Edge detection design project

02203 - Design of Digital Systems (Fall 2014)

Team 5
s918819 Nicolas Bøtkjær
s073188 Anders Greve

Technical University of Denmark
Department of Electrical Engineering

DTU Compute
Institut for Matematik og Computer Science

Abstract

02203 Design of Digital Systems - Edge detection Report d.
s918819 Nicolas Bøtkjær
s073188 Anders Greve

Contents

Chapter 1

Introduction

Husk at builde 2 gange for at referencer og sidetal opdaterer ellers er resultatet ??

Chapter 2

Theory

2.1 HW

PUT some text
put some cite [1, p.11 eq.2.6], [2, p.11 eq.2.6]

2.2 MEM

Put some picture figure 2.1

Figure 2.1: A picture of Bagsværd Sø

A picture more figure 2.2 a reference to section 2.1

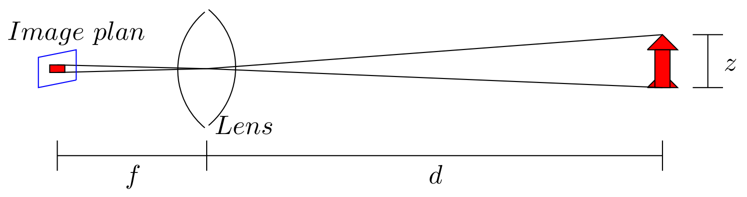

Figure 2.2: Some focal length stof a tikz picture (we only like because of vector graphic)

Chapter 3

Results

3.1 A

put some text

3.2 B

put some text

Chapter 4

Conclusion

put some text
And some more text
and some more text

Bibliography

[1] Pong P. Chu *RTL HARDWARE DESIGN USING VHDL* 2006 Wiley

[2] Jens Sparsø *Edge detector project description* 2014 Note

Appendix

A1 gcd.vhdl

```vhdl
--
-----------------------------------------------------------------------------
--
--  Title       : Finite state machine and datapath of the GCD
--              :
--  Developers : Anders Greve(s073188) and Nicolas Bøtkjær (s918819)
--              :
--  Purpose     : This design is the FSM and Datapath of the Greatest Common
    Divisor
--              :
--  Notes       : Implementation of Euclids GCD algorithm with repeated
    subtraction.
--              : Basic implemention without any optimization.
--              :
--  Revision    :  02203 fall 2014 v.1
--
--
-----------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY gcd IS
    PORT (clk:      IN std_logic;             -- The clock signal.
          reset:    IN std_logic;             -- Reset the module.
          req:      IN std_logic;             -- Start computation.
          AB:       IN unsigned(7 downto 0);  -- The two operands.
          ack:      OUT std_logic;            -- Computation is complete.
          C:        OUT unsigned(7 downto 0));  -- The result.
END gcd;

architecture FSMD of gcd is
-- FSMD States
type state_type is ( InputA, LoadA, RegAdone, InputB, LoadB, CmpAB, UpdateA,
    UpdateB, DoneC );
-- Declare signals
signal reg_a,next_reg_a,next_reg_b,reg_b : unsigned(7 downto 0);
signal state, next_state : state_type;

begin
   -- Combinatoriel logic
   CL: process (req,AB,state,reg_a,reg_b,reset)
   begin
      next_reg_a <= reg_a;
      next_reg_b <= reg_b;
      ack <= '0';
      C <= (others =>'Z');

      case (state) is
         When InputA =>
            if req = '1' then
               next_state <= LoadA;
            else
               next_state <= InputA;
            end if;

         When LoadA =>
            next_state <= RegAdone;
            next_reg_a <= AB;

         When RegAdone =>
            ack <= '1';
            if req = '0' then
               next_state <= InputB;
            else
               next_state <= RegAdone;
            end if;

         When InputB =>
            if req = '1' then
               next_state <= LoadB;
            else
               next_state <= InputB;
            end if;

         When LoadB=>
            next_reg_b <= AB;
            next_state <= CmpAB;

         When CmpAB =>
            if reg_a = reg_b then
               next_state <= DoneC;
            elsif reg_a > reg_b then
               next_state <= UpdateA;
            else
               next_state <= UpdateB; -- A < B
            end if;

         When UpdateA =>
            next_reg_a <= reg_a - reg_b;
            next_state <= CmpAB;

         When UpdateB =>
            next_reg_b <= reg_b - reg_a;
            next_state <= CmpAB;

         When DoneC =>
```

```
            C <= reg_a;
            ack <= '1';
            if req = '0' then
               next_state <= InputA;
            else
               next_state <= DoneC;
            end if;

      end case;
   end process CL;

   -- Registers
   seq: process (clk, reset)
   begin
      if reset = '1' then
         state <= InputA;          -- Reset to initial state
      elsif rising_edge(clk) then
         -- Update all registers
         state <= next_state;
         reg_a <= next_reg_a;
         reg_b <= next_reg_b;
      end if;
   end process seq;

end fsmd;
```

A2.1 gcd_res_shr.vhdl

```vhdl
--
-----------------------------------------------------------------------------------------
--
--  Title       : Finite state machine and datapath of the GCD
--              :
--  Developers : Anders Greve(s073188) and Nicolas Bøtkjær (s918819)
--              :
--  Purpose     : This design is the FSM and Datapath of the Greatest Common
    Divisor
--              :
--  Notes       : Implementation of Euclids GCD algorithm with repeated
    subtraction.
--              : Operator sharing is implementetd for both subtraction
--              : and multiplexing.
--              :
--  Revision    :  02203 fall 2014 v.1
--
--
-----------------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

ENTITY gcd IS
    PORT (clk:      IN std_logic;           -- The clock signal.
           reset:   IN std_logic;           -- Reset the module.
           req:     IN std_logic;           -- Start computation.
           AB:      IN unsigned(7 downto 0); -- The two operands.
           ack:     OUT std_logic;          -- Computation is complete.
           C:       OUT unsigned(7 downto 0)); -- The result.
END gcd;

architecture FSMD_res_sharing of gcd is
-- FSMD States
type state_type is ( InputA, LoadA, RegAdone, InputB, LoadB, CmpAB, UpdateA,
    UpdateB, DoneC );
-- Declare signals
signal state, next_state : state_type;
signal reg_a,next_reg_a,next_reg_b,reg_b : unsigned(7 downto 0);
signal op1, op2, Y : signed(8 downto 0); -- One extra bit to hold the sign-bit
    .
```

```
signal C_int : unsigned (7 downto 0);
signal ABorALU : std_logic;

begin
   -- Share the subtraction
   Y <= op1 - op2;
   -- Share the multiplexer (AB input or result from subtraction)
   C_int <= unsigned(Y(7 downto 0)) when ABorALU = '0' else AB ;

   -- Combinatoriel logic
   CL: process (req, AB, state, reg_a, reg_b, C_int, Y, reset)
   begin
      -- Default values.
      C <= (others =>'Z');
      ABorALU <= '0';
      next_reg_a <= reg_a;
      next_reg_b <= reg_b;
      ack <= '0';
      op1 <= signed('0' & std_logic_vector(reg_a));
      op2 <= signed('0' & std_logic_vector(reg_b));

      case (state) is

      When InputA =>
         if req = '1' then
            next_state <= LoadA;
         else
            next_state <= InputA;
         end if;

      When LoadA =>
         next_state <= RegAdone;
         next_reg_a <=  C_int;
         ABorALU <= '1';

      When RegAdone =>
         ack <= '1';
         if req = '0' then
            next_state <= InputB;
         else
            next_state <= RegAdone;
         end if;

      When InputB =>
         if req = '1' then
            next_state <= LoadB;
         else
            next_state <= InputB;
         end if;

      When LoadB=>
         next_state <= CmpAB;
         next_reg_b <= C_int;
         ABorALU <= '1';

      When CmpAB =>
```

```
         if Y(8) = '1' then -- If sign bit is set op2 > op1
            next_state <= UpdateB;
         elsif Y(7 downto 0) = 0 then
            next_state <= DoneC;
         else
            next_state <= UpdateA;
         end if;

      When UpdateA =>
         op1 <= signed('0' & std_logic_vector(reg_a));
         op2 <= signed('0' & std_logic_vector(reg_b));
         next_reg_a <=  C_int;
         next_state <= CmpAB;

      When UpdateB =>
         op1 <= signed('0' & std_logic_vector(reg_b));
         op2 <= signed('0' & std_logic_vector(reg_a));
         next_reg_b <= C_int;
         next_state <= CmpAB;

      When DoneC =>
         ack <= '1';
         C <= reg_a;
         if req = '0' then
            next_state <= InputA;
         else
            next_state <= DoneC;
         end if;

      end case;
   end process CL;

   -- Registers
   seq: process (clk, reset)
   begin
      if reset = '1' then
         state <= InputA;          -- Reset to initial state
      elsif rising_edge(clk) then
         state <= next_state;
         reg_a <= next_reg_a;
         reg_b <= next_reg_b;
      end if;
   end process seq;

end FSMD_res_sharing;
```