30010 Programmeringsprojekt: Reflex Ball

Mads Friis Bornebusch (s123627)

Tobias Tuxen (s120213)

Kristian Sloth Lauszus (s123808)

Reflex Ball
Group Number: 15
DTU Space
June 21, 2013

Resumé

I dette programmeringsprojekt har vi skrevet og designet al koden til spillet Reflex Ball. Koden er skrevet i C i compileren Z8Encore! og implementeret på en Zilog 6403 mircocontroller.

Vi har konkluderet at den skrevne kode implementerer det ønskede kredsløb på Mircrocontrolleren, da alle spillets facetter er blevet gennemtestet og giver det ønskede output. Som slutresultat har vi et fuldt funktionelt ReflexBall-spil med mange udvidelser der bl.a. inkluderer brikker (Arkanoid-stil), power-ups, forskellige levels og styring med ret. Der er indtil videre ikke fundet nogle bugs i slutversionen af spillet.

Abstract

In this programming project we have written and designed all the code to the game Reflex Ball. The code has been written in C in the compiler Z8Encore! and has been implemented on a Zilog 6403 Mircrocontroller.

We have concluded that the written code implements the desired circuit on the Microcontroller as as all the different facilities of the game has been thoroughly tested and is in compliance with the expected result. In the end we have a fully functional Reflex Ball Game with many expansions, which amongst others, include bricks (Arkanoid style), power-ups, different levels and steering-wheel game controller. So far no bugs has been found in the final version of the game.

`figs/forside.png`

Figure 1: Screenshot from our implementation of ReflexBall

Fordord

Denne rapport er skrevet som en del af eksaminationen i DTU kursus 30010 Programmeringsprojekt. Alle tre, på forsiden nævnte, gruppemedlemmer har bidraget til rapporten på lige vis. Vi har lavet alle forberedelsesøvelser, skrevet koden og afsnittene i rapporten i fællesskab. Denne rapport beskriver vores arbejde og resultater.

Contents

1 Introduktion

This report documents the design and implementation of the control part of a vending-machine for soft-drinks.

The project was divided into three laboratory exercises, in which parts of the system was designed. After the finalization of all three laboratory exercises a complete vending machine control circuit was designed and implemented on a Basys2 FPGA board. The optional parts of the lab exercises is included in the elaboration of each of the exercises, which also redefines some of the mandatory functions, and update/change them as the requirements of each part changes a we move towards the finished control part.

2 Basalt design

The vending machine should be able to sell only one item at a time, not returning any money. Surplus will be left for next purchase. The price of an item is selected automatically when the user selects a product. The machine only accepts 1, 2 and 5 kr. The price of the selected item is displayed as a two-decimal-value on two seven-segment-displays as will the current sum-available-for-purchase.

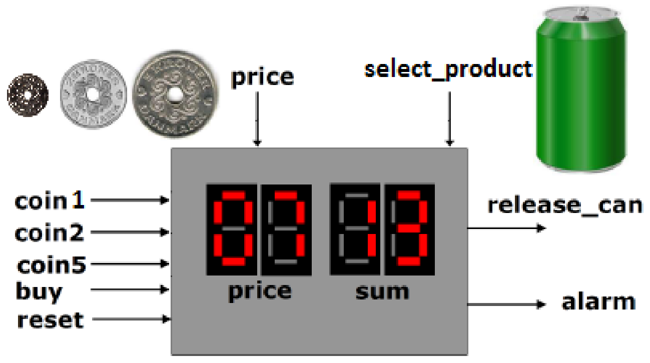


Figure 2: Sketch of the vending machine

As seen on figure 2 there is a number of user inputs. coin1, coin2, coin5 and buy is emulated by push-buttons on the FPGA board. The fifth 'input' select_product, is emulated by switches, which choose between the different types of products, and the prices matching to every type.

When a product is chosen by a switch, the name of the product is shown for approximately 1.3 seconds. After this, the price is shown on the two digits to the left. The rightmost two digits represents the sum of the coins 'inserted' by the push-buttons coin1, coin2 and coin5. When the buy button is asserted, the price is subtracted from the sum if sum is greater than or equal to the price of the product. In the case where the sum is lower than the price, an alarm signal wlll be asserted, causing all four digits to blink for a fixed amount of time (approximately 1.3 seconds) on a 3Hz clock. If the 'Reset' switch is asserted, the sum is set to zero while the price of the products persists.

In designing the Vending Machine, we have created a number og VHDL entities, each controlling a part of the complete machine. Below is a block-diagram of our VHDL implementation of the machine.

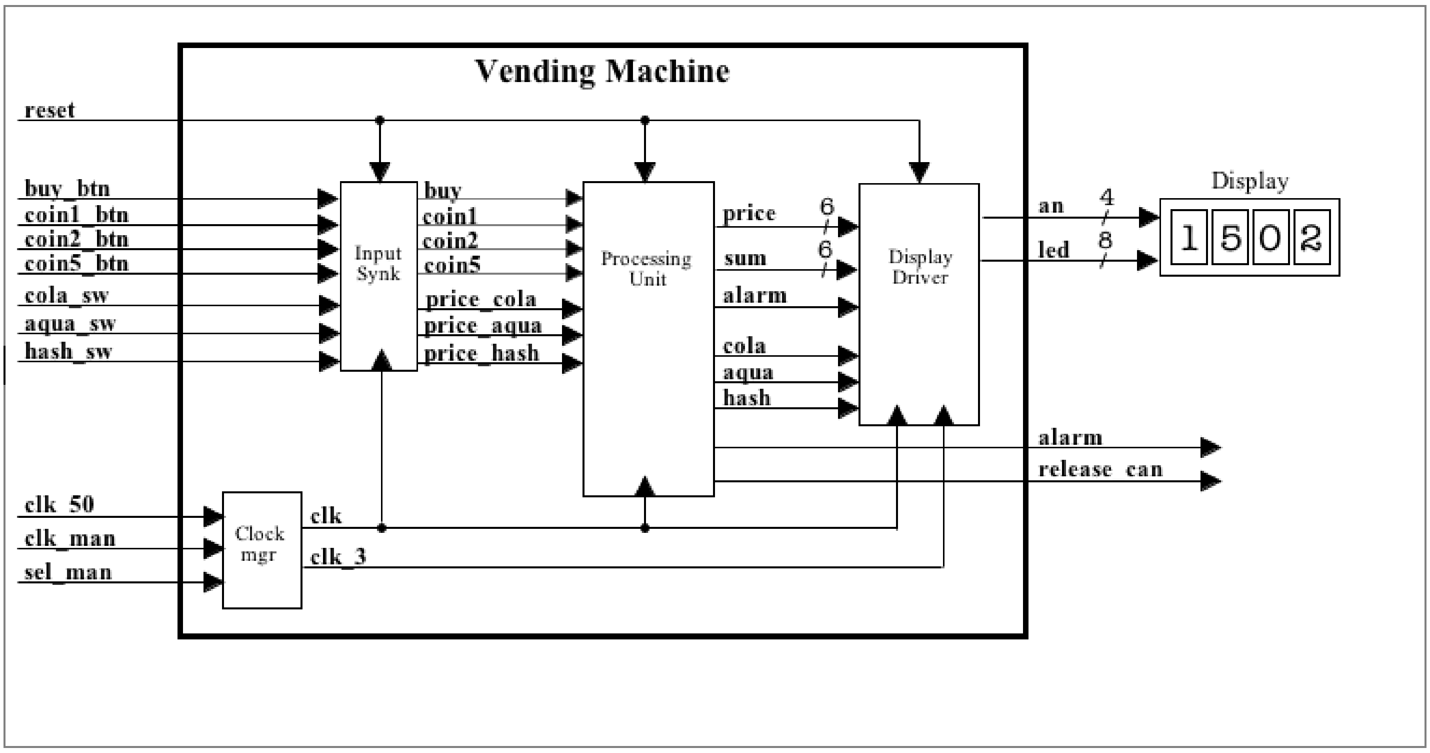


Figure 3: The structure of our VHDL design, including the structure, entities and components

3 Introduktion

This report documents the design and implementation of the control part of a vending-machine for soft-drinks.

The project was divided into three laboratory exercises, in which parts of the system was designed. After the finalization of all three laboratory exercises a complete vending machine control circuit was designed and implemented on a Basys2 FPGA board. The optional parts of the lab exercises is included in the elaboration of each of the exercises, which also redefines some of the mandatory functions, and update/change them as the requirements of each part changes a we move towards the finished control part.

4 Implementation og tests

4.1 Hej Mads

4.2 Hallo

Clock Manager

In this entity the different clock signals are generated as dividers of the master on-board 50 MHz clock.
We generate two sub-clocks. One `clk` that runs at $50000000/(2^16) = 762.94$ Hz. And a slow clock `clk_3` for the alarm-blink signal, which runs at $50000000/(2^24) = 2.98$ Hz.

Input Synchronization and avoiding metastability

All the inputs from the outside world must be synchronized to the clock. This is done to avoid meta-stability, which is a very undesirable condition in digital systems. Meta-stability can occur if an input value is changed too close to the clock edge violating either the setup- or holdtime of the gate. If the setup/hold times are violated, there is no way to tell if the input will go low or high, which of course is a problem in itself. The biggest problem however is not the uncertainty of whether the signal goes low or high, but the fact that there's no way of knowing exactly how long the value will be afloat between low and high. It is only possible to give probabilities of how long a metastable state will last. An example of metastability in practice can be seen on figure 4, where several experiments shows the uncertainty of the whether the signal is going high or low an more importantly the uncertainty of how long the metastable state lasts.

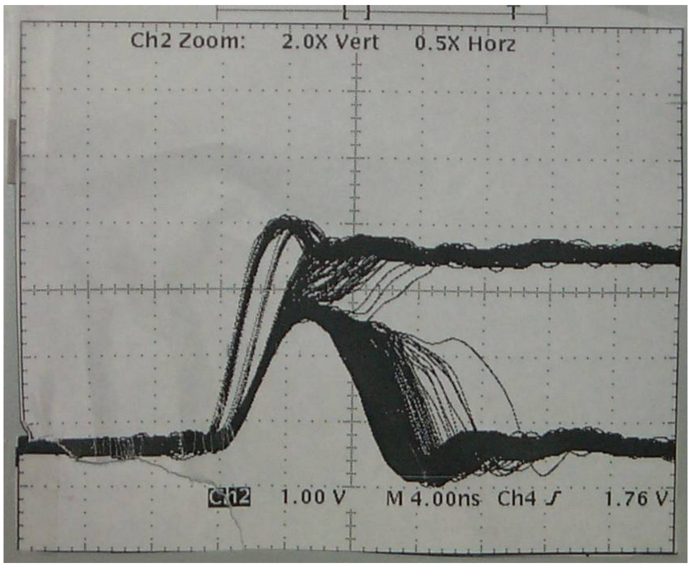


Figure 4: A practical example of metastability

Synchronization is generally done by passing the input value into one or more flip-flop's which is synchronized to the clock. Some signals in our design (`coin1`, `coin2`, `coin5` and `buy`) not only need to be synchronized with the clock, but also need their pass-on signal to be activated for exactly one clock cycle only. This is due to the fact that even though you press a button for only as short a while as you are able to, the clock still ticks at about 760Hz which, in almost every situation, will lead to an active input for several clock cycles. This 1-clockcycle-active is done by running the signal through 2 consecutive flip-flops and only letting the pass-on signal go high as long as the output of the first flip-flop is high and the output from the second flip-flop is low. In order to get the pass-on signal activated again, the input signal needs to be turned off first.

Implementing the display-driver

To begin with as a part of the lab-exercises in this course we implemented the display-driver as a 'Hexadecimal to seven-segment display' state machine, having one state per possible hexadecimal. This was implemented as a Process in VHDL. When the 4-bit binary input `d` changes value (see Appendix A - display_driver), the LED's of the display reflects the change.

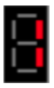


Figure 5: The representation of the 4-bit binary value "0001"

In order to display hex-numbers on a seven-segment display, we have build a combinational circuit in VHDL to convert a 4-bit binary number into the patterns which represent the number on a single seven-segment display.

There is no clever logic between the input and output in this case, since the LED's needed to display the different characters have no logical relation to the value that needs to be on the display. Therefore the implementation is a direct implementation of the state-table.

We implemented our state-table in a `Process(d)`. The input signal `d` is represented as an `UNSIGNED` 4-bit-vector. This process has a nested case statement, which represents the truth table.

The signal `led` is an `UNSIGNED` 8-bit-vector, 7-bits for the lines and one for the dot.

In this frame the first two lines of the `Process(d)` is show.

```
 Case d IS
WHEN "00000" => led <= NOT "11111100"; - 0
WHEN "00001" => led <= NOT "01100000"; - 1
```

Expanding design to binary-coded-decimal instead of hexadecimal

This hex-design was quickly abandoned, as it is not practical to read a price-tag in hexadecimal. Therefore we expanded our design to display binary-coded-decimal (BCD) values on the display instead.

The seven-segment logic was left untouched as the displays still must be able to display the numbers from 0-9, the hex-decimal-states was left untouched but is not used in our design. Later the state table was expanded further to enable us to display the characters needed for the displaying of product-names. See appendix A, figure 11.
The input word `(d)` was changed from a 4-bit `UNSIGNED` to a 5-bit `UNSIGNED`. `d` had to be expanded to make room for the ekstra characters needed for the product names.
Further more the input values to display `price` This allows the two 2-decimal displays (price & sum) to display values between 0-63.

Time multiplexing the four seven-segment-displays

To enable the displays to represent four different values to the user time multiplexing was implemented. Time multiplexing means cycling through the displays, turning them on one by one and displaying the corresponding value on the target display.

This is implemented by a 2-bit counter, incremented by the rising_edge of the clock. The 2-bit overflow makes sure that the counter-value `m` stays between 0-3. The counter is incremented like so: `m_next <= m + 1`. This counter is used a the selector to a multiplexer, selecting the right value for the corresponding display. The counter-value `m` also controls which display-anode `(an)` is turned on. Thus, each display is only turned on when it's value is represented on the 8-bit `led`-vector.

Implementing the Processing Unit

The Processing Unit is the central arithmetic core unit of the vending machine. This component calculates the total amount of coins inserted, and determines whether or not enough coins are inserted to purchase a product and if this is the case, subtracts the product of price from the total amount inserted. The Processing Unit is also the component taking care of asserting an alarm signal if a buy is attempted with too low total coin amount inserted as well as setting a the price and displaying the product name when a product is chosen. Below it is explained how the Processing Unit operates when one of the four different types of inputs are asserted:

- If `Reset` is asserted the sum will be set to zero.

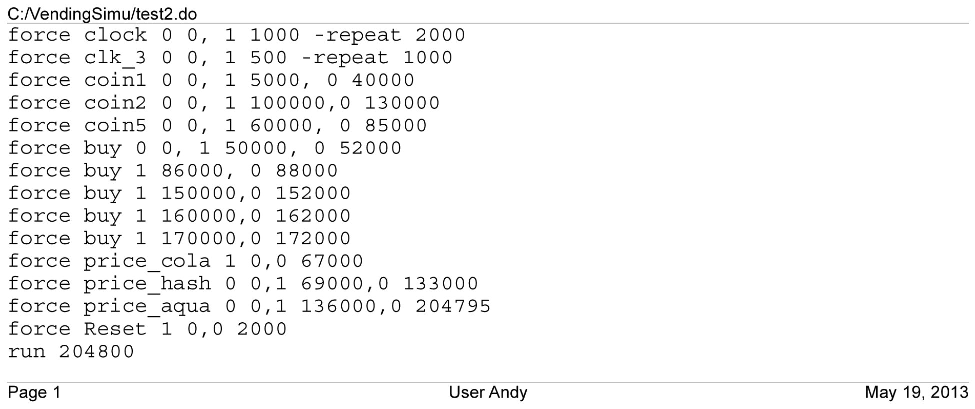
- If one of the three `price_product` signal are asserted the `product` signal will be set to 1 for limited time period. This will cause the `price_out` signal to be set to the value of the product as well as displaying the name of the product on the LEDs for as long as `product` is 1.

- If one of the three `coin` input signal are asserted the Processing Unit will cause the sum signal to be incremented by the value of the coin inserted.

- If the `buy` input is asserted, the machine will calculate whether buy is greater than or equal to price. If this is the case a `release_can` signal is set to 1 and the price is subtracted from the sum. Notice that the `sum` signal won't be set to zero after a successful buy, because the machine is designed in such a way that it doesn't give back money (a good business design!). On the other hand if the `buy` input is asserted when sum is less than price, the alarm signal will be set to 1 for a limited amount of time, causing the display to blink on a 3 Hz clock for a approximately a couple of seconds. Notice here that the alarm counter `alarm_count` runs on the normal 763 Hz clock and therefore the alarm signal is not synchronized to the 3 Hz clock, meaning that the alarm signal can start and stop in the middle of a 3 Hz clock period.

In the `process(coin1, coin2, coin5, buy, clock)`, we have chosen nested if else statements so that Reset have highest precedence, then the coins, then alarm and finally buy. It's obvious that we want Reset to overrule all other inputs, but the coin inputs have higher precedence than alarm as you would always want a machine to register when coins are inserted even though the alarm is on (unless you a mean business man!). buy however has lower precedence than alarm, as we want the machine to stop alarming (aka blinking) before another buy can be attempted.

To test whether the written VHDL code for this component works as intended, we simulate it in the ModelSim environment. This is done by writing a .do file for ModelSim, which sets the input signals to relevant combinations, and simulating clock signals. By doing this it is tested if the output signals are as intended. The .do file can be seen in figure 6.



C:/VendingSimu/test2.do

```
force clock 0 0, 1 1000 -repeat 2000
force clk_3 0 0, 1 500 -repeat 1000
force coin1 0 0, 1 5000, 0 40000
force coin2 0 0, 1 100000,0 130000
force coin5 0 0, 1 60000, 0 85000
force buy 0 0, 1 50000, 0 52000
force buy 1 86000, 0 88000
force buy 1 150000,0 152000
force buy 1 160000,0 162000
force buy 1 170000,0 172000
force price_cola 1 0,0 67000
force price_hash 0 0,1 69000,0 133000
force price_aqua 0 0,1 136000,0 204795
force Reset 1 0,0 2000
run 204800
```

Page 1 User Andy May 19, 2013

Figure 6: Do-file for the Processing Unit ModelSim simulation

The timing diagram in figure 7. is imported from ModelSim and shows the simulated components response to different inputs. As expected the `price_out` changes to the wanted value when a product signal is switched on. It is also seen that `cola_out`, `hash_out` and `aqua_out` respectively is set to 1 when the when the price switch for each of the products is asserted. This is the signal which are passed on to the display driver, which writes the product name on the display. Since the name is set to be there for approximately 2 seconds, this simulation does not show that these `-_out` signal are de-asserted after this period of time, since the simulation only cover 0.002 s, even though the simulation clock ticks faster. It is also seen that the calculating mechanism works correctly since the price is subtracted from the sum and yields the correct value, when buy is asserted. It is seen as well that the `alarm_out` signal is asserted (in the last case where price is higher than sum) exactly as it should. It should be remembered that these signals have not been through the input synchronizer which ensure that coin, buy and price signals only are set to one for a single clock period when activated on the Basys board.

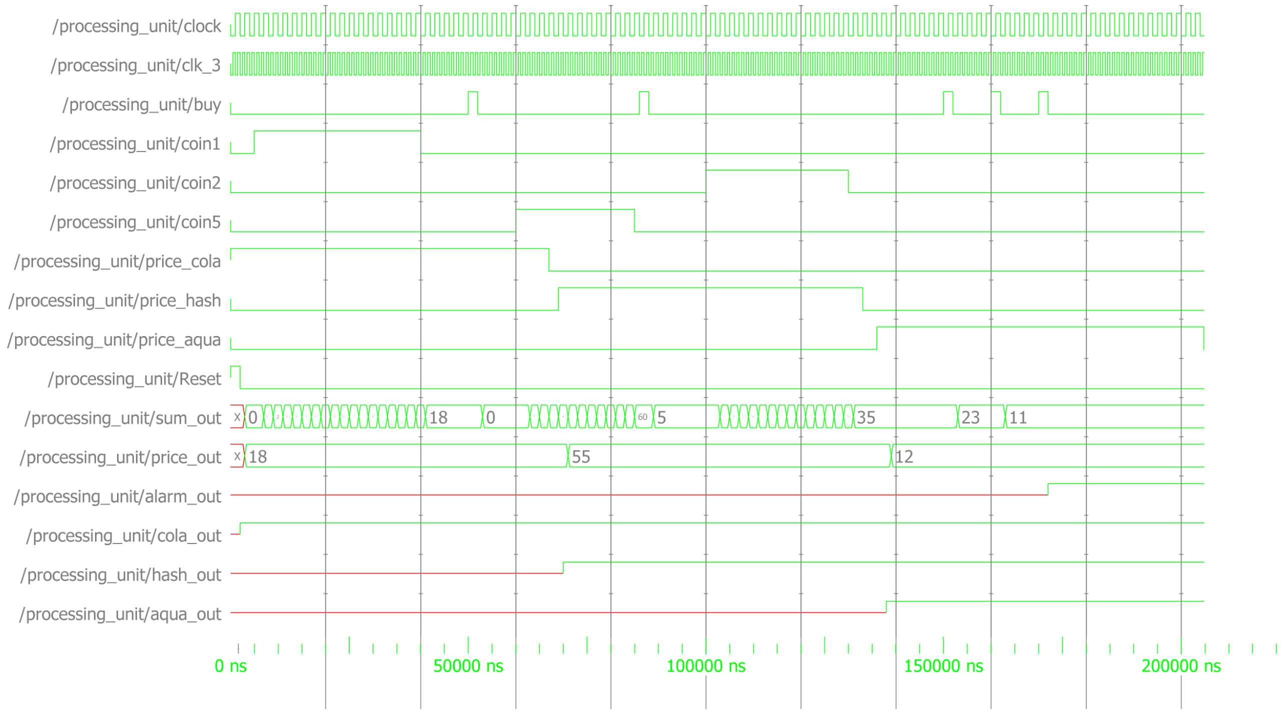


Figure 7: Timing Diagram for the ModelSim simulation

The structure of the Vending Machine

The overall structure of the vending machine can be seen in vending_machine.vhd-file. Here the signals of the different components are tied together to the overall basys-board input and output signals. The components included in vending_machine.vhd includes the clock_manager.vhd, display_driver.vhd, input_synchronizer.vhd and processing_unit.vhd

Generally the overall structure is that the input signals from the Basys board are connected to the input_synchronizer to make the signal go high for only one clock cycle when one of the switches or buttons are asserted. Next the signals are sent to the processing unit to compute sum, price, alarm and so on and then finally to display_driver where the signals are connected back to the Basys board.

On figure 8 the overall functionality of the Vending Machine is displayed as an ASM-Chart. The machine has 'wait' state where price and sum is displayed. The machine will remain in this state until either a coin is inserted, a product is selected or a buy is attempted. If a product is selected, the product name will be displayed and a the price of that specific product will be set. If a coin is inserted sum will be incremented by the coins value. If buy is asserted and sum is less than price, the alarm signal will be set to 1 causing the both price and sum

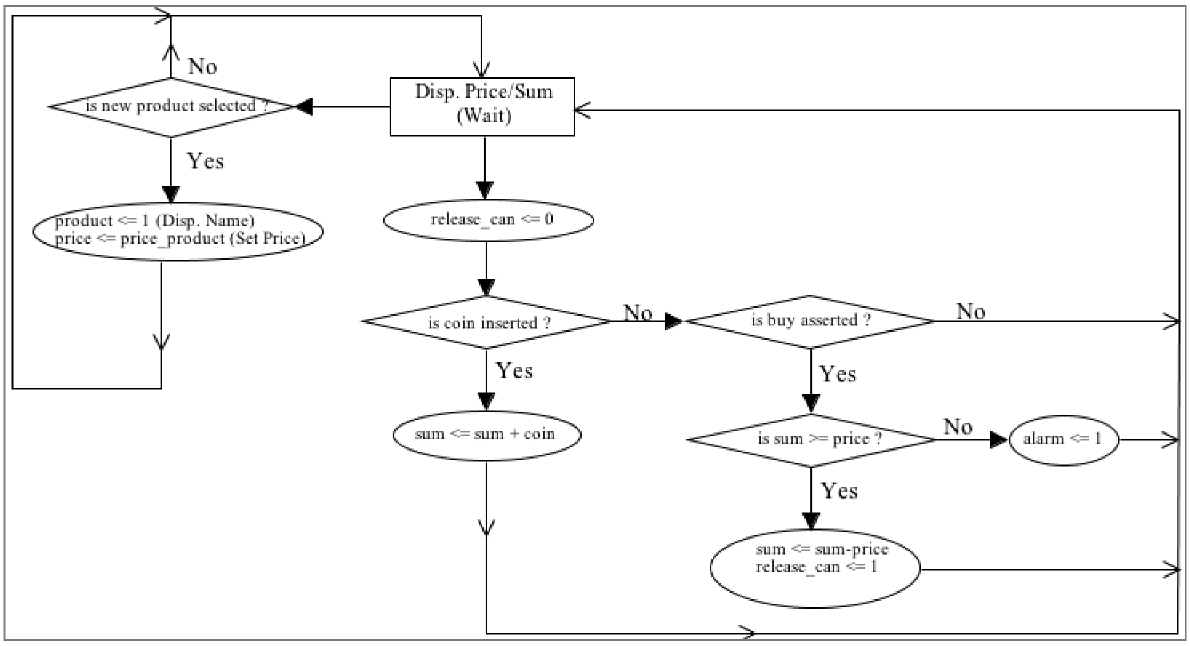


Figure 8: ASM-Chart showing the overall functionality of the vending machine

on the display to blink for a while. If sum is greater than or equal to price, the release_can signal is set to 1 and sum is subtracted the value of that specific product.

5 Resultater

In the end we have designed, tested and implemented following components:

- A clock_manager with an addition of 3 Hz clock `clk_3`
- An input_synchronizer, to synchronize signals from the outside world to the clock, to avoid meta-stability and ensure that some of the signals only goes high for one clock cycle only. This includes buy_out, coin1_out, coin2_out and coin5_out, cola_out, hash_out and aqua_out.
- A display_driver to convert and display price, reset, alarm, sum, cola, hash and aqua-signals on the LED-displays
- A processing_unit to calculate the total amount of coins inserted, comparing price with sum when buy is asserted, and either asserting alarm if sum $<$ price, or subtracting price from sum if sum $\geq$ price. The processing unit also passes on the signals which assert product text on the led display, when a product type is chosen.
- An overall structural component which connects all of the above mentioned components with the in- and outputs on the Basys board.

6 Diskussion

In this section we briefly discuss some of the design choises we have made as well as some expansions we have included in design and finally ideas for further improvements.

Below some of the minor modifications we have made in comparison with the design given in the assignment is shown.

- the release_can signal has been removed from the code as it doesn't have any real function and therefore, in our opinion, only serves to make the code more confusing. In a practical situation it would be a very quick operation to re-add the release_can signal.
- Added coin1 signal which as the name implies is a coin value of 1. we felt this was a logic expansion as most people have 1kr. coins in their purse. As coin1 is placed on one of the buttons on the Basys Board, Reset is instead put on a switch.

In our design of the vending machine we have included the following bigger modifications (expansions).

- Decimal display instead of Hexadecimal display. This is done for both the price and sum outputs displayed on the LEDs. The conversion from a bit-vector to decimal is done with a BCD in the display_driver.
- Blink on the display when the alarm signal is asserted. This will cause all both sum and price to blink on a 3Hz clock. The 3Hz clock is made by clock dividing the 763Hz clock display in the clock_manager.
- Product names on LED display. When a product is chosen it's name is being displayed for approximately 2 seconds on the LEDs. All others operations can still be done while the product is displayed.
- Different products to choose from. This includes the 3 products being cola, aqua and hash. These products have different prices which are set when a product is chosen. Choosing a new product will not reset the sum (the total coin amount inserted).

Having designed the circuit with the expansions that we wanted, there is especially one idea that we had to leave unimplemented.

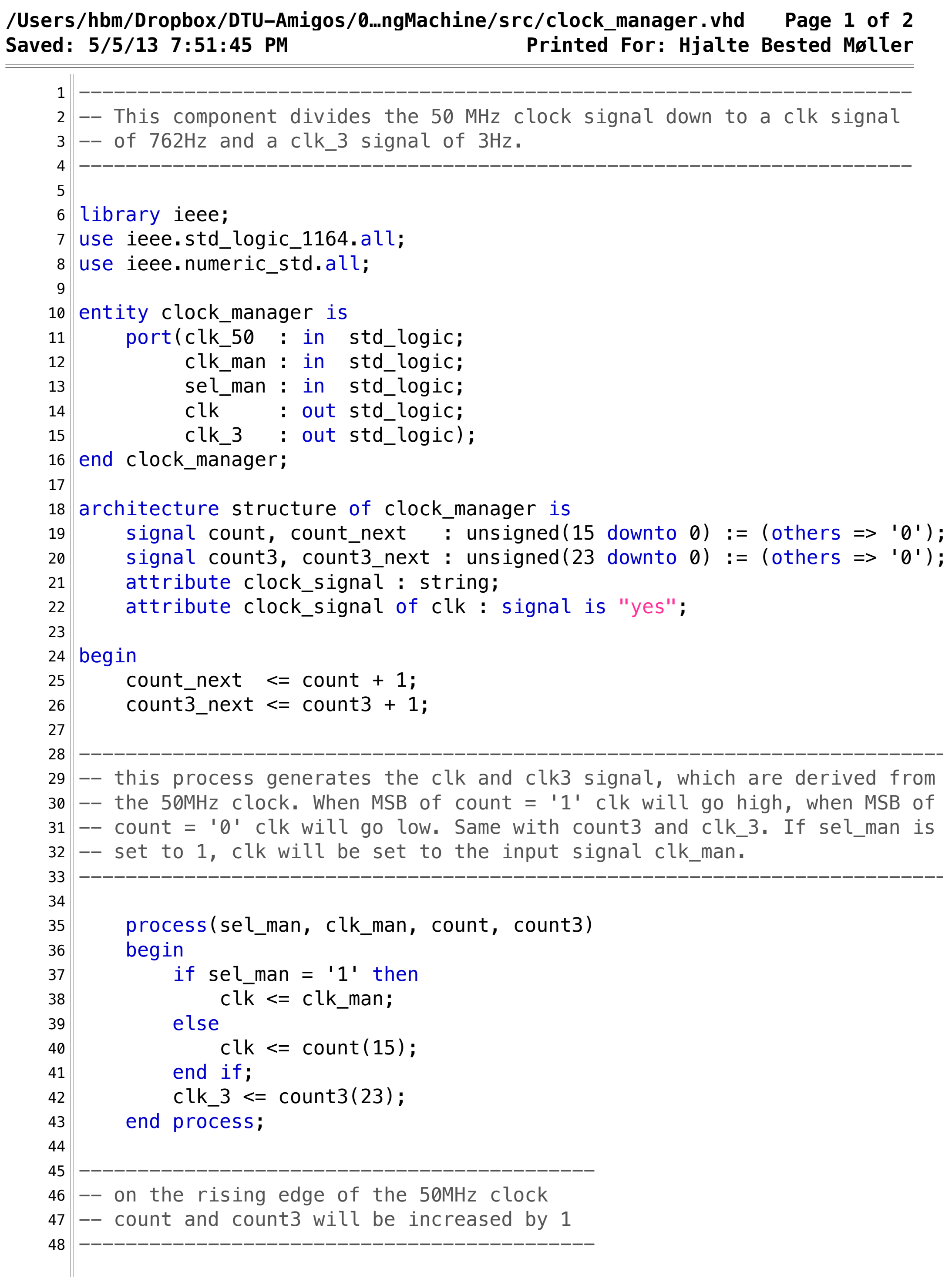
When designing the machine we felt one aspect missing was making the machine able to repay the left-over coin amount when a product is purchased with a cash_out signal. A good way to implement this would be to integer divide the left-over amount with 5 then divide the left-over from this division with 2 and finally divide with 1. In this way it is known how many 5-coins, how many 2-coins and how many 1-coins should be payed back to the user. We find that paying the customers back with these coin amounts makes sense as the machine is running on them and therefore seldomly runs out of them. We have chosen not to implement this an expansion however, as we have no place to send these coin values to. We could of course try to display the cash_out as sentences on the display, telling the customer how many coins of which amounts was returned, but then again it is hard to write on the seven-segment displays as many letters (like M) cannot be displayed on them properly.

7 Konklusion

Efter spillet et blevet designet, skrevet og uploadet til Zilog 6403 microcontrolleren, er alle spillets facetter blevet gennemtestet og fundet i overensstemmelse med det forventede resultat. Vi kan derfor konkludere at den skrevne kode implementerer spillet Reflex Ball på microcontrolleren og i hyperterminalen korrekt i forhold til specifikationskravene. Derudover kan vi også konkludere at alle udvidelse virker som forventet.

A Appendiks 1

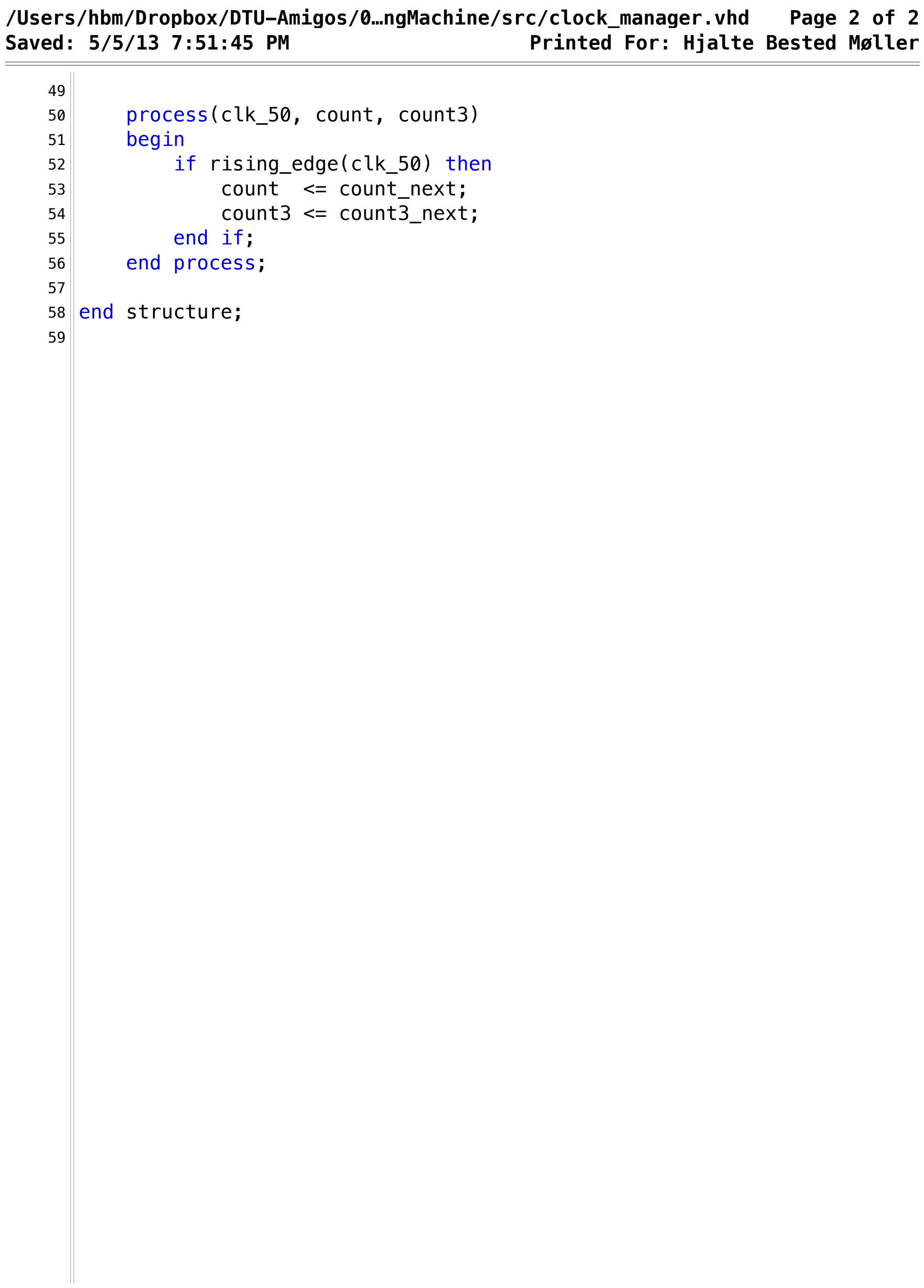
This appendix contains the complete VHDL-code used, and written/modified by us



/Users/hbm/Dropbox/DTU-Amigos/0…ngMachine/src/clock_manager.vhd   Page 1 of 2
Saved: 5/5/13 7:51:45 PM   Printed For: Hjalte Bested Møller

```vhdl
----------------------------------------------------------------------
-- This component divides the 50 MHz clock signal down to a clk signal
-- of 762Hz and a clk_3 signal of 3Hz.
----------------------------------------------------------------------

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

entity clock_manager is
    port(clk_50  : in  std_logic;
         clk_man : in  std_logic;
         sel_man : in  std_logic;
         clk     : out std_logic;
         clk_3   : out std_logic);
end clock_manager;

architecture structure of clock_manager is
    signal count, count_next   : unsigned(15 downto 0) := (others => '0');
    signal count3, count3_next : unsigned(23 downto 0) := (others => '0');
    attribute clock_signal : string;
    attribute clock_signal of clk : signal is "yes";

begin
    count_next  <= count + 1;
    count3_next <= count3 + 1;

-------------------------------------------------------------------------
-- this process generates the clk and clk3 signal, which are derived from
-- the 50MHz clock. When MSB of count = '1' clk will go high, when MSB of
-- count = '0' clk will go low. Same with count3 and clk_3. If sel_man is
-- set to 1, clk will be set to the input signal clk_man.
-------------------------------------------------------------------------

    process(sel_man, clk_man, count, count3)
    begin
        if sel_man = '1' then
            clk <= clk_man;
        else
            clk <= count(15);
        end if;
        clk_3 <= count3(23);
    end process;

--------------------------------------------
-- on the rising edge of the 50MHz clock
-- count and count3 will be increased by 1
--------------------------------------------
```

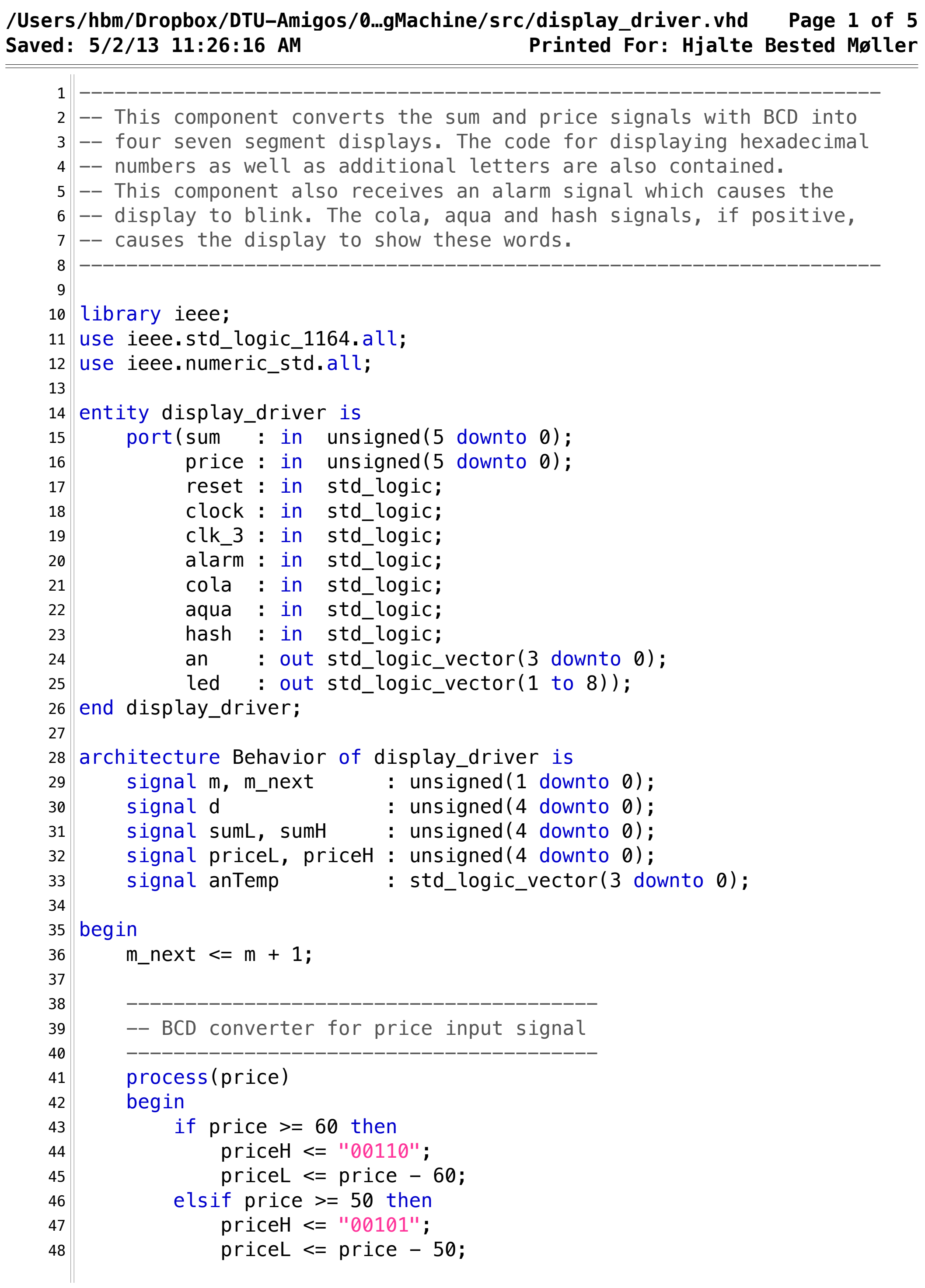
Figure 9: The clock manager, include our own 3 Hz clock for use in the alarm signal



/Users/hbm/Dropbox/DTU-Amigos/0…ngMachine/src/clock_manager.vhd   Page 2 of 2
Saved: 5/5/13 7:51:45 PM   Printed For: Hjalte Bested Møller

```vhdl

    process(clk_50, count, count3)
    begin
        if rising_edge(clk_50) then
            count  <= count_next;
            count3 <= count3_next;
        end if;
    end process;

end structure;

```

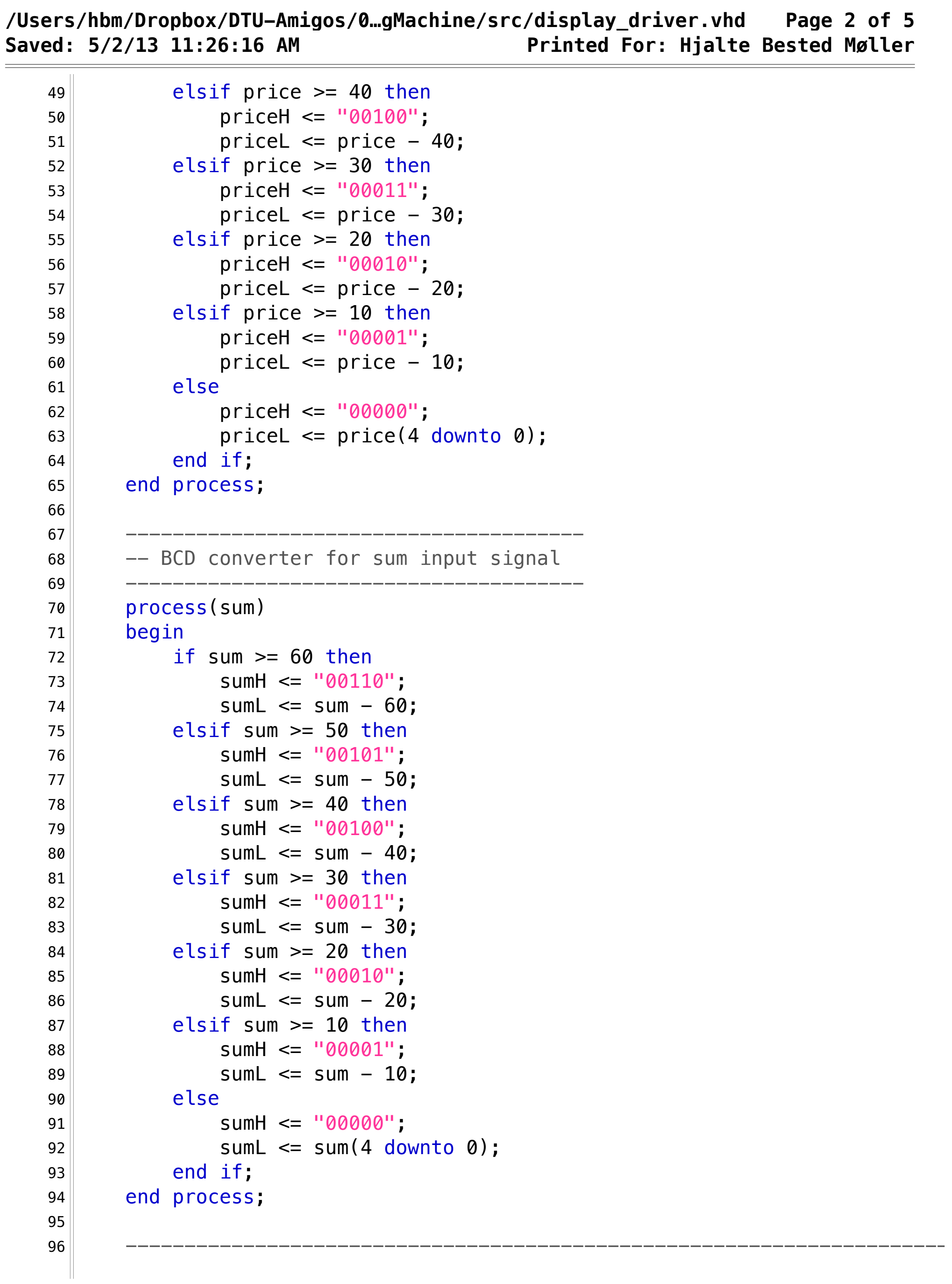
Figure 10: Page 2



/Users/hbm/Dropbox/DTU–Amigos/0…gMachine/src/display_driver.vhd   Page 1 of 5
Saved: 5/2/13 11:26:16 AM   Printed For: Hjalte Bested Møller

```vhdl
---------------------------------------------------------------------
-- This component converts the sum and price signals with BCD into
-- four seven segment displays. The code for displaying hexadecimal
-- numbers as well as additional letters are also contained.
-- This component also receives an alarm signal which causes the
-- display to blink. The cola, aqua and hash signals, if positive,
-- causes the display to show these words.
---------------------------------------------------------------------

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

entity display_driver is
    port(sum   : in  unsigned(5 downto 0);
         price : in  unsigned(5 downto 0);
         reset : in  std_logic;
         clock : in  std_logic;
         clk_3 : in  std_logic;
         alarm : in  std_logic;
         cola  : in  std_logic;
         aqua  : in  std_logic;
         hash  : in  std_logic;
         an    : out std_logic_vector(3 downto 0);
         led   : out std_logic_vector(1 to 8));
end display_driver;

architecture Behavior of display_driver is
    signal m, m_next      : unsigned(1 downto 0);
    signal d              : unsigned(4 downto 0);
    signal sumL, sumH     : unsigned(4 downto 0);
    signal priceL, priceH : unsigned(4 downto 0);
    signal anTemp         : std_logic_vector(3 downto 0);

begin
    m_next <= m + 1;

    ----------------------------------------
    -- BCD converter for price input signal
    ----------------------------------------
    process(price)
    begin
        if price >= 60 then
            priceH <= "00110";
            priceL <= price - 60;
        elsif price >= 50 then
            priceH <= "00101";
            priceL <= price - 50;
```

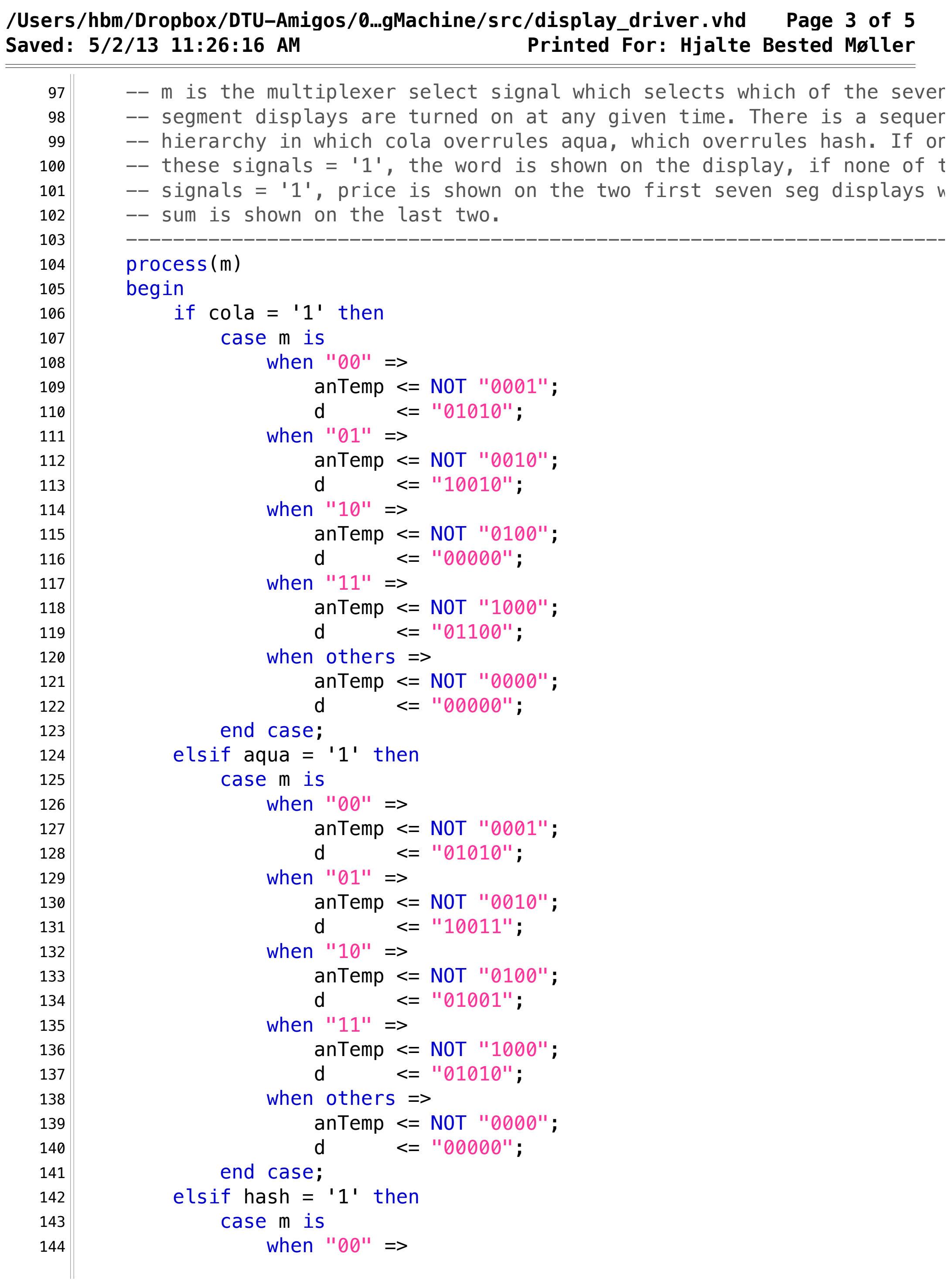
Figure 11: Display driver for displaying symbols/numbers on each of the four seven segment displays



/Users/hbm/Dropbox/DTU-Amigos/0…gMachine/src/display_driver.vhd   Page 2 of 5
Saved: 5/2/13 11:26:16 AM   Printed For: Hjalte Bested Møller

```
        elsif price >= 40 then
            priceH <= "00100";
            priceL <= price - 40;
        elsif price >= 30 then
            priceH <= "00011";
            priceL <= price - 30;
        elsif price >= 20 then
            priceH <= "00010";
            priceL <= price - 20;
        elsif price >= 10 then
            priceH <= "00001";
            priceL <= price - 10;
        else
            priceH <= "00000";
            priceL <= price(4 downto 0);
        end if;
    end process;

    ---------------------------------------
    -- BCD converter for sum input signal
    ---------------------------------------
    process(sum)
    begin
        if sum >= 60 then
            sumH <= "00110";
            sumL <= sum - 60;
        elsif sum >= 50 then
            sumH <= "00101";
            sumL <= sum - 50;
        elsif sum >= 40 then
            sumH <= "00100";
            sumL <= sum - 40;
        elsif sum >= 30 then
            sumH <= "00011";
            sumL <= sum - 30;
        elsif sum >= 20 then
            sumH <= "00010";
            sumL <= sum - 20;
        elsif sum >= 10 then
            sumH <= "00001";
            sumL <= sum - 10;
        else
            sumH <= "00000";
            sumL <= sum(4 downto 0);
        end if;
    end process;

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

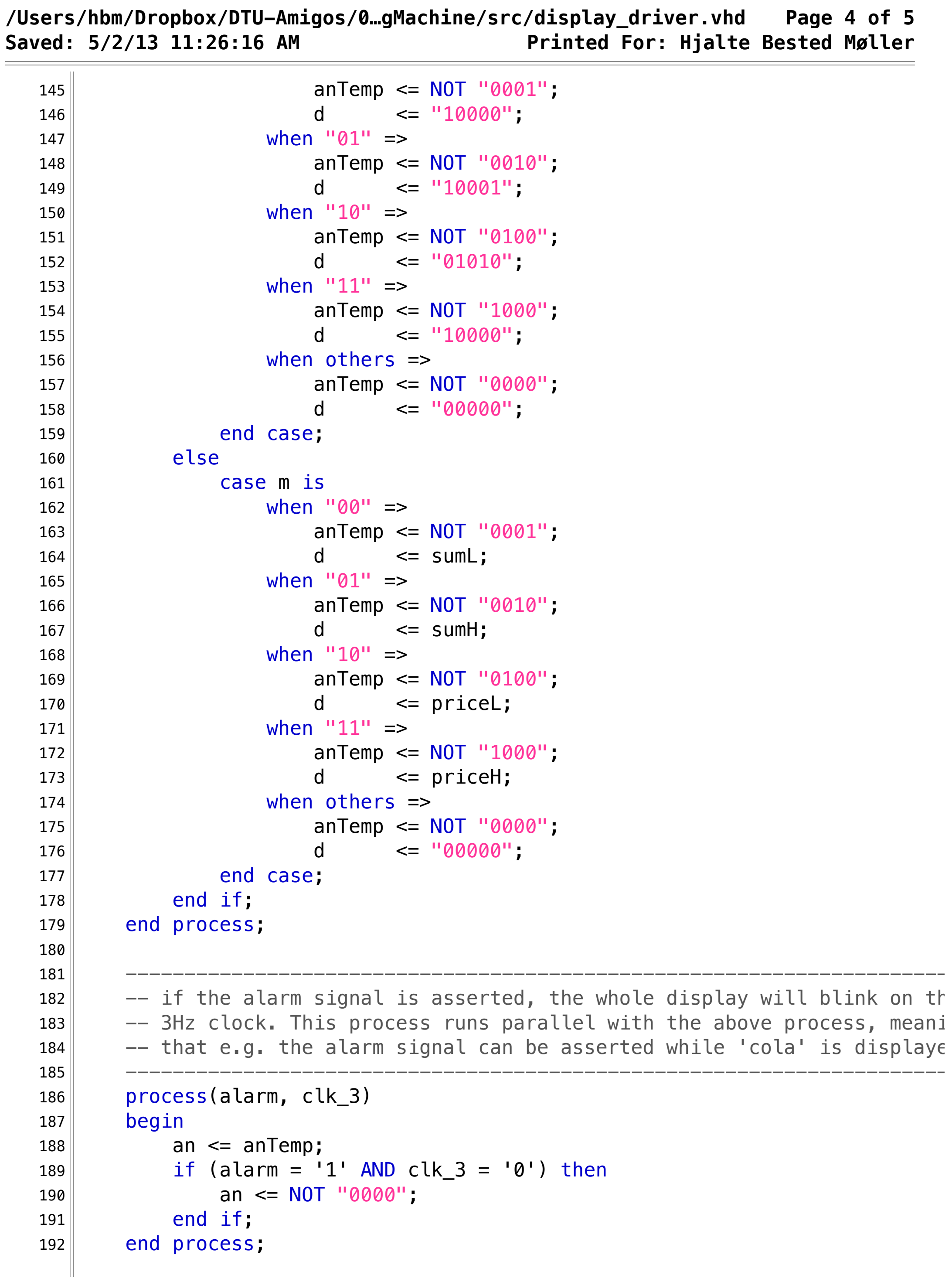
Figure 12: Page 2



/Users/hbm/Dropbox/DTU-Amigos/0…gMachine/src/display_driver.vhd   Page 3 of 5
Saved: 5/2/13 11:26:16 AM   Printed For: Hjalte Bested Møller

```
    -- m is the multiplexer select signal which selects which of the sever
    -- segment displays are turned on at any given time. There is a sequer
    -- hierarchy in which cola overrules aqua, which overrules hash. If or
    -- these signals = '1', the word is shown on the display, if none of t
    -- signals = '1', price is shown on the two first seven seg displays w
    -- sum is shown on the last two.
    -----------------------------------------------------------------------
    process(m)
    begin
        if cola = '1' then
            case m is
                when "00" =>
                    anTemp <= NOT "0001";
                    d      <= "01010";
                when "01" =>
                    anTemp <= NOT "0010";
                    d      <= "10010";
                when "10" =>
                    anTemp <= NOT "0100";
                    d      <= "00000";
                when "11" =>
                    anTemp <= NOT "1000";
                    d      <= "01100";
                when others =>
                    anTemp <= NOT "0000";
                    d      <= "00000";
            end case;
        elsif aqua = '1' then
            case m is
                when "00" =>
                    anTemp <= NOT "0001";
                    d      <= "01010";
                when "01" =>
                    anTemp <= NOT "0010";
                    d      <= "10011";
                when "10" =>
                    anTemp <= NOT "0100";
                    d      <= "01001";
                when "11" =>
                    anTemp <= NOT "1000";
                    d      <= "01010";
                when others =>
                    anTemp <= NOT "0000";
                    d      <= "00000";
            end case;
        elsif hash = '1' then
            case m is
                when "00" =>
```

Figure 13: Page 3



/Users/hbm/Dropbox/DTU-Amigos/0…gMachine/src/display_driver.vhd   Page 4 of 5
Saved: 5/2/13 11:26:16 AM   Printed For: Hjalte Bested Møller

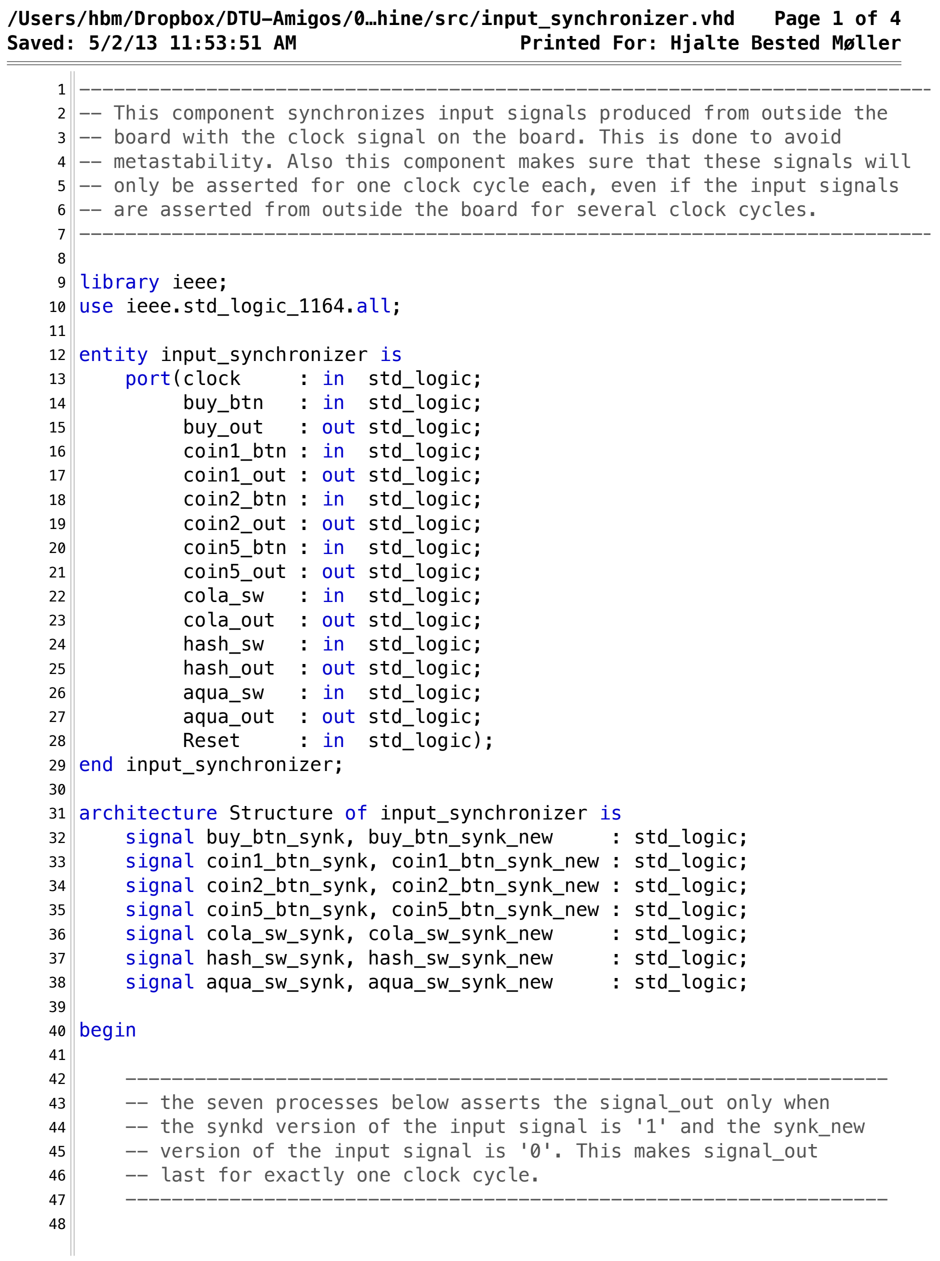
```
                    anTemp <= NOT "0001";
                    d      <= "10000";
                when "01" =>
                    anTemp <= NOT "0010";
                    d      <= "10001";
                when "10" =>
                    anTemp <= NOT "0100";
                    d      <= "01010";
                when "11" =>
                    anTemp <= NOT "1000";
                    d      <= "10000";
                when others =>
                    anTemp <= NOT "0000";
                    d      <= "00000";
            end case;
        else
            case m is
                when "00" =>
                    anTemp <= NOT "0001";
                    d      <= sumL;
                when "01" =>
                    anTemp <= NOT "0010";
                    d      <= sumH;
                when "10" =>
                    anTemp <= NOT "0100";
                    d      <= priceL;
                when "11" =>
                    anTemp <= NOT "1000";
                    d      <= priceH;
                when others =>
                    anTemp <= NOT "0000";
                    d      <= "00000";
            end case;
        end if;
    end process;

    ----------------------------------------------------------------------
    -- if the alarm signal is asserted, the whole display will blink on th
    -- 3Hz clock. This process runs parallel with the above process, meani
    -- that e.g. the alarm signal can be asserted while 'cola' is displaye
    ----------------------------------------------------------------------
    process(alarm, clk_3)
    begin
        an <= anTemp;
        if (alarm = '1' AND clk_3 = '0') then
            an <= NOT "0000";
        end if;
    end process;
```

Figure 14: Page 4

/Users/hbm/Dropbox/DTU-Amigos/0…gMachine/src/display_driver.vhd Page 5 of 5
Saved: 5/2/13 11:26:16 AM Printed For: Hjalte Bested Møller

```

    ----------------------------------------------------------------
    -- this process assigns each hexadecimal number (0-F) as well
    -- as some additional letters to different values of d.
    ----------------------------------------------------------------
    process(d)
    begin
        case d is
            when "00000" => led <= NOT "11111100"; -- 0
            when "00001" => led <= NOT "01100000"; -- 1
            when "00010" => led <= NOT "11011010"; -- 2
            when "00011" => led <= NOT "11110010"; -- 3
            when "00100" => led <= NOT "01100110"; -- 4
            when "00101" => led <= NOT "10110110"; -- 5
            when "00110" => led <= NOT "10111110"; -- 6
            when "00111" => led <= NOT "11100000"; -- 7
            when "01000" => led <= NOT "11111110"; -- 8
            when "01001" => led <= NOT "11100110"; -- 9
            when "01010" => led <= NOT "11101110"; -- A
            when "01011" => led <= NOT "00111110"; -- b
            when "01100" => led <= NOT "10011100"; -- C
            when "01101" => led <= NOT "01111010"; -- d
            when "01110" => led <= NOT "10011110"; -- E
            when "01111" => led <= NOT "10001110"; -- F
            when "10000" => led <= NOT "01101110"; -- H
            when "10001" => led <= NOT "10110110"; -- S
            when "10010" => led <= NOT "00011100"; -- L
            when "10011" => led <= NOT "01111100"; -- U
            when others  => led <= (others => '0');
        end case;
    end process;

    ------------------------------------------------
    -- the two bit vector m is increased by one on
    -- each positive clock edge of the 762Hz clock
    ------------------------------------------------
    process(clock)
    begin
        if rising_edge(clock) then
            m <= m_next;
        end if;
    end process;
end Behavior;

```

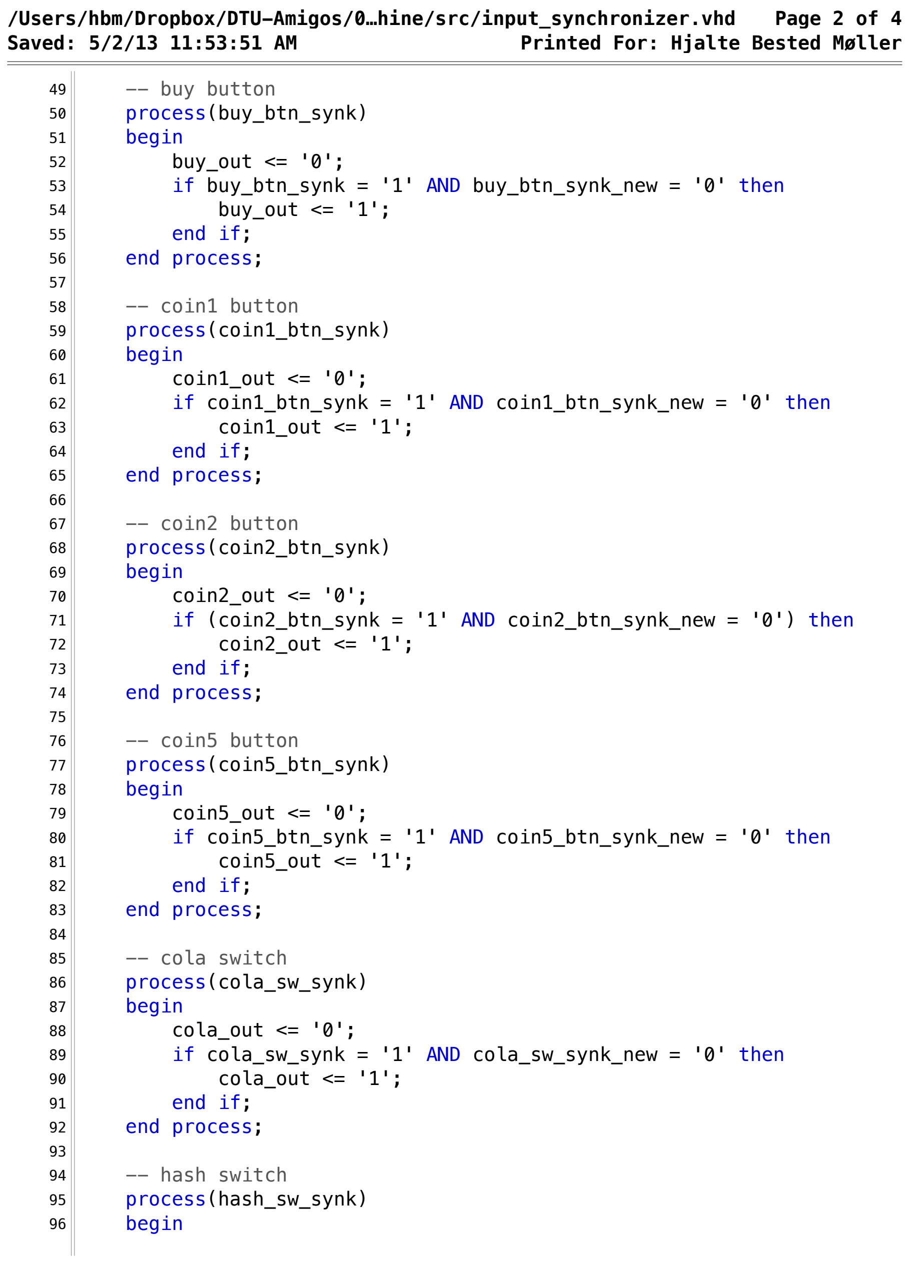
Figure 15: Page 5



/Users/hbm/Dropbox/DTU-Amigos/0…hine/src/input_synchronizer.vhd   Page 1 of 4
Saved: 5/2/13 11:53:51 AM   Printed For: Hjalte Bested Møller

```vhdl
--------------------------------------------------------------------------
-- This component synchronizes input signals produced from outside the
-- board with the clock signal on the board. This is done to avoid
-- metastability. Also this component makes sure that these signals will
-- only be asserted for one clock cycle each, even if the input signals
-- are asserted from outside the board for several clock cycles.
--------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity input_synchronizer is
    port(clock     : in  std_logic;
         buy_btn   : in  std_logic;
         buy_out   : out std_logic;
         coin1_btn : in  std_logic;
         coin1_out : out std_logic;
         coin2_btn : in  std_logic;
         coin2_out : out std_logic;
         coin5_btn : in  std_logic;
         coin5_out : out std_logic;
         cola_sw   : in  std_logic;
         cola_out  : out std_logic;
         hash_sw   : in  std_logic;
         hash_out  : out std_logic;
         aqua_sw   : in  std_logic;
         aqua_out  : out std_logic;
         Reset     : in  std_logic);
end input_synchronizer;

architecture Structure of input_synchronizer is
    signal buy_btn_synk, buy_btn_synk_new     : std_logic;
    signal coin1_btn_synk, coin1_btn_synk_new : std_logic;
    signal coin2_btn_synk, coin2_btn_synk_new : std_logic;
    signal coin5_btn_synk, coin5_btn_synk_new : std_logic;
    signal cola_sw_synk, cola_sw_synk_new     : std_logic;
    signal hash_sw_synk, hash_sw_synk_new     : std_logic;
    signal aqua_sw_synk, aqua_sw_synk_new     : std_logic;

begin

    ------------------------------------------------------------------
    -- the seven processes below asserts the signal_out only when
    -- the synkd version of the input signal is '1' and the synk_new
    -- version of the input signal is '0'. This makes signal_out
    -- last for exactly one clock cycle.
    ------------------------------------------------------------------

```

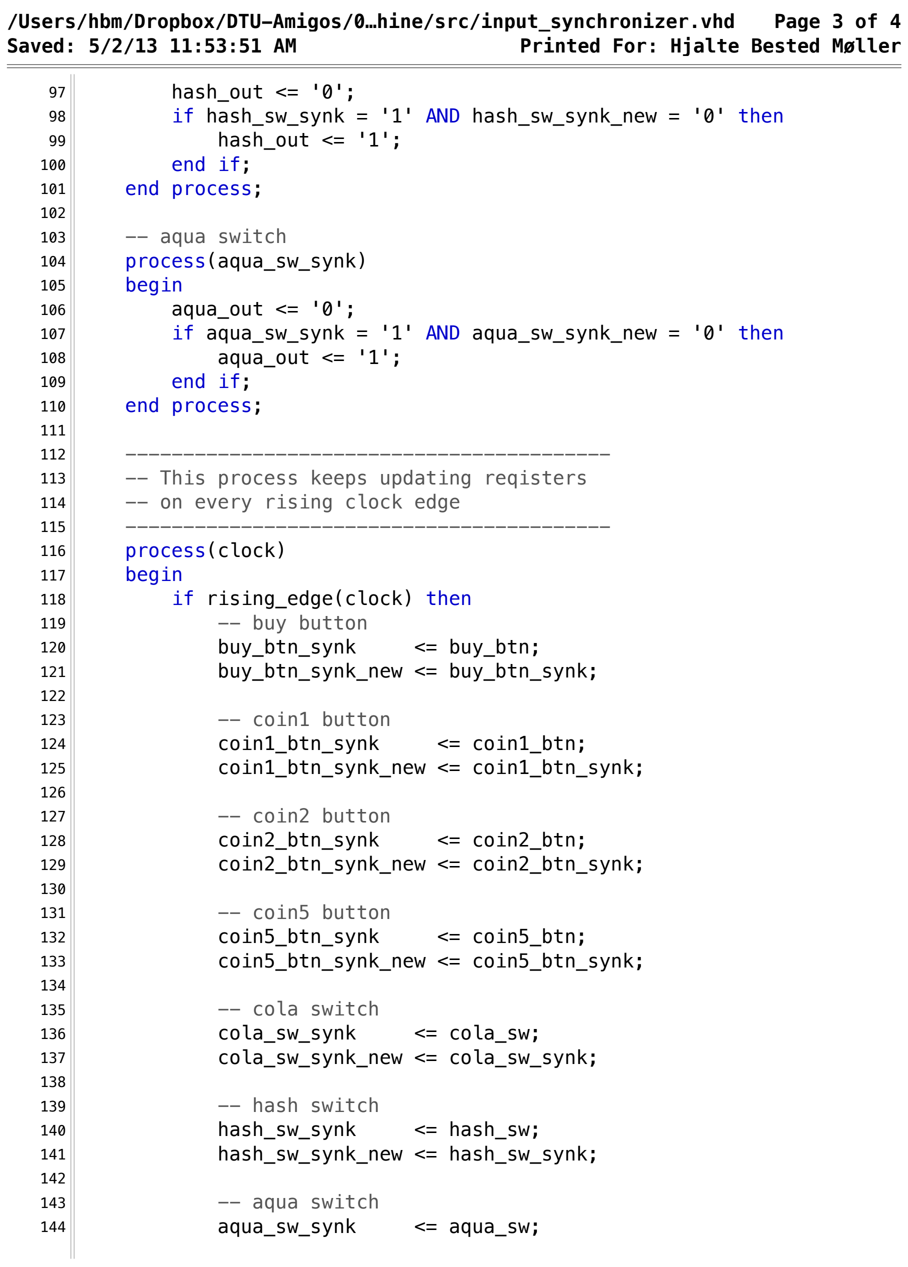
Figure 16: VHDL code for the input synchronizer which synchronizes the input signals to the clock



/Users/hbm/Dropbox/DTU-Amigos/0…hine/src/input_synchronizer.vhd   Page 2 of 4
Saved: 5/2/13 11:53:51 AM   Printed For: Hjalte Bested Møller

```
    -- buy button
    process(buy_btn_synk)
    begin
        buy_out <= '0';
        if buy_btn_synk = '1' AND buy_btn_synk_new = '0' then
            buy_out <= '1';
        end if;
    end process;

    -- coin1 button
    process(coin1_btn_synk)
    begin
        coin1_out <= '0';
        if coin1_btn_synk = '1' AND coin1_btn_synk_new = '0' then
            coin1_out <= '1';
        end if;
    end process;

    -- coin2 button
    process(coin2_btn_synk)
    begin
        coin2_out <= '0';
        if (coin2_btn_synk = '1' AND coin2_btn_synk_new = '0') then
            coin2_out <= '1';
        end if;
    end process;

    -- coin5 button
    process(coin5_btn_synk)
    begin
        coin5_out <= '0';
        if coin5_btn_synk = '1' AND coin5_btn_synk_new = '0' then
            coin5_out <= '1';
        end if;
    end process;

    -- cola switch
    process(cola_sw_synk)
    begin
        cola_out <= '0';
        if cola_sw_synk = '1' AND cola_sw_synk_new = '0' then
            cola_out <= '1';
        end if;
    end process;

    -- hash switch
    process(hash_sw_synk)
    begin
```

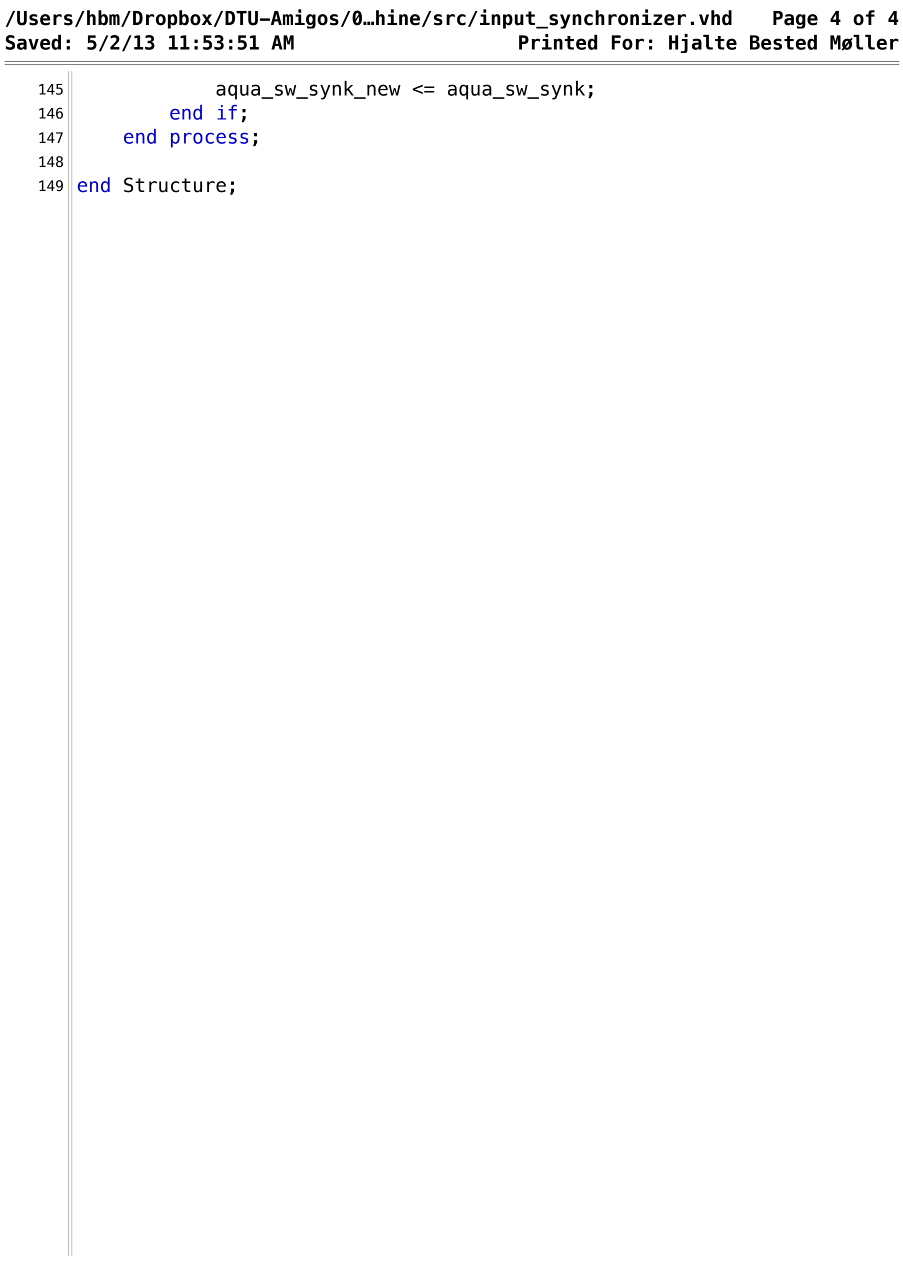
Figure 17: Page 2



/Users/hbm/Dropbox/DTU-Amigos/0…hine/src/input_synchronizer.vhd   Page 3 of 4
Saved: 5/2/13 11:53:51 AM   Printed For: Hjalte Bested Møller

```vhdl
        hash_out <= '0';
        if hash_sw_synk = '1' AND hash_sw_synk_new = '0' then
            hash_out <= '1';
        end if;
    end process;

    -- aqua switch
    process(aqua_sw_synk)
    begin
        aqua_out <= '0';
        if aqua_sw_synk = '1' AND aqua_sw_synk_new = '0' then
            aqua_out <= '1';
        end if;
    end process;

    ------------------------------------------
    -- This process keeps updating reqisters
    -- on every rising clock edge
    ------------------------------------------
    process(clock)
    begin
        if rising_edge(clock) then
            -- buy button
            buy_btn_synk     <= buy_btn;
            buy_btn_synk_new <= buy_btn_synk;

            -- coin1 button
            coin1_btn_synk     <= coin1_btn;
            coin1_btn_synk_new <= coin1_btn_synk;

            -- coin2 button
            coin2_btn_synk     <= coin2_btn;
            coin2_btn_synk_new <= coin2_btn_synk;

            -- coin5 button
            coin5_btn_synk     <= coin5_btn;
            coin5_btn_synk_new <= coin5_btn_synk;

            -- cola switch
            cola_sw_synk     <= cola_sw;
            cola_sw_synk_new <= cola_sw_synk;

            -- hash switch
            hash_sw_synk     <= hash_sw;
            hash_sw_synk_new <= hash_sw_synk;

            -- aqua switch
            aqua_sw_synk     <= aqua_sw;
```

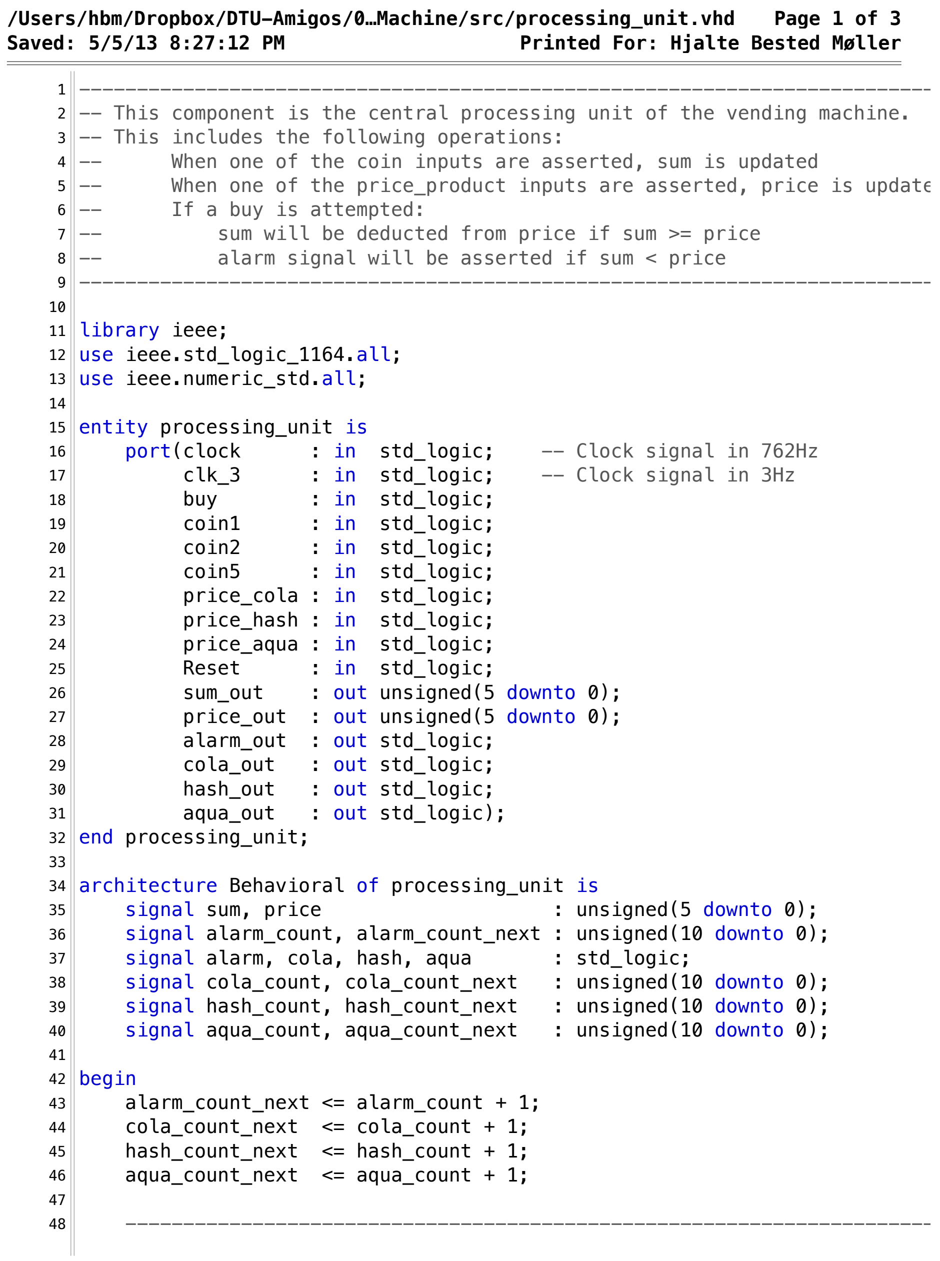
Figure 18: Page 3



/Users/hbm/Dropbox/DTU-Amigos/0…hine/src/input_synchronizer.vhd Page 4 of 4
Saved: 5/2/13 11:53:51 AM Printed For: Hjalte Bested Møller

```vhdl
            aqua_sw_synk_new <= aqua_sw_synk;
        end if;
    end process;

end Structure;
```

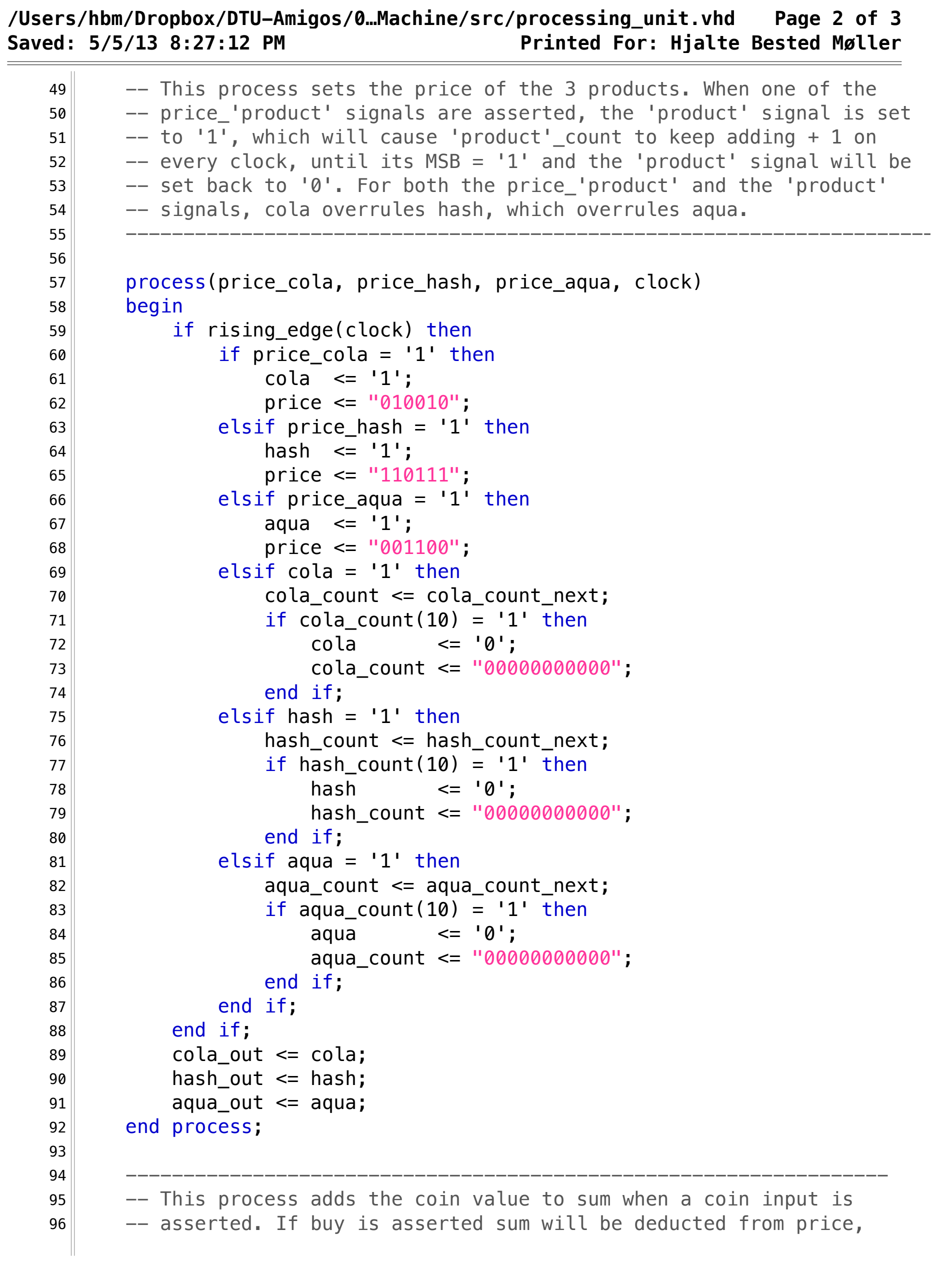
Figure 19: Page 4



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/processing_unit.vhd   Page 1 of 3
Saved: 5/5/13 8:27:12 PM   Printed For: Hjalte Besterd Møller

```
--------------------------------------------------------------------------
-- This component is the central processing unit of the vending machine.
-- This includes the following operations:
--      When one of the coin inputs are asserted, sum is updated
--      When one of the price_product inputs are asserted, price is update
--      If a buy is attempted:
--          sum will be deducted from price if sum >= price
--          alarm signal will be asserted if sum < price
--------------------------------------------------------------------------

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

entity processing_unit is
    port(clock      : in  std_logic;    -- Clock signal in 762Hz
         clk_3      : in  std_logic;    -- Clock signal in 3Hz
         buy        : in  std_logic;
         coin1      : in  std_logic;
         coin2      : in  std_logic;
         coin5      : in  std_logic;
         price_cola : in  std_logic;
         price_hash : in  std_logic;
         price_aqua : in  std_logic;
         Reset      : in  std_logic;
         sum_out    : out unsigned(5 downto 0);
         price_out  : out unsigned(5 downto 0);
         alarm_out  : out std_logic;
         cola_out   : out std_logic;
         hash_out   : out std_logic;
         aqua_out   : out std_logic);
end processing_unit;

architecture Behavioral of processing_unit is
    signal sum, price                     : unsigned(5 downto 0);
    signal alarm_count, alarm_count_next : unsigned(10 downto 0);
    signal alarm, cola, hash, aqua       : std_logic;
    signal cola_count, cola_count_next   : unsigned(10 downto 0);
    signal hash_count, hash_count_next   : unsigned(10 downto 0);
    signal aqua_count, aqua_count_next   : unsigned(10 downto 0);

begin
    alarm_count_next <= alarm_count + 1;
    cola_count_next  <= cola_count + 1;
    hash_count_next  <= hash_count + 1;
    aqua_count_next  <= aqua_count + 1;

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

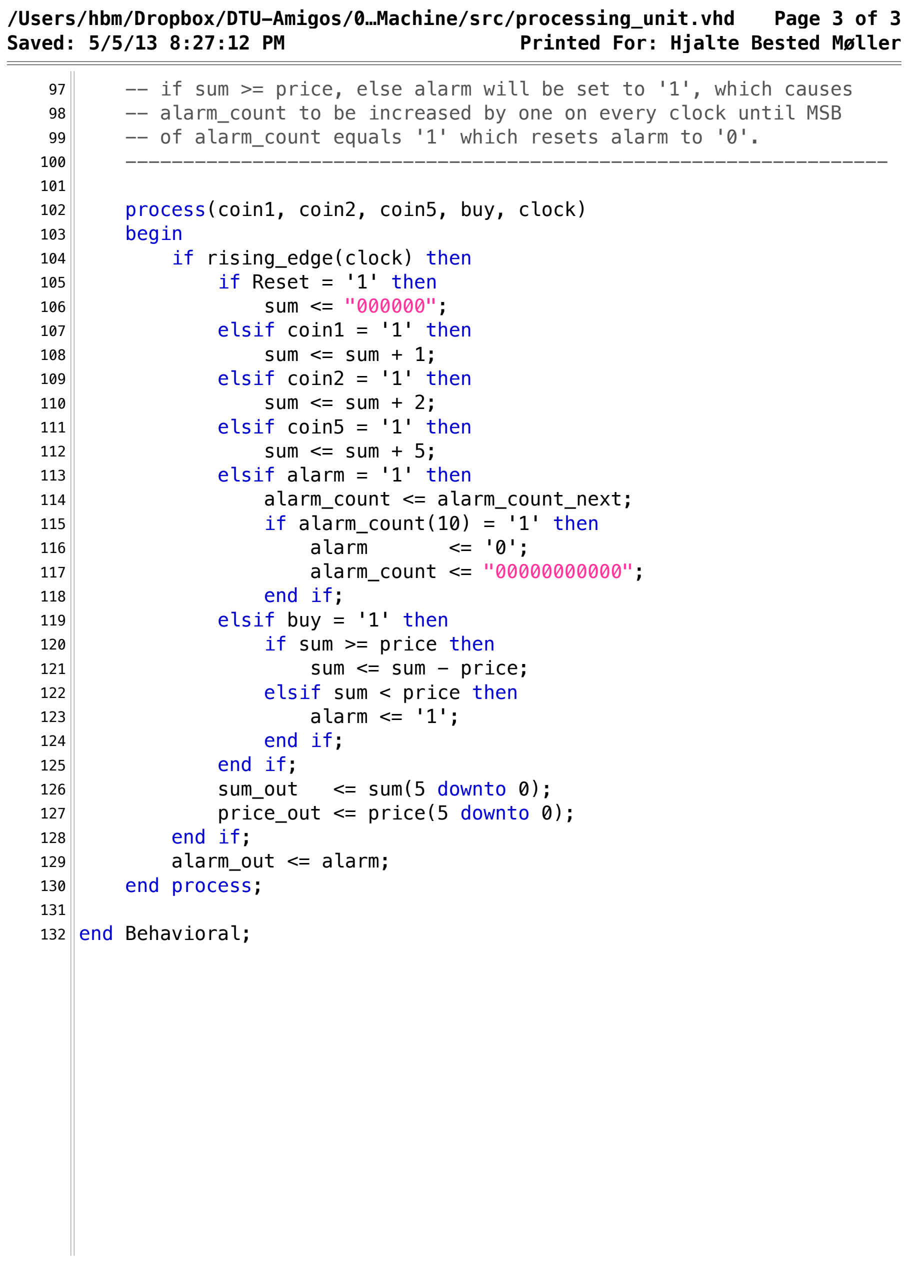
Figure 20: Processing unit, for calculating/setting price/sum



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/processing_unit.vhd   Page 2 of 3
Saved: 5/5/13 8:27:12 PM   Printed For: Hjalte Bested Møller

```vhdl
-- This process sets the price of the 3 products. When one of the
-- price_'product' signals are asserted, the 'product' signal is set
-- to '1', which will cause 'product'_count to keep adding + 1 on
-- every clock, until its MSB = '1' and the 'product' signal will be
-- set back to '0'. For both the price_'product' and the 'product'
-- signals, cola overrules hash, which overrules aqua.
----------------------------------------------------------------------

process(price_cola, price_hash, price_aqua, clock)
begin
    if rising_edge(clock) then
        if price_cola = '1' then
            cola  <= '1';
            price <= "010010";
        elsif price_hash = '1' then
            hash  <= '1';
            price <= "110111";
        elsif price_aqua = '1' then
            aqua  <= '1';
            price <= "001100";
        elsif cola = '1' then
            cola_count <= cola_count_next;
            if cola_count(10) = '1' then
                cola       <= '0';
                cola_count <= "00000000000";
            end if;
        elsif hash = '1' then
            hash_count <= hash_count_next;
            if hash_count(10) = '1' then
                hash       <= '0';
                hash_count <= "00000000000";
            end if;
        elsif aqua = '1' then
            aqua_count <= aqua_count_next;
            if aqua_count(10) = '1' then
                aqua       <= '0';
                aqua_count <= "00000000000";
            end if;
        end if;
    end if;
    cola_out <= cola;
    hash_out <= hash;
    aqua_out <= aqua;
end process;

------------------------------------------------------------------
-- This process adds the coin value to sum when a coin input is
-- asserted. If buy is asserted sum will be deducted from price,
```

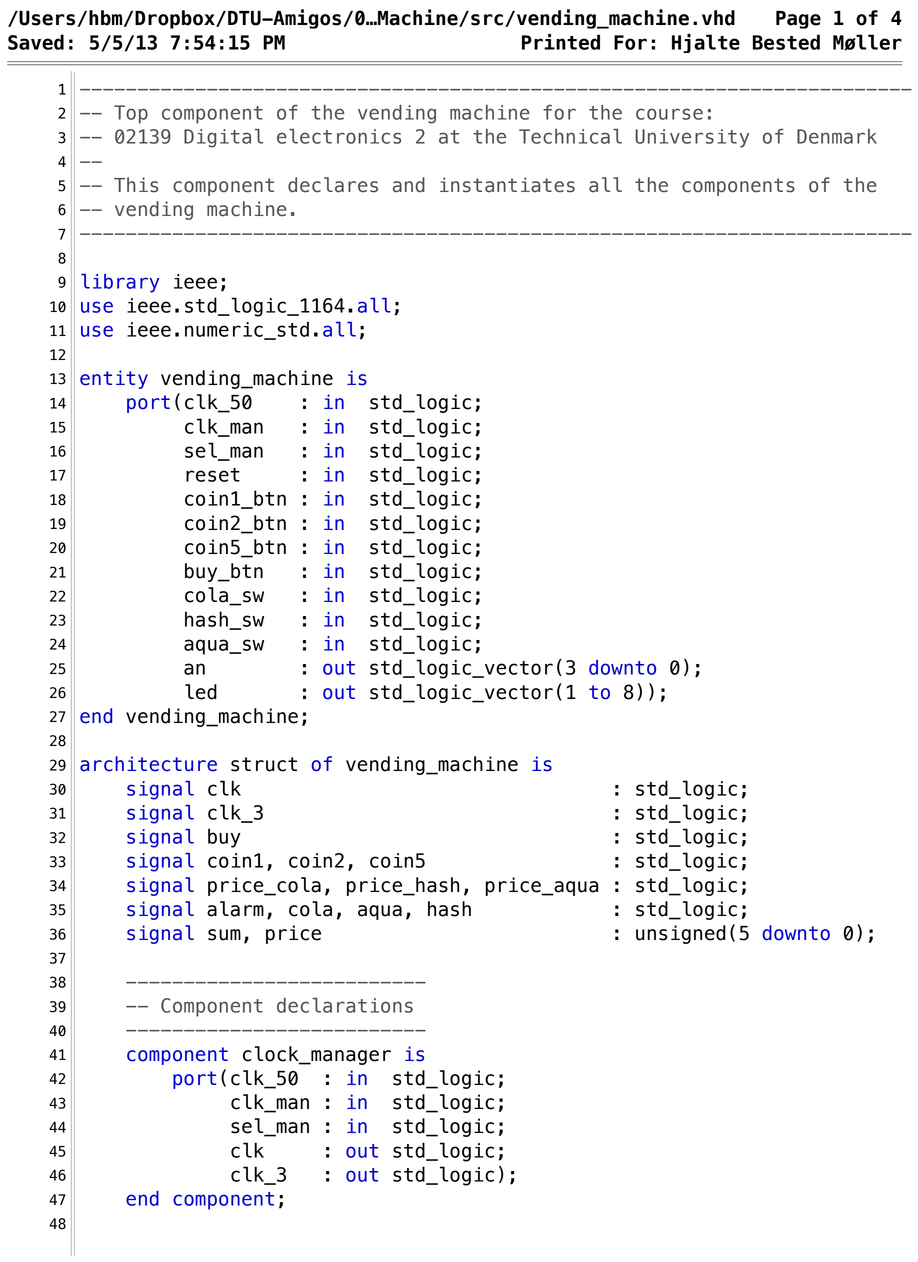
Figure 21: Page 2



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/processing_unit.vhd   Page 3 of 3
Saved: 5/5/13 8:27:12 PM   Printed For: Hjalte Bested Møller

```vhdl
    -- if sum >= price, else alarm will be set to '1', which causes
    -- alarm_count to be increased by one on every clock until MSB
    -- of alarm_count equals '1' which resets alarm to '0'.
    ------------------------------------------------------------------

    process(coin1, coin2, coin5, buy, clock)
    begin
        if rising_edge(clock) then
            if Reset = '1' then
                sum <= "000000";
            elsif coin1 = '1' then
                sum <= sum + 1;
            elsif coin2 = '1' then
                sum <= sum + 2;
            elsif coin5 = '1' then
                sum <= sum + 5;
            elsif alarm = '1' then
                alarm_count <= alarm_count_next;
                if alarm_count(10) = '1' then
                    alarm       <= '0';
                    alarm_count <= "00000000000";
                end if;
            elsif buy = '1' then
                if sum >= price then
                    sum <= sum - price;
                elsif sum < price then
                    alarm <= '1';
                end if;
            end if;
            sum_out   <= sum(5 downto 0);
            price_out <= price(5 downto 0);
        end if;
        alarm_out <= alarm;
    end process;

end Behavioral;
```

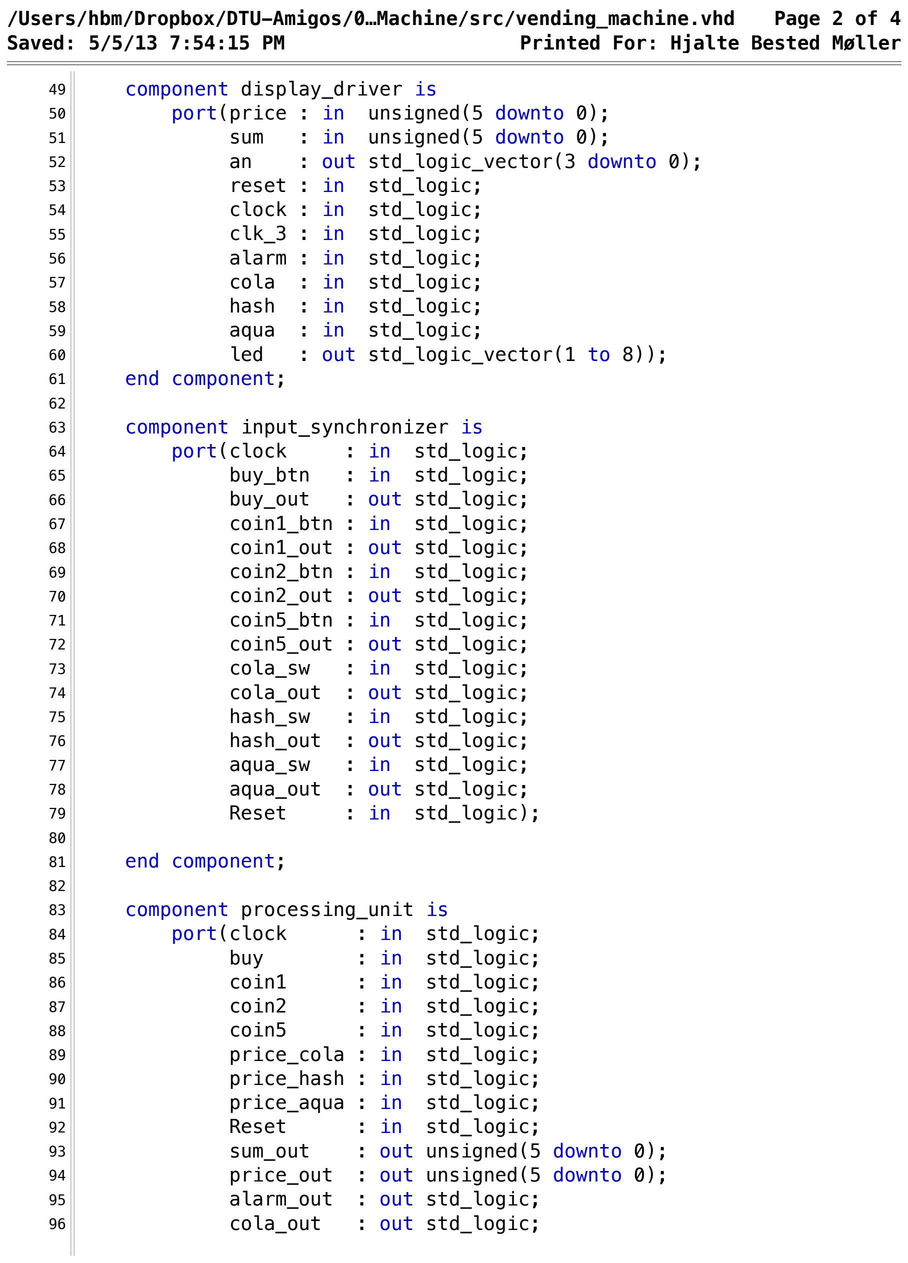
Figure 22: Page 3



/Users/hbm/Dropbox/DTU–Amigos/0…Machine/src/vending_machine.vhd   Page 1 of 4
Saved: 5/5/13 7:54:15 PM   Printed For: Hjalte Bested Møller

```vhdl
------------------------------------------------------------------------
-- 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.
------------------------------------------------------------------------

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

entity vending_machine is
    port(clk_50    : in  std_logic;
         clk_man   : in  std_logic;
         sel_man   : in  std_logic;
         reset     : in  std_logic;
         coin1_btn : in  std_logic;
         coin2_btn : in  std_logic;
         coin5_btn : in  std_logic;
         buy_btn   : in  std_logic;
         cola_sw   : in  std_logic;
         hash_sw   : in  std_logic;
         aqua_sw   : in  std_logic;
         an        : out std_logic_vector(3 downto 0);
         led       : out std_logic_vector(1 to 8));
end vending_machine;

architecture struct of vending_machine is
    signal clk                                 : std_logic;
    signal clk_3                               : std_logic;
    signal buy                                 : std_logic;
    signal coin1, coin2, coin5                 : std_logic;
    signal price_cola, price_hash, price_aqua : std_logic;
    signal alarm, cola, aqua, hash             : std_logic;
    signal sum, price                          : unsigned(5 downto 0);

    --------------------------
    -- Component declarations
    --------------------------
    component clock_manager is
        port(clk_50  : in  std_logic;
             clk_man : in  std_logic;
             sel_man : in  std_logic;
             clk     : out std_logic;
             clk_3   : out std_logic);
    end component;

```

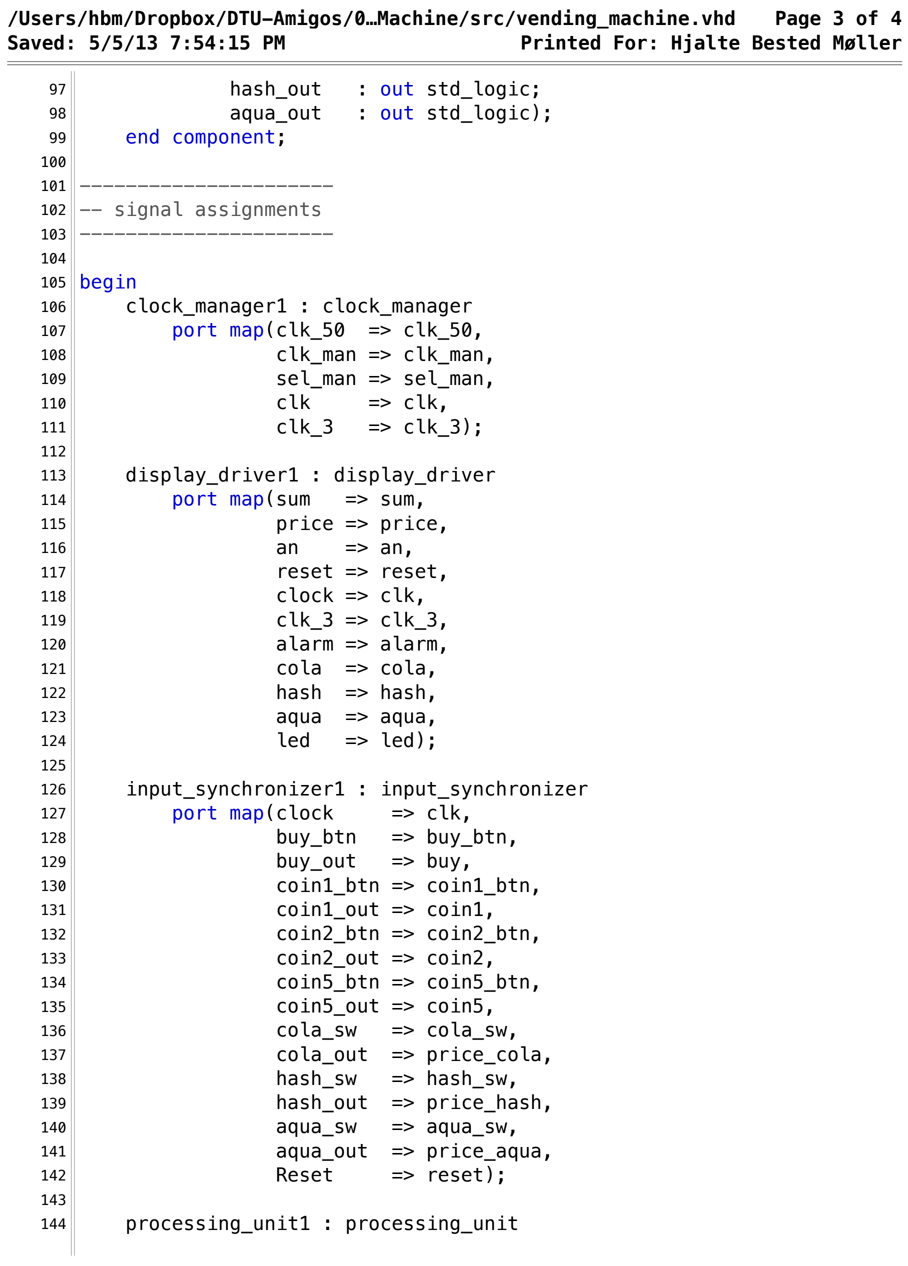
Figure 23: The complete vending machine, assembling all the components



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/vending_machine.vhd   Page 2 of 4
Saved: 5/5/13 7:54:15 PM   Printed For: Hjalte Bested Møller

```
    component display_driver is
        port(price : in  unsigned(5 downto 0);
             sum   : in  unsigned(5 downto 0);
             an    : out std_logic_vector(3 downto 0);
             reset : in  std_logic;
             clock : in  std_logic;
             clk_3 : in  std_logic;
             alarm : in  std_logic;
             cola  : in  std_logic;
             hash  : in  std_logic;
             aqua  : in  std_logic;
             led   : out std_logic_vector(1 to 8));
    end component;

    component input_synchronizer is
        port(clock     : in  std_logic;
             buy_btn   : in  std_logic;
             buy_out   : out std_logic;
             coin1_btn : in  std_logic;
             coin1_out : out std_logic;
             coin2_btn : in  std_logic;
             coin2_out : out std_logic;
             coin5_btn : in  std_logic;
             coin5_out : out std_logic;
             cola_sw   : in  std_logic;
             cola_out  : out std_logic;
             hash_sw   : in  std_logic;
             hash_out  : out std_logic;
             aqua_sw   : in  std_logic;
             aqua_out  : out std_logic;
             Reset     : in  std_logic);

    end component;

    component processing_unit is
        port(clock      : in  std_logic;
             buy        : in  std_logic;
             coin1      : in  std_logic;
             coin2      : in  std_logic;
             coin5      : in  std_logic;
             price_cola : in  std_logic;
             price_hash : in  std_logic;
             price_aqua : in  std_logic;
             Reset      : in  std_logic;
             sum_out    : out unsigned(5 downto 0);
             price_out  : out unsigned(5 downto 0);
             alarm_out  : out std_logic;
             cola_out   : out std_logic;
```

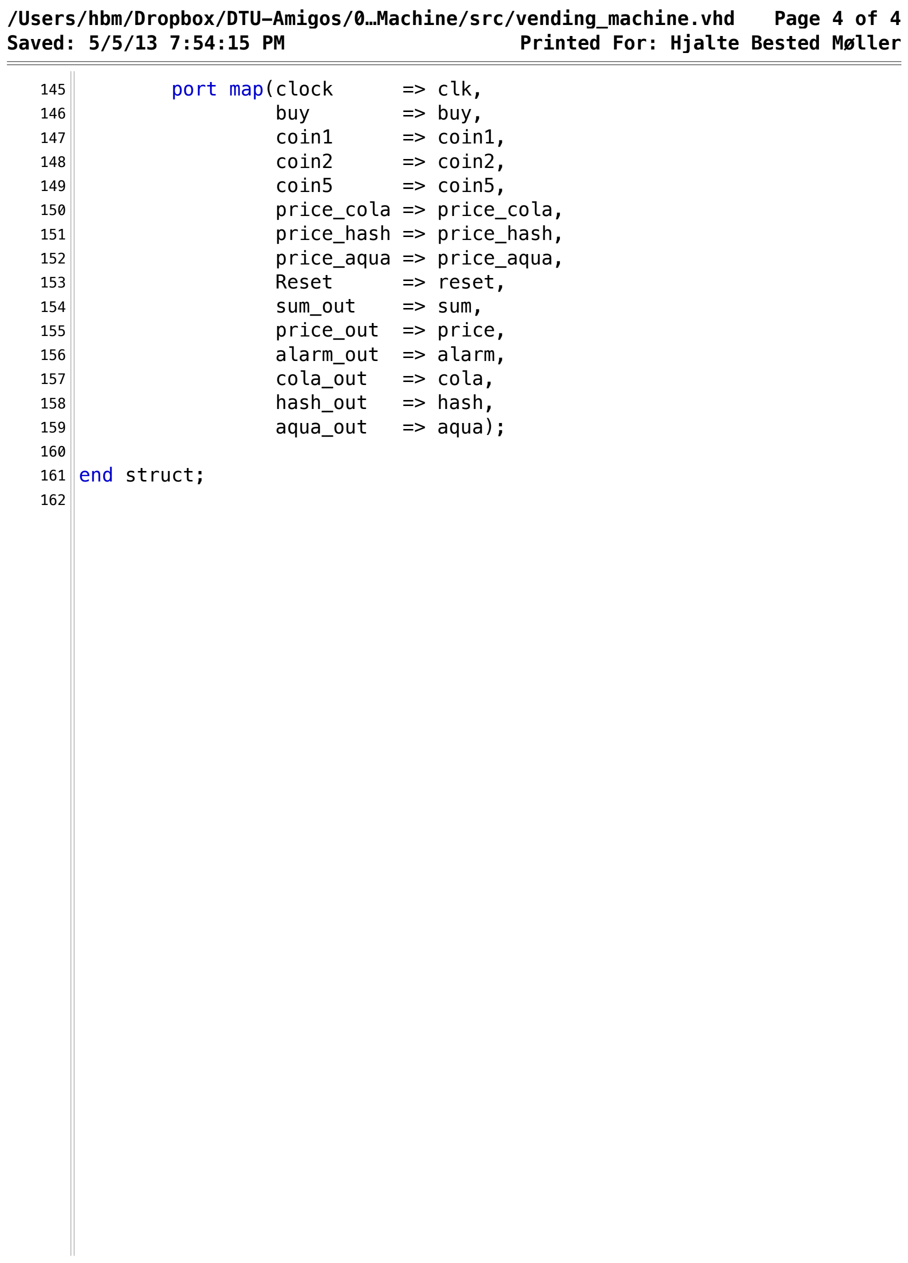
Figure 24: Page 2



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/vending_machine.vhd Page 3 of 4
Saved: 5/5/13 7:54:15 PM Printed For: Hjalte Bested Møller

```
             hash_out   : out std_logic;
             aqua_out   : out std_logic);
    end component;

----------------------
-- signal assignments
----------------------

begin
    clock_manager1 : clock_manager
        port map(clk_50  => clk_50,
                 clk_man => clk_man,
                 sel_man => sel_man,
                 clk     => clk,
                 clk_3   => clk_3);

    display_driver1 : display_driver
        port map(sum   => sum,
                 price => price,
                 an    => an,
                 reset => reset,
                 clock => clk,
                 clk_3 => clk_3,
                 alarm => alarm,
                 cola  => cola,
                 hash  => hash,
                 aqua  => aqua,
                 led   => led);

    input_synchronizer1 : input_synchronizer
        port map(clock     => clk,
                 buy_btn   => buy_btn,
                 buy_out   => buy,
                 coin1_btn => coin1_btn,
                 coin1_out => coin1,
                 coin2_btn => coin2_btn,
                 coin2_out => coin2,
                 coin5_btn => coin5_btn,
                 coin5_out => coin5,
                 cola_sw   => cola_sw,
                 cola_out  => price_cola,
                 hash_sw   => hash_sw,
                 hash_out  => price_hash,
                 aqua_sw   => aqua_sw,
                 aqua_out  => price_aqua,
                 Reset     => reset);

    processing_unit1 : processing_unit
```

Figure 25: Page 3



/Users/hbm/Dropbox/DTU-Amigos/0…Machine/src/vending_machine.vhd   Page 4 of 4
Saved: 5/5/13 7:54:15 PM   Printed For: Hjalte Bested Møller

```
        port map(clock      => clk,
                 buy        => buy,
                 coin1      => coin1,
                 coin2      => coin2,
                 coin5      => coin5,
                 price_cola => price_cola,
                 price_hash => price_hash,
                 price_aqua => price_aqua,
                 Reset      => reset,
                 sum_out    => sum,
                 price_out  => price,
                 alarm_out  => alarm,
                 cola_out   => cola,
                 hash_out   => hash,
                 aqua_out   => aqua);

end struct;

```

Figure 26: Page 4