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

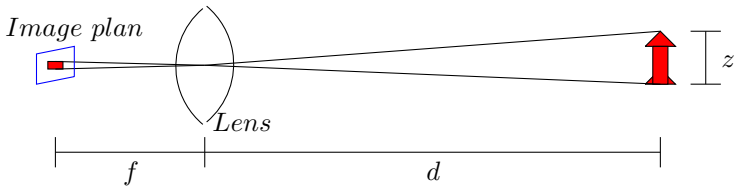

Figure 2.2: Some focal length stof a tikz picture (we only like because of vector graphic)

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

```
1  --
   ------------------------------------------------------------------------------
2  --
3  --  Title      : Finite state machine and datapath of the GCD
4  --             :
5  --  Developers : Anders Greve(s073188) and Nicolas Bøtkjær (s918819)
6  --             :
7  --  Purpose    : This design is the FSM and Datapath of the Greatest Common
   Divisor
8  --             :
9  --  Notes      : Implementation of Euclids GCD algorithm with repeated
   subtration.
10 --             : Basic implemention without any optimization.
11 --             :
12 --  Revision   :  02203 fall 2014 v.1
13 --
14 --
   ------------------------------------------------------------------------------

15
16 library IEEE;
17 use IEEE.STD_LOGIC_1164.ALL;
18 use IEEE.NUMERIC_STD.ALL;
19
20 ENTITY gcd IS
21     PORT (clk:      IN std_logic;             -- The clock signal.
22           reset:    IN std_logic;             -- Reset the module.
23           req:      IN std_logic;             -- Start computation.
24           AB:       IN unsigned(7 downto 0); -- The two operands.
25           ack:      OUT std_logic;            -- Computation is complete.
26           C:        OUT unsigned(7 downto 0)); -- The result.
27 END gcd;
28
29 architecture FSMD of gcd is
30 -- FSMD States
31 type state_type is ( InputA, LoadA, RegAdone, InputB, LoadB, CmpAB, UpdateA,
       UpdateB, DoneC );
32 -- Declare signals
33 signal reg_a,next_reg_a,next_reg_b,reg_b : unsigned(7 downto 0);
34 signal state, next_state : state_type;
35
36 begin
37    -- Combinatoriel logic
```

```
38    CL: process (req,AB,state,reg_a,reg_b,reset)
39    begin
40       next_reg_a <= reg_a;
41       next_reg_b <= reg_b;
42       ack <= '0';
43       C <= (others =>'Z');
44
45       case (state) is
46          When InputA =>
47             if req = '1' then
48                next_state <= LoadA;
49             else
50                next_state <= InputA;
51             end if;
52
53          When LoadA =>
54             next_state <= RegAdone;
55             next_reg_a <= AB;
56
57          When RegAdone =>
58             ack <= '1';
59             if req = '0' then
60                next_state <= InputB;
61             else
62                next_state <= RegAdone;
63             end if;
64
65          When InputB =>
66             if req = '1' then
67                next_state <= LoadB;
68             else
69                next_state <= InputB;
70             end if;
71
72          When LoadB=>
73             next_reg_b <= AB;
74             next_state <= CmpAB;
75
76          When CmpAB =>
77             if reg_a = reg_b then
78                next_state <= DoneC;
79             elsif reg_a > reg_b then
80                next_state <= UpdateA;
81             else
82                next_state <= UpdateB; -- A < B
83             end if;
84
85          When UpdateA =>
86             next_reg_a <= reg_a - reg_b;
87             next_state <= CmpAB;
88
89          When UpdateB =>
90             next_reg_b <= reg_b - reg_a;
91             next_state <= CmpAB;
92
93          When DoneC =>
```

```
94              C <= reg_a;
95              ack <= '1';
96              if req = '0' then
97                  next_state <= InputA;
98              else
99                  next_state <= DoneC;
100             end if;
101
102         end case;
103     end process CL;
104
105     -- Registers
106     seq: process (clk, reset)
107     begin
108         if reset = '1' then
109             state <= InputA;          -- Reset to initial state
110         elsif rising_edge(clk) then
111             -- Update all registers
112             state <= next_state;
113             reg_a <= next_reg_a;
114             reg_b <= next_reg_b;
115         end if;
116     end process seq;
117
118 end fsmd;
```

A2.1 gcd_res_shr.vhdl

```
1  --
   ------------------------------------------------------------------------------
2  --
3  --  Title      : Finite state machine and datapath of the GCD
4  --             :
5  --  Developers : Anders Greve(s073188) and Nicolas Bøtkjær (s918819)
6  --             :
7  --  Purpose    : This design is the FSM and Datapath of the Greatest Common
       Divisor
8  --             :
9  --  Notes      : Implementation of Euclids GCD algorithm with repeated
       subtration.
10 --             : Operator sharing is implementetd for both subtraction
11 --             : and multiplexing.
12 --             :
13 --  Revision   :  02203 fall 2014 v.1
14 --
15 --
   ------------------------------------------------------------------------------

16
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.ALL;
19 use IEEE.NUMERIC_STD.ALL;
20
21 ENTITY gcd IS
22     PORT (clk:      IN std_logic;             -- The clock signal.
23           reset:    IN std_logic;             -- Reset the module.
24           req:      IN std_logic;             -- Start computation.
25           AB:       IN unsigned(7 downto 0); -- The two operands.
26           ack:      OUT std_logic;            -- Computation is complete.
27           C:        OUT unsigned(7 downto 0)); -- The result.
28 END gcd;
29
30 architecture FSMD_res_sharing of gcd is
31 -- FSMD States
32 type state_type is ( InputA, LoadA, RegAdone, InputB, LoadB, CmpAB, UpdateA,
       UpdateB, DoneC );
33 -- Declare signals
34 signal state, next_state : state_type;
35 signal reg_a,next_reg_a,next_reg_b,reg_b : unsigned(7 downto 0);
36 signal op1, op2, Y : signed(8 downto 0); -- One extra bit to hold the sign-bit
       .
```

```
37 signal C_int : unsigned (7 downto 0);
38 signal ABorALU : std_logic;
39
40 begin
41    -- Share the subtraction
42    Y <= op1 - op2;
43    -- Share the multiplexer (AB input or result from subtraction)
44    C_int <= unsigned(Y(7 downto 0)) when ABorALU = '0' else AB ;
45
46    -- Combinatoriel logic
47    CL: process (req, AB, state, reg_a, reg_b, C_int, Y, reset)
48    begin
49       -- Default values.
50       C <= (others =>'Z');
51       ABorALU <= '0';
52       next_reg_a <= reg_a;
53       next_reg_b <= reg_b;
54       ack <= '0';
55       op1 <= signed('0' & std_logic_vector(reg_a));
56       op2 <= signed('0' & std_logic_vector(reg_b));
57
58       case (state) is
59
60       When InputA =>
61          if req = '1' then
62             next_state <= LoadA;
63          else
64             next_state <= InputA;
65          end if;
66
67       When LoadA =>
68          next_state <= RegAdone;
69          next_reg_a <=  C_int;
70          ABorALU <= '1';
71
72       When RegAdone =>
73          ack <= '1';
74          if req = '0' then
75             next_state <= InputB;
76          else
77             next_state <= RegAdone;
78          end if;
79
80       When InputB =>
81          if req = '1' then
82             next_state <= LoadB;
83          else
84             next_state <= InputB;
85          end if;
86
87       When LoadB=>
88          next_state <= CmpAB;
89          next_reg_b <= C_int;
90          ABorALU <= '1';
91
92       When CmpAB =>
```

```
93            if Y(8) = '1' then -- If sign bit is set op2 > op1
94               next_state <= UpdateB;
95            elsif Y(7 downto 0) = 0 then
96               next_state <= DoneC;
97            else
98               next_state <= UpdateA;
99            end if;
100
101         When UpdateA =>
102            op1 <= signed('0' & std_logic_vector(reg_a));
103            op2 <= signed('0' & std_logic_vector(reg_b));
104            next_reg_a <=  C_int;
105            next_state <= CmpAB;
106
107         When UpdateB =>
108            op1 <= signed('0' & std_logic_vector(reg_b));
109            op2 <= signed('0' & std_logic_vector(reg_a));
110            next_reg_b <= C_int;
111            next_state <= CmpAB;
112
113         When DoneC =>
114            ack <= '1';
115            C <= reg_a;
116            if req = '0' then
117               next_state <= InputA;
118            else
119               next_state <= DoneC;
120            end if;
121
122         end case;
123      end process CL;
124
125      -- Registers
126      seq: process (clk, reset)
127      begin
128         if reset = '1' then
129            state <= InputA;          -- Reset to initial state
130         elsif rising_edge(clk) then
131            state <= next_state;
132            reg_a <= next_reg_a;
133            reg_b <= next_reg_b;
134         end if;
135      end process seq;
136
137   end FSMD_res_sharing;
```