reset)
        begin
                if reset = '1' then
                        current_state <= default_state;
                        sum_val <= (others=>'0');
                elsif rising_edge(clk) then
                        current_state <= next_state;
                        if output_enable = '1' then
                                sum_val_new <= sum_val_temp;
                        end if;

                        -- If this is high we will update the coin sum after we have made sure
it is valid
                        if sum_enable = '1' then
                                sum_val <= unsigned(sum_val_new(6 downto 0));
                                new_value <= '1'; -- Bit for the serial interface - used to
indicate that a new value has been set
                        else
```

```
                                        new_value <= '0';
                                end if;
                        end if;
                end process;

end Behavioral;
```

display_manager.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity display_manager is
        port (
                clk_50 : in std_logic; -- 50MHz clock
                clk : in std_logic;
                reset : in std_logic;
                tx : out std_logic;
                price : in std_logic_vector(5 downto 0);
                coin_sum: in std_logic_vector(6 downto 0);
                seven_segment : out std_logic_vector (7 downto 0);
                digit_select : out std_logic_vector(3 downto 0);
                buy : in std_logic;
                release_can : in std_logic;
                alarm : in std_logic;
                new_value : in std_logic
        );
end display_manager;

architecture Behavioral of display_manager is
        signal seven_segment_temp, seven_segment_text, seven_segment_10base : std_logic_vector (7
downto 0);
        signal digit_select_temp, digit_select_text, digit_select_10base : std_logic_vector(3
downto 0);

-------------------------------------------------------------------------------
-- Display 10 base
-- Used to display the binary price and coin sum as a 10 base number
-------------------------------------------------------------------------------

        COMPONENT display_10base
        PORT(
                clk_50 : IN std_logic;
                clk : IN std_logic;
                reset : IN std_logic;
                seven_segment : OUT std_logic_vector(7 downto 0);
                digit_select : OUT std_logic_vector(3 downto 0);
                price : IN std_logic_vector(5 downto 0);
                coin_sum : IN std_logic_vector(6 downto 0);
                new_value : IN std_logic;
                tx : OUT std_logic
                );
        END COMPONENT;

-------------------------------------------------------------------------------
-- Display text
-- Used to scroll COLA or PEPSI when a can is bought
-- A cola or pepsi is decided by the flavor bit
-- Will show Err when alarm is high
-------------------------------------------------------------------------------

        COMPONENT display_text
        PORT(
                clk : IN std_logic;
                seven_segment : OUT std_logic_vector(7 downto 0);
                digit_select : OUT std_logic_vector(3 downto 0);
                reset : IN std_logic;
                release_can : IN std_logic;
                alarm : IN std_logic;
                flavor : IN std_logic
                );
        END COMPONENT;

-------------------------------------------------------------------------------
```

```
begin
	seven_segment <= seven_segment_temp;
	digit_select <= digit_select_temp;

	-- The buy button selects which instance that should be controlling the seven segment dis-
play
	process(buy,seven_segment_text,digit_select_text,seven_segment_10base,digit_select_10base)
	begin
		if buy = '1' then
			seven_segment_temp <= seven_segment_text;
			digit_select_temp <= digit_select_text;
		else
			seven_segment_temp <= seven_segment_10base;
			digit_select_temp <= digit_select_10base;
		end if;
	end process;

	-- Display text instance
	Inst_display_text: display_text PORT MAP(
		clk => clk,
		reset => reset,
		seven_segment => seven_segment_text,
		digit_select => digit_select_text,
		release_can => release_can,
		alarm => alarm,
		flavor => price(0) -- Determine flavor based on if it's an equal or unequal num-
ber
	);

	-- Display 10 base instance
	Inst_display_10base: display_10base PORT MAP(
		clk_50 => clk_50,
		clk => clk,
		reset => reset,
		seven_segment => seven_segment_10base,
		digit_select => digit_select_10base,
		price => price,
		coin_sum => coin_sum,
		new_value => new_value,
		tx => tx
	);

end Behavioral;
```

display_text.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity display_text is
	port(
		clk : in std_logic;
		reset : in std_logic;
		seven_segment : out std_logic_vector (7 downto 0);
		digit_select : out std_logic_vector(3 downto 0);
		release_can : in std_logic;
		alarm : in std_logic;
		flavor : in std_logic
	);
end display_text;

architecture Behavioral of display_text is
	-- State signals
	type state_type is (state0, state1, state2, state3, state4, state5, state6, state7,
state8, error_state);
	signal current_state, next_state : state_type;

	-- Used to scroll the text slowly across the segments
	signal cnt, cnt_reg : std_logic_vector(7 downto 0);
	signal display_enable : std_logic;

	signal segment_selector, segment_selector_next : std_logic_vector(1 downto 0);
	signal digit0, digit1, digit2, digit3 : std_logic_vector(7 downto 0);
```

```
	-- These constants are used to write letters to the seven segment display
	constant OFF : std_logic_vector(7 downto 0) := "11111111";
	constant C : std_logic_vector(7 downto 0) := "11000110";
	constant O : std_logic_vector(7 downto 0) := "11000000";
	constant L : std_logic_vector(7 downto 0) := "11000111";
	constant A : std_logic_vector(7 downto 0) := "10001000";
	constant E : std_logic_vector(7 downto 0) := "10000110";
	constant r : std_logic_vector(7 downto 0) := "10101111";
	constant P : std_logic_vector(7 downto 0) := "10001100";
	constant S : std_logic_vector(7 downto 0) := "10010010";
	constant I : std_logic_vector(7 downto 0) := "11001111";

begin
	segment_selector_next <= segment_selector + 1;

	process(clk,reset)
	begin
		if reset = '1' then
			cnt_reg <= (others => '0');
			current_state <= state0;
		elsif rising_edge(clk) then
			cnt_reg <= cnt;
			segment_selector <= segment_selector_next;
			if display_enable = '1' then
				current_state <= next_state;
			end if;
		end if;
	end process;

	-- This will control the multiplexing of the display
	process(segment_selector,digit0,digit1,digit2,digit3)
	begin
		if segment_selector = "00" then
			digit_select <= "1110";
			seven_segment <= digit0;
		elsif segment_selector = "01" then
			digit_select <= "1101";
			seven_segment <= digit1;
		elsif segment_selector = "10" then
			digit_select <= "1011";
			seven_segment <= digit2;
		else
			digit_select <= "0111";
			seven_segment <= digit3;
		end if;
	end process;

	-- This will generate an enable signal with a frequency of 3.81 Hz
	-- This will ensure that the text is scrolled nice and slowly
	process(cnt_reg)
	begin
		display_enable <= '0';
		if cnt_reg = 200 then
			cnt <= (others => '0');
			display_enable <= '1';
		else
			cnt <= cnt_reg + 1;
		end if;
	end process;

	-- Next state logic for the FSM
	process(current_state,alarm,release_can,flavor)
	begin
		next_state <= current_state;

		if alarm = '1' then
			next_state <= error_state; -- If alarm is high we will print Err on the
display
		elsif release_can = '0' then
			next_state <= state0; -- Start the scrolling sequence
		else
			case current_state is
				when state0 => next_state <= state1;
				when state1 => next_state <= state2;
				when state2 => next_state <= state3;
				when state3 => next_state <= state4;
```

```
                                        when state4 => next_state <= state5;
                                        when state5 => next_state <= state6;
                                        when state6 => next_state <= state7;
                                        when state7 =>
                                                if flavor = '0' then
                                                        next_state <= state0;
                                                else
                                                        next_state <= state8; -- Since COLA and
PEPSI are not equal length we need one more state to write PEPSI
                                                end if;
                                        when state8 => next_state <= state0;
                                        when error_state => next_state <= state0;
                                        when others => next_state <= state0;
                                end case;
                        end if;
                end process;

                -- State logic for the FSM
                -- This will set the different digits depending on the state
                process(current_state,flavor)
                begin
                        if current_state = state0 then
                                digit0 <= OFF;
                                digit1 <= OFF;
                                digit2 <= OFF;
                                digit3 <= OFF;
                        elsif current_state = state1 then
                                if flavor = '0' then
                                        digit0 <= C;
                                else
                                        digit0 <= P;
                                end if;
                                digit1 <= OFF;
                                digit2 <= OFF;
                                digit3 <= OFF;
                        elsif current_state = state2 then
                                if flavor = '0' then
                                        digit0 <= O;
                                        digit1 <= C;
                                else
                                        digit0 <= E;
                                        digit1 <= P;
                                end if;
                                digit2 <= OFF;
                                digit3 <= OFF;
                        elsif current_state = state3 then
                                if flavor = '0' then
                                        digit0 <= L;
                                        digit1 <= O;
                                        digit2 <= C;
                                else
                                        digit0 <= P;
                                        digit1 <= E;
                                        digit2 <= P;
                                end if;
                                digit3 <= OFF;
                        elsif current_state = state4 then
                                if flavor = '0' then
                                        digit0 <= A;
                                        digit1 <= L;
                                        digit2 <= O;
                                        digit3 <= C;
                                else
                                        digit0 <= S;
                                        digit1 <= P;
                                        digit2 <= E;
                                        digit3 <= P;
                                end if;
                        elsif current_state = state5 then
                                if flavor = '0' then
                                        digit0 <= OFF;
                                        digit1 <= A;
                                        digit2 <= L;
                                        digit3 <= O;
                                else
                                        digit0 <= I;
```

```
                                digit1 <= S;
                                digit2 <= P;
                                digit3 <= E;
                        end if;
                elsif current_state = state6 then
                        digit0 <= OFF;
                        if flavor = '0' then
                                digit1 <= OFF;
                                digit2 <= A;
                                digit3 <= L;
                        else
                                digit1 <= I;
                                digit2 <= S;
                                digit3 <= P;
                        end if;
                elsif current_state = state7 then
                        digit0 <= OFF;
                        digit1 <= OFF;
                        if flavor = '0' then
                                digit2 <= OFF;
                                digit3 <= A;
                        else
                                digit2 <= I;
                                digit3 <= S;
                        end if;
                elsif current_state = state8 then
                        digit0 <= OFF;
                        digit1 <= OFF;
                        digit2 <= OFF;
                        digit3 <= I;
                elsif current_state = error_state then
                        digit0 <= OFF;
                        digit1 <= r;
                        digit2 <= r;
                        digit3 <= E;
                else
                        digit0 <= OFF;
                        digit1 <= OFF;
                        digit2 <= OFF;
                        digit3 <= OFF;
                end if;
        end process;

end Behavioral;
```

display_10base.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity display_10base is
        port (
                clk_50 : in std_logic; -- 50MHz clock
                reset : in std_logic;
                tx : out std_logic;
                price : in std_logic_vector(5 downto 0);
                coin_sum: in std_logic_vector(6 downto 0);
                seven_segment : out std_logic_vector (7 downto 0);
                digit_select : out std_logic_vector(3 downto 0);
                clk : in std_logic;
                new_value : in std_logic
        );
end display_10base;

architecture decoder of display_10base is
        signal selector, selector_next : std_logic_vector(1 downto 0);
        signal digit_out : std_logic_vector(3 downto 0);
        signal digit0, digit2, digit3 : std_logic_vector(3 downto 0);
        signal digit1 : std_logic_vector(2 downto 0);
        signal price_temp : std_logic_vector(6 downto 0);
        signal q1, q2 : std_logic_vector(7 downto 0);

        signal new_enable : std_logic;
```

```
----------------------------------------------------------------------------
-- Binary to decimal converter
-- Used to convert price and coin sum into 10 base numbers
----------------------------------------------------------------------------

	COMPONENT bcdtab
	PORT(
		address : IN std_logic_vector(6 downto 0);
		q : OUT std_logic_vector(7 downto 0)
		);
	END COMPONENT;

----------------------------------------------------------------------------
-- Serial interface (UART)
-- Used to write the value of coin sum when a new value is updated
-- Only transmit is implemented
----------------------------------------------------------------------------

	COMPONENT serial_interface
	PORT(
		clk_50 : IN std_logic;
		reset : IN std_logic;
		digit0 : IN std_logic_vector(3 downto 0);
		digit1 : IN std_logic_vector(3 downto 0);
		new_value : IN std_logic;
		tx : OUT std_logic
		);
	END COMPONENT;

----------------------------------------------------------------------------

begin
	-- Serial interface interface
	Inst_serial_interface: serial_interface PORT MAP(
		clk_50 => clk_50,
		reset => reset,
		digit0 => digit3,
		digit1 => digit2,
		tx => tx,
		new_value => new_enable -- When this goes high, the serial interface will send
the coin sum
	);

	-- BCD instances
	Inst_bcdtab1: bcdtab PORT MAP(
		address => price_temp,
		q => q1
	);
	price_temp <= '0' & price;

	Inst_bcdtab2: bcdtab PORT MAP(
		address => coin_sum,
		q => q2
	);

	-- Set the segments to the desired value
	process(digit_out)
	begin
		case digit_out is
			when "0000" => seven_segment <= "11000000";
			when "0001" => seven_segment <= "11111001";
			when "0010" => seven_segment <= "10100100";
			when "0011" => seven_segment <= "10110000";
			when "0100" => seven_segment <= "10011001";
			when "0101" => seven_segment <= "10010010";
			when "0110" => seven_segment <= "10000010";
			when "0111" => seven_segment <= "11111000";
			when "1000" => seven_segment <= "10000000";
			when "1001" => seven_segment <= "10010000";
			when others => seven_segment <= (others=>'1');
		end case;
	end process;

	-- Used for the display multiplexing
	process(selector,digit0,digit1,digit2,digit3)
	begin
```

```
                if selector = "00" then
                        digit_select <= "1110";
                        digit_out <= digit0;
                elsif selector = "01" then
                        digit_select <= "1101";
                        digit_out <= '0' & digit1;
                elsif selector = "10" then
                        digit_select <= "1011";
                        digit_out <= digit2;
                else
                        digit_select <= "0111";
                        digit_out <= digit3;
                end if;
        end process;

        -- This will update the segment selector and update the different
        -- digits based on price and coin sum
        process(clk)
        begin
                if rising_edge(clk) then
                        selector <= selector_next;
                        digit0 <= q1(3 downto 0);
                        digit1 <= q1(6 downto 4);
                        digit2 <= q2(3 downto 0);
                        digit3 <= q2(7 downto 4);

                        -- Used to indicate when the computation is done and it is
                        -- okay for the serial interface to send the current coin sum
                        if new_value = '1' then
                                new_enable <= '1';
                        else
                                new_enable <= '0';
                        end if;
                end if;
        end process;

        selector_next <= selector + 1;

end decoder;
```

serial_interface.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;

entity serial_interface is
        port (
                clk_50 : in std_logic; -- 50MHz clock
                reset : in std_logic;
                digit0 : in std_logic_vector(3 downto 0);
                digit1 : in std_logic_vector(3 downto 0);
                tx : out std_logic;
                new_value : in std_logic
        );
end serial_interface;

architecture Behavioral of serial_interface is
        type state_type is (state_idle, state_start, state0, state1, state2, state3, state4,
state5, state6, state7, state_stop, state_digit_check, state_digit_increment,
state_digit_increment_stop);
        signal current_state, next_state : state_type;

        signal cnt, cnt_reg : std_logic_vector(9 downto 0);
        signal serial_enable : std_logic;

        signal digit, digit_next : std_logic_vector(1 downto 0);
        signal digit_enable : std_logic;
        constant CR : std_logic_vector(3 downto 0) := "1101"; -- Carriage return
        constant LF : std_logic_vector(3 downto 0) := "1010"; -- Line feed

begin
        digit_next <= digit + 1;

        process(clk_50,reset)
```

```
	begin
		if reset = '1' then
			cnt_reg <= (others => '0');
			digit <= (others => '0');
			current_state <= state_idle;
		elsif rising_edge(clk_50) then
			cnt_reg <= cnt;
			if serial_enable = '1' then
				current_state <= next_state;
				if digit_enable = '1' then
					digit <= digit_next;
				end if;
			end if;
		end if;
	end process;

	-- Used to generate the serial baudrate of 115200
	process(cnt_reg)
	begin
		serial_enable <= '0';
		if cnt_reg = 434 then -- Set the baudrate to 115200 (50MHz/115200=434)
			cnt <= (others => '0');
			serial_enable <= '1';
		else
			cnt <= cnt_reg + 1;
		end if;
	end process;

	-- Next state logic for the FSM
	process(current_state,new_value,digit)
	begin
		next_state <= current_state;

		case current_state is
			when state_idle =>
				if new_value = '1' then
					next_state <= state_start;
				end if;
			when state_start => next_state <= state0;
			when state0 => next_state <= state1;
			when state1 => next_state <= state2;
			when state2 => next_state <= state3;
			when state3 => next_state <= state4;
			when state4 => next_state <= state5;
			when state5 => next_state <= state6;
			when state6 => next_state <= state7;
			when state7 => next_state <= state_stop;
			when state_stop => next_state <= state_digit_check;
			when state_digit_check =>
				if digit = "11" then -- Check if we have send all characters
					if new_value = '0' then -- Only send the data once
						next_state <= state_digit_increment_stop;
					end if;
				else
					next_state <= state_digit_increment;
				end if;
			when state_digit_increment => next_state <= state_start;
			when state_digit_increment_stop => next_state <= state_idle;
			when others => next_state <= state_idle;
		end case;
	end process;

	-- State logic for the FSM
	process(current_state,digit,digit0,digit1) -- Send the output as 8N1
	begin
		digit_enable <= '0';
		tx <= '1'; -- The start bit sets TX low after that we simply write the byte and
set TX high again

		if current_state = state_start then
			tx <= '0'; -- Set TX low to indicate that we want to send a new byte
		elsif current_state = state0 then
			if digit = "00" then
				tx <= digit0(0);
			elsif digit = "01" then
				tx <= digit1(0);
```

```
                        elsif digit = "10" then
                                tx <= CR(0);
                        else
                                tx <= LF(0);
                        end if;
                elsif current_state = state1 then
                        if digit = "00" then
                                tx <= digit0(1);
                        elsif digit = "01" then
                                tx <= digit1(1);
                        elsif digit = "10" then
                                tx <= CR(1);
                        else
                                tx <= LF(1);
                        end if;
                elsif current_state = state2 then
                        if digit = "00" then
                                tx <= digit0(2);
                        elsif digit = "01" then
                                tx <= digit1(2);
                        elsif digit = "10" then
                                tx <= CR(2);
                        else
                                tx <= LF(2);
                        end if;
                elsif current_state = state3 then
                        if digit = "00" then
                                tx <= digit0(3);
                        elsif digit = "01" then
                                tx <= digit1(3);
                        elsif digit = "10" then
                                tx <= CR(3);
                        else
                                tx <= LF(3);
                        end if;
                elsif current_state = state4 or current_state = state5 then
                        if digit = "00" or digit = "01" then
                                tx <= '1'; -- Convert to ASCII
                        else
                                tx <= '0';
                        end if;
                elsif current_state = state6 or current_state = state7 then
                        tx <= '0';
                elsif current_state = state_stop then
                        tx <= '1'; -- Set it high again after the transfer
                elsif current_state = state_digit_increment or current_state =
state_digit_increment_stop then
                        digit_enable <= '1'; -- This will increment "digit"
                end if;
        end process;

end Behavioral;
```

bcdtab.vhd

```
--
--      bcdtab.vhd
--
--      generated VHDL table for BCD conversion
--
--              DONT edit this file!
--              generated by GenBcdConv
--

library ieee;
use ieee.std_logic_1164.all;

entity bcdtab is
port (
   address : in std_logic_vector(6 downto 0);
   q : out std_logic_vector(7 downto 0)
);
end bcdtab;

architecture rtl of bcdtab is
```

```
begin

process(address) begin

case address is
    when "0000000" => q <= "00000000";
    when "0000001" => q <= "00000001";
    when "0000010" => q <= "00000010";
    when "0000011" => q <= "00000011";
    when "0000100" => q <= "00000100";
    when "0000101" => q <= "00000101";
    when "0000110" => q <= "00000110";
    when "0000111" => q <= "00000111";
    when "0001000" => q <= "00001000";
    when "0001001" => q <= "00001001";
    when "0001010" => q <= "00010000";
    when "0001011" => q <= "00010001";
    when "0001100" => q <= "00010010";
    when "0001101" => q <= "00010011";
    when "0001110" => q <= "00010100";
    when "0001111" => q <= "00010101";
    when "0010000" => q <= "00010110";
    when "0010001" => q <= "00010111";
    when "0010010" => q <= "00011000";
    when "0010011" => q <= "00011001";
    when "0010100" => q <= "00100000";
    when "0010101" => q <= "00100001";
    when "0010110" => q <= "00100010";
    when "0010111" => q <= "00100011";
    when "0011000" => q <= "00100100";
    when "0011001" => q <= "00100101";
    when "0011010" => q <= "00100110";
    when "0011011" => q <= "00100111";
    when "0011100" => q <= "00101000";
    when "0011101" => q <= "00101001";
    when "0011110" => q <= "00110000";
    when "0011111" => q <= "00110001";
    when "0100000" => q <= "00110010";
    when "0100001" => q <= "00110011";
    when "0100010" => q <= "00110100";
    when "0100011" => q <= "00110101";
    when "0100100" => q <= "00110110";
    when "0100101" => q <= "00110111";
    when "0100110" => q <= "00111000";
    when "0100111" => q <= "00111001";
    when "0101000" => q <= "01000000";
    when "0101001" => q <= "01000001";
    when "0101010" => q <= "01000010";
    when "0101011" => q <= "01000011";
    when "0101100" => q <= "01000100";
    when "0101101" => q <= "01000101";
    when "0101110" => q <= "01000110";
    when "0101111" => q <= "01000111";
    when "0110000" => q <= "01001000";
    when "0110001" => q <= "01001001";
    when "0110010" => q <= "01010000";
    when "0110011" => q <= "01010001";
    when "0110100" => q <= "01010010";
    when "0110101" => q <= "01010011";
    when "0110110" => q <= "01010100";
    when "0110111" => q <= "01010101";
    when "0111000" => q <= "01010110";
    when "0111001" => q <= "01010111";
    when "0111010" => q <= "01011000";
    when "0111011" => q <= "01011001";
    when "0111100" => q <= "01100000";
    when "0111101" => q <= "01100001";
    when "0111110" => q <= "01100010";
    when "0111111" => q <= "01100011";
    when "1000000" => q <= "01100100";
    when "1000001" => q <= "01100101";
    when "1000010" => q <= "01100110";
    when "1000011" => q <= "01100111";
    when "1000100" => q <= "01101000";
    when "1000101" => q <= "01101001";
```

```
    when "1000110" => q <= "01110000";
    when "1000111" => q <= "01110001";
    when "1001000" => q <= "01110010";
    when "1001001" => q <= "01110011";
    when "1001010" => q <= "01110100";
    when "1001011" => q <= "01110101";
    when "1001100" => q <= "01110110";
    when "1001101" => q <= "01110111";
    when "1001110" => q <= "01111000";
    when "1001111" => q <= "01111001";
    when "1010000" => q <= "10000000";
    when "1010001" => q <= "10000001";
    when "1010010" => q <= "10000010";
    when "1010011" => q <= "10000011";
    when "1010100" => q <= "10000100";
    when "1010101" => q <= "10000101";
    when "1010110" => q <= "10000110";
    when "1010111" => q <= "10000111";
    when "1011000" => q <= "10001000";
    when "1011001" => q <= "10001001";
    when "1011010" => q <= "10010000";
    when "1011011" => q <= "10010001";
    when "1011100" => q <= "10010010";
    when "1011101" => q <= "10010011";
    when "1011110" => q <= "10010100";
    when "1011111" => q <= "10010101";
    when "1100000" => q <= "10010110";
    when "1100001" => q <= "10010111";
    when "1100010" => q <= "10011000";
    when "1100011" => q <= "10011001"; -- 99
    when others => q <= "00000000";

end case;
end process;

end rtl;
```

Basys2Vending.ucf

```
# Pin assignments for our Vending Machine

# Clock pin for Basys2 Board
NET "clk_50" LOC = "B8";

# Connected to Basys2 onBoard 7seg display
NET "seven_segment<0>" LOC = "L14"; # Bank = 1, Signal name = CA
NET "seven_segment<1>" LOC = "H12"; # Bank = 1, Signal name = CB
NET "seven_segment<2>" LOC = "N14"; # Bank = 1, Signal name = CC
NET "seven_segment<3>" LOC = "N11"; # Bank = 2, Signal name = CD
NET "seven_segment<4>" LOC = "P12"; # Bank = 2, Signal name = CE
NET "seven_segment<5>" LOC = "L13"; # Bank = 1, Signal name = CF
NET "seven_segment<6>" LOC = "M12"; # Bank = 1, Signal name = CG
NET "seven_segment<7>" LOC = "N13"; # Bank = 1, Signal name = DP

# Anodes
NET "digit_select<3>" LOC = "K14"; # Bank = 1, Signal name = AN3
NET "digit_select<2>" LOC = "M13"; # Bank = 1, Signal name = AN2
NET "digit_select<1>" LOC = "J12"; # Bank = 1, Signal name = AN1
NET "digit_select<0>" LOC = "F12"; # Bank = 1, Signal name = AN0

# Pin assignment for LEDs
NET "alarm" LOC = "G1" ; # Bank = 3, Signal name = LD7
NET "release_can" LOC = "M5" ;  # Bank = 2, Signal name = LD0

# Pin assignment for SWs
NET "sel_man"  LOC = "N3";  # Bank = 2, Signal name = SW7
NET "reset"    LOC = "E2";  # Bank = 3, Signal name = SW6
NET "price<5>" LOC = "F3";  # Bank = 3, Signal name = SW5
NET "price<4>" LOC = "G3";  # Bank = 3, Signal name = SW4
NET "price<3>" LOC = "B4";  # Bank = 3, Signal name = SW3
NET "price<2>" LOC = "K3";  # Bank = 3, Signal name = SW2
NET "price<1>" LOC = "L3";  # Bank = 3, Signal name = SW1
NET "price<0>" LOC = "P11";  # Bank = 2, Signal name = SW0

# Pin assignment for buttons
```

```
NET "buy"      LOC = "A7";  # Bank = 1, Signal name = BTN3
NET "coin5"    LOC = "M4";  # Bank = 0, Signal name = BTN2
NET "coin2"    LOC = "C11"; # Bank = 2, Signal name = BTN1
NET "clk_man" LOC = "G12"; # Bank = 0, Signal name = BTN0

NET "tx" LOC = "D12";
```

Appendix B – Test benches

Vi lavede to test benches. En til at teste CPU'en og en til at teste seriel-kommunikationen. De to testbenches findes nedenfor.

vending_machine_test.vhd

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   23:34:15 05/20/2013
-- Design Name:
-- Module Name:
C:/Users/Lauszus/Documents/GitHub/Basys2/VendingMachine/src/vending_machine_test.vhd
-- Project Name:  vending_machine
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: vending_machine
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY vending_machine_test IS
END vending_machine_test;

ARCHITECTURE behavior OF vending_machine_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT vending_machine
    PORT(
         clk_50 : IN  std_logic;
         clk_man : IN  std_logic;
         sel_man : IN  std_logic;
         reset : IN  std_logic;
         coin2 : IN  std_logic;
         coin5 : IN  std_logic;
         buy : IN  std_logic;
         price : IN  std_logic_vector(5 downto 0);
         release_can : OUT  std_logic;
         alarm : OUT  std_logic;
         seven_segment : OUT  std_logic_vector(7 downto 0);
         digit_select : OUT  std_logic_vector(3 downto 0);
         tx : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal clk_50 : std_logic := '0';
   signal clk_man : std_logic := '0';
   signal sel_man : std_logic := '1';
   signal reset : std_logic := '0';
   signal coin2 : std_logic := '0';
```

```
   signal coin5 : std_logic := '0';
   signal buy : std_logic := '0';
   signal price : std_logic_vector(5 downto 0) := (others => '0');

         --Outputs
   signal release_can : std_logic;
   signal alarm : std_logic;
   signal seven_segment : std_logic_vector(7 downto 0);
   signal digit_select : std_logic_vector(3 downto 0);
   signal tx : std_logic;

   -- Clock period definitions
   constant clk_50_period : time := 10 ns;
   constant clk_man_period : time := 10 ns;

BEGIN

         -- Instantiate the Unit Under Test (UUT)
   uut: vending_machine PORT MAP (
          clk_50 => clk_50,
          clk_man => clk_man,
          sel_man => sel_man,
          reset => reset,
          coin2 => coin2,
          coin5 => coin5,
          buy => buy,
          price => price,
          release_can => release_can,
          alarm => alarm,
          seven_segment => seven_segment,
          digit_select => digit_select,
          tx => tx
        );

   -- Clock process definitions
   clk_50_process :process
   begin
                  clk_50 <= '0';
                  wait for clk_50_period/2;
                  clk_50 <= '1';
                  wait for clk_50_period/2;
   end process;

   clk_man_process :process
   begin
                  clk_man <= '0';
                  wait for clk_man_period/2;
                  clk_man <= '1';
                  wait for clk_man_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- insert stimulus here

      reset <= '1';
      wait for clk_50_period*5;
      reset <= '0';
      wait for clk_50_period*5;

      -- insert stimulus here
      price <= "000111";
      wait for clk_50_period*5;
      buy <= '1';
      wait for clk_50_period*5;
      assert alarm = '0' report "error: alarm should be 1";
      buy <= '0';

      coin2 <= '1';
      wait for clk_50_period*5;
      coin2 <= '0';
      wait for clk_50_period*5;
      coin5 <= '1';
      wait for clk_50_period*5;
```

```
      coin5 <= '0';
      buy <= '1';
      wait for clk_50_period*5;
      assert release_can = '0' report "error: release_can should be 1";

      wait;
   end process;

END;
```

serial_interface_test.vhd

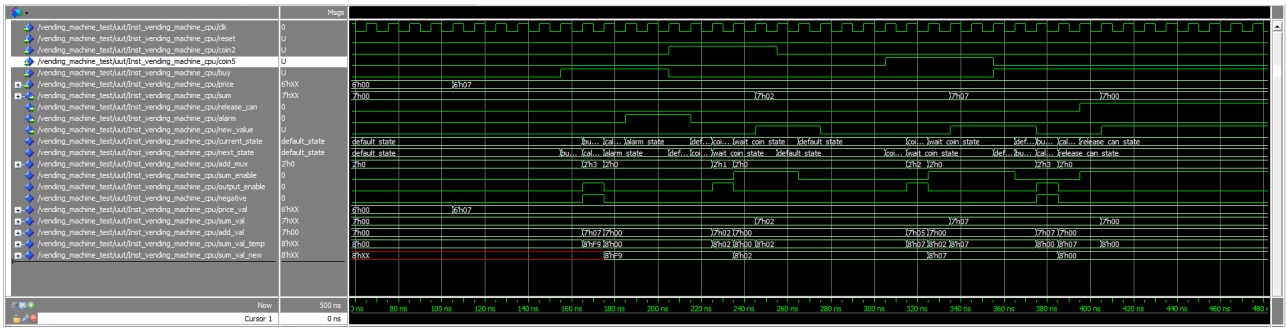
```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   00:30:20 05/21/2013
-- Design Name:
-- Module Name:
C:/Users/Lauszus/Documents/GitHub/Basys2/VendingMachine/src/serial_interface_test.vhd
-- Project Name:  vending_machine
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: serial_interface
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY serial_interface_test IS
END serial_interface_test;

ARCHITECTURE behavior OF serial_interface_test IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT serial_interface
    PORT(
         clk_50 : IN  std_logic;
         reset : IN  std_logic;
         digit0 : IN  std_logic_vector(3 downto 0);
         digit1 : IN  std_logic_vector(3 downto 0);
         tx : OUT  std_logic;
         new_value : IN  std_logic
        );
    END COMPONENT;


   --Inputs
   signal clk_50 : std_logic := '0';
   signal reset : std_logic := '0';
   signal digit0 : std_logic_vector(3 downto 0) := (others => '0');
   signal digit1 : std_logic_vector(3 downto 0) := (others => '0');
   signal new_value : std_logic := '0';

         --Outputs
   signal tx : std_logic;
```

```
   -- Clock period definitions
   constant clk_50_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: serial_interface PORT MAP (
          clk_50 => clk_50,
          reset => reset,
          digit0 => digit0,
          digit1 => digit1,
          tx => tx,
          new_value => new_value
        );

   -- Clock process definitions
   clk_50_process :process
   begin
                   clk_50 <= '0';
                   wait for clk_50_period/2;
                   clk_50 <= '1';
                   wait for clk_50_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      reset <= '1';
      wait for clk_50_period*10;
      reset <= '0';
      wait for clk_50_period*10;

      digit0 <= "0111";
      digit1 <= "0101";
      new_value <= '1';

      wait;
   end process;

END;
```

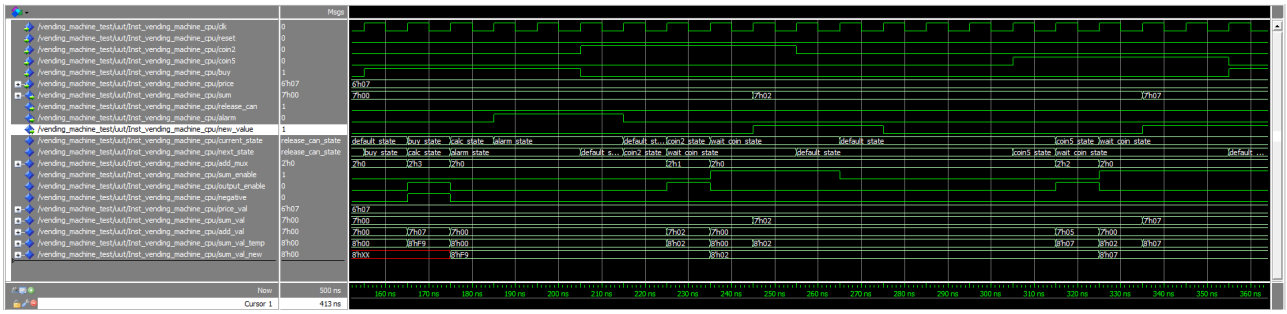
Appendix C – Simulering af CPU'en

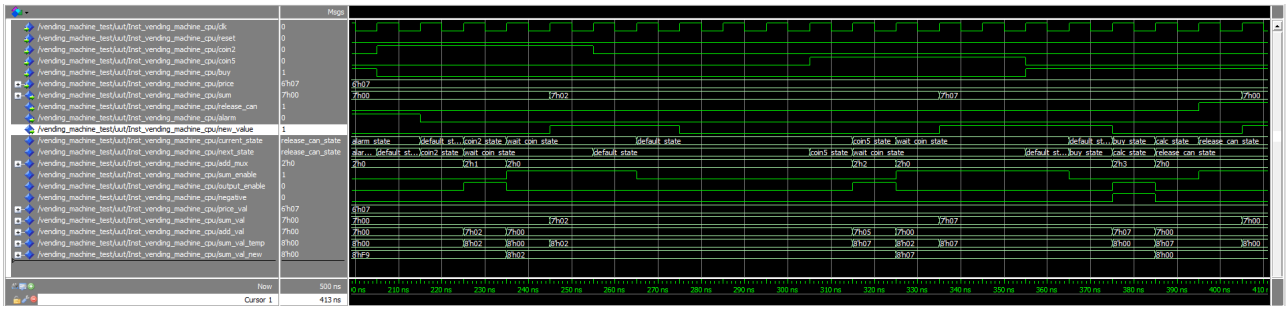
Nedenfor er vist en samling af screenshots fra simuleringen af CPU'en i Modelsim.

Det øverste viser et overblik over hele simuleringen, mens de to andre viser et udsnit af denne simulering. Dette medfører at man kan læse de enkelte states og skulle forhåbentligt gøre det mere overskueligt.



Figur 14 – Simuleringen af CPU'en.



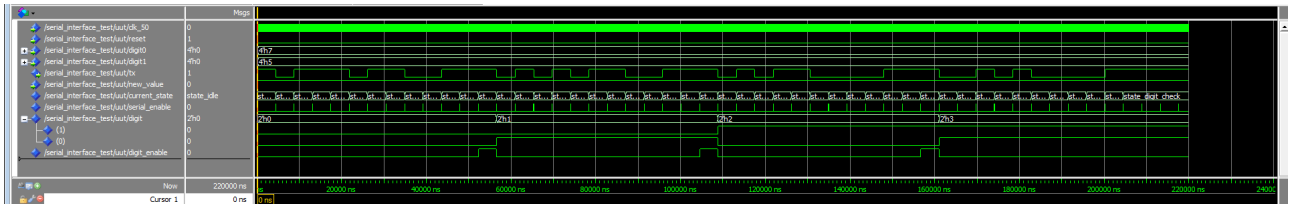


Figur 15 – Closeup af simuleringen af CPU'en.

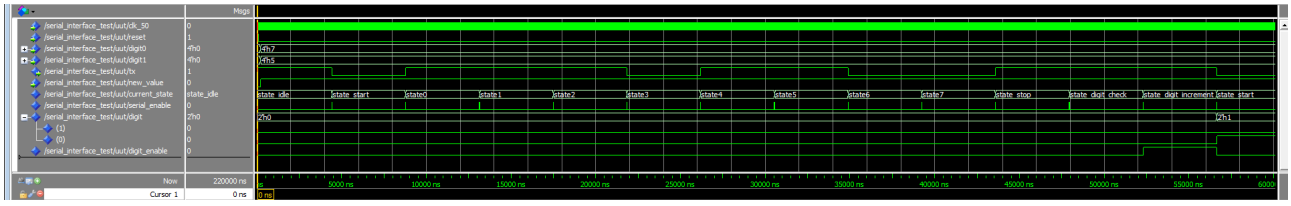
Appendix D – Simulering af serielkommunikation

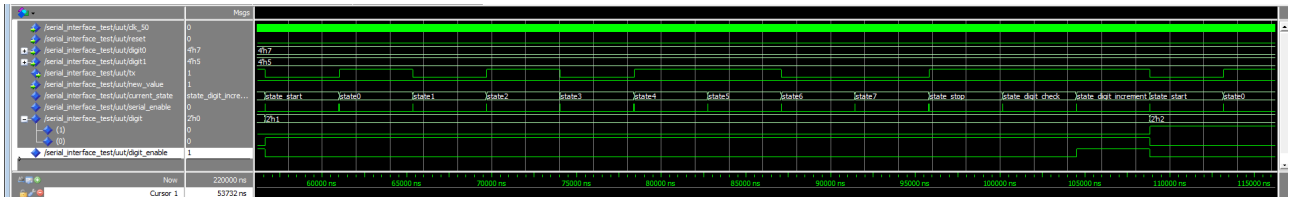
Nedenfor er vist en samling af screenshots fra simuleringen af serielkommunikationen i Modelsim.

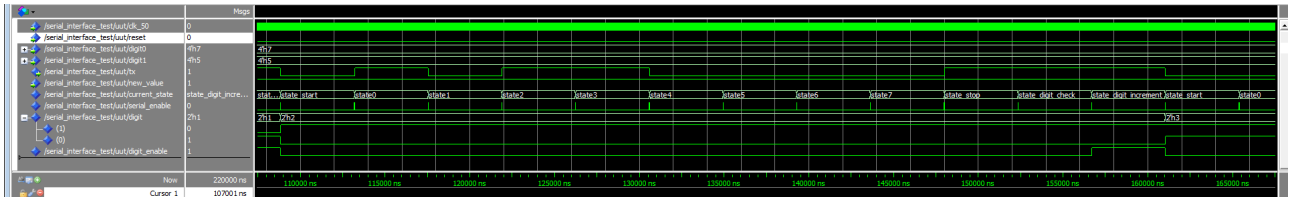
Det øverste viser et overblik over hele simuleringen, mens de fire andre viser et udsnit af denne simulering.

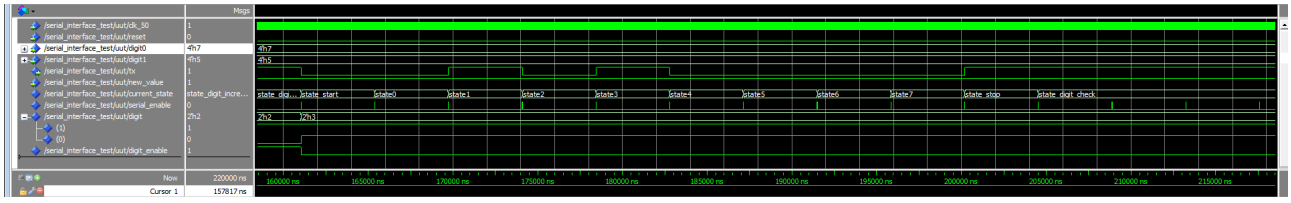


Figur 16 – Simuleringen af serielkommunikationen.









Figur 17 – Closeup af simuleringen af serielkommunikationen.