Lab 6 : Greatest common divisor

Name: Sait Sarper Özaslan

Section: 102-2

ID: 22002861

Date: 28/11/2022

Purpose: The purpose of this lab is to use either combinational circuit or a FSM machine integrated into BASYS3 to find the greatest common divisor of two 8-bit numbers.

Questions and Answers:

- What was your algorithm to calculate the GCD?

I used the Euclidian theorem to calculate the Greatest common divisor. To give an example from the wanted values of 140 and 12 from the lab sheet. I can subtract 12 from 140 and repeat this result until the result of the operation is smaller is less than 12. Then I swap the bigger number and smaller number's places and repeat this procedure until the result of the operation is equal to subtractor, when this condition is met the result will be the greatest common divisor. As an example,

140 – 12 = 128

128 - 12= 116 …   ... 20 - 12 = 8, 12 - 8= 4, 8 – 4 = 4 -----> 4 is the GCD.

- Is your module a combinational circuit or an FSM? If the latter, is it a Moore machine or a Mealy machine? Would it be cheaper to implement GCD as a combinational circuit or as an FSM? Would it be faster? What are the drawbacks in each case?

My module is mainly a Moore FSM. It can be said that a combinational circuit would be faster if the designing process that comes through two 8-bit numbers' truth table is disregarded. In other words, although it will take a considerable amount of time to design the combinational circuit, it will work faster than the FSM. As combinational circuits are faster for arithmetic operations, I used them in my arithmetic operations.

- How many clock cycles did it take in your simulation to calculate the GCD? Do you think this can be optimized? How so?

It took 28 clock ticks to find the GCD. By changing the computation method from the Euclidian theorem it is possible to find a shorter method that finds the greatest common divisor. However writing the code of these were harder in general so I refrained from writing them.

Methodology:
To find the greatest common divisor of two numbers I used the Euclidian theorem.

To implement this theorem into the BASYS3 I decided to use a 3-state machine that consists of hold, compare(swap), subtract. Essentially, I first register these two numbers to the state machine, then switch to swap state which compares the numbers with each other and swap numbers if the bigger number is the subtractor. After the swap state I move into the subtract state which subtracts the bigger number from the smaller number. After each subtraction the result goes into the swap stage again to check which of the numbers are bigger. This process goes back and forth until the result is the same as the subtractor.

Design Specifications:
I wrote the code in a modular fashion where I separated the comparator and subtractor functions as a sub module under the greatest common divisor. I used the comparator and subtractor from the arithmetic logic unit and expanded them into 8 bit version. The main module gcd_main.vhdl consists of 5 inputs and 2 outputs.

- Clock(in): 100 MHz base clock of BASYS3
- Reset(in): The rest function to load 0 to the A and B
- A(in): A 8-bit number to find the greatest common divisor.
- B(in): An 8-bit number to find the greatest common divisor.
- Start(in): İmplemented to the BASYS 3 function runs the whole function when pressed.
- Ready(out): The ready is 1 when the machine is in hold state. Implemented into LEDs.
- Outp(out): An 8-bit number whose output is the greatest common divisor. Implemented into LEDS

This main module also consists of 3 states: Hold, Swap, Subtract

- Hold: This state is the state where numbers are loaded and this state remains active when two of the numbers are equal
- Swap: This is the state where numbers are compared to each other and swaps the numbers so that the bigger number will be the subtracted and the smaller number will be the subtractor. This state will then switch to subtract state. I use comparator module in this state
- Subtract: This is the state where two numbers are subtracted from each other. Then the result of this operation and subtracted will move onto the swap state. I use the subtractor module in this state.

To implement comparator, I used the function I had written in the Arithmetic Logic Unit. When the numbers are equal the result (1) gets into the function and the function moves onto the hold state. When the result of the subtraction is bigger than the subtracted number result(0) gets into the function and the state changes into subtract. If it's smaller result(2) will get implement into the function which will swap the numbers and move to the subtractor state.

To implement subtractor, I used the function of subtraction in it's combinational circuit design of full adders.

Results:

The first thing I needed to check before implementing this function into BASYS3 was to check the code I had written in simulation.

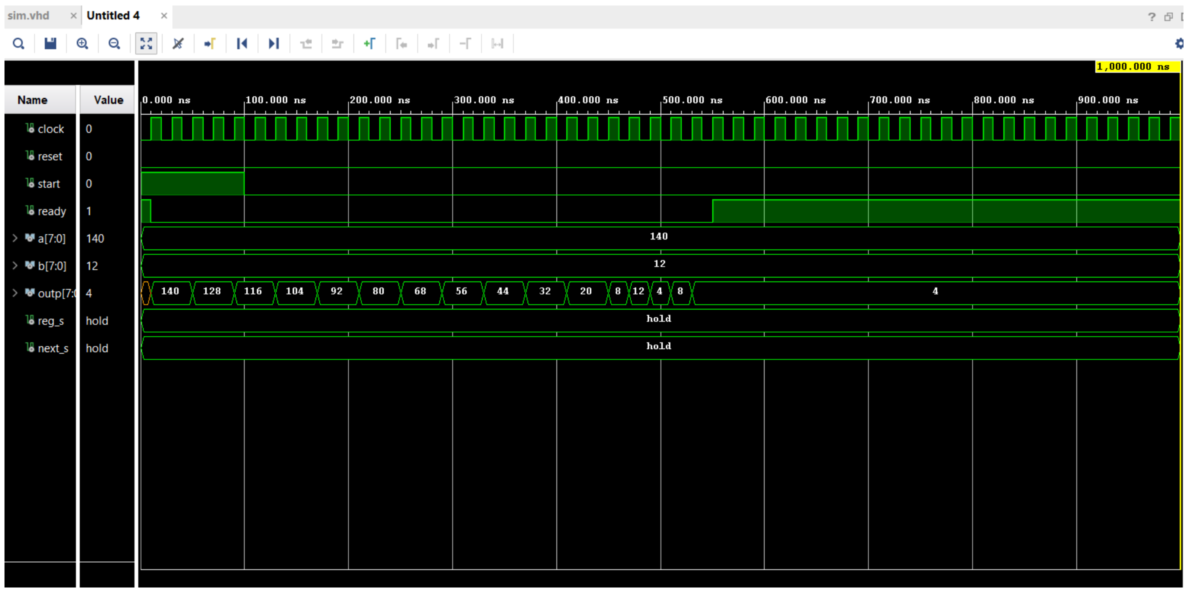

Figure 1: Simulation of GCD

After realizing that the simulation works I implemented the code into the BASYS 3 and tried a few examples. Here is the schematic first then some examples.

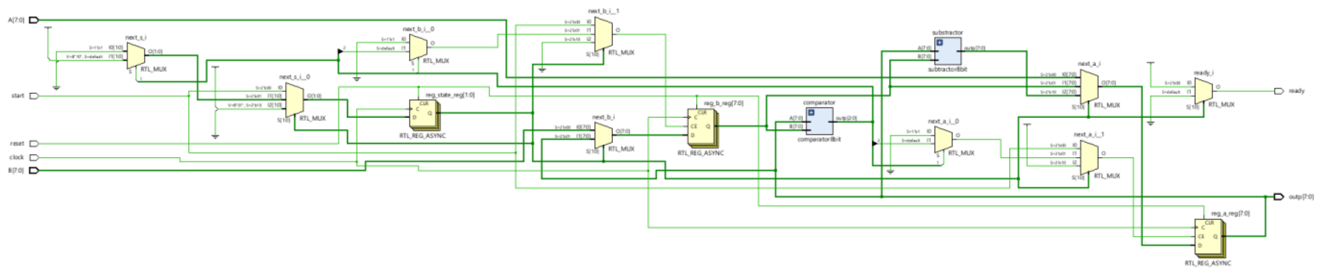

Figure 2: RTL Schematic of GCD

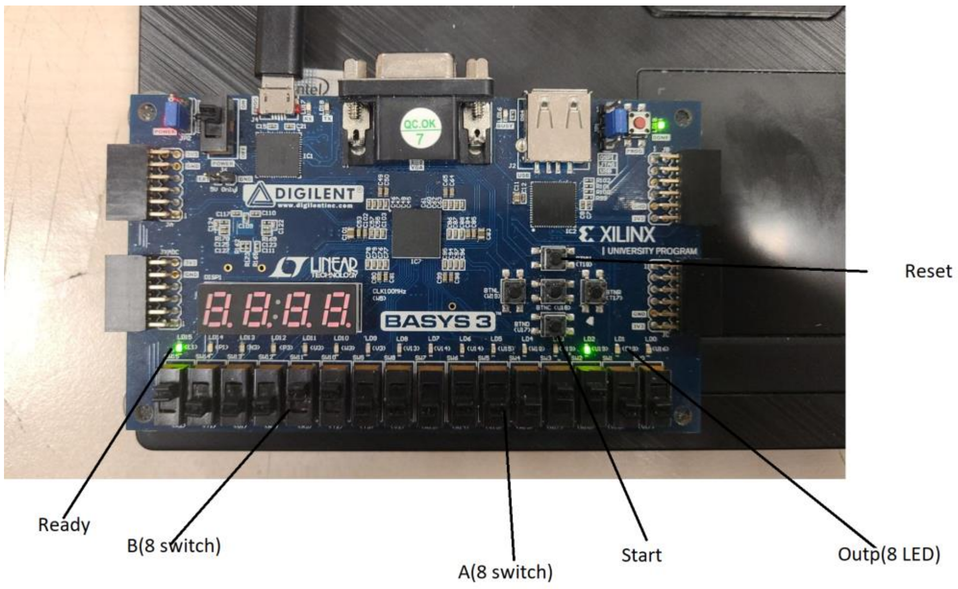

Figure 3: Constraint of GCD

Example 1: GCD of 140 and 12

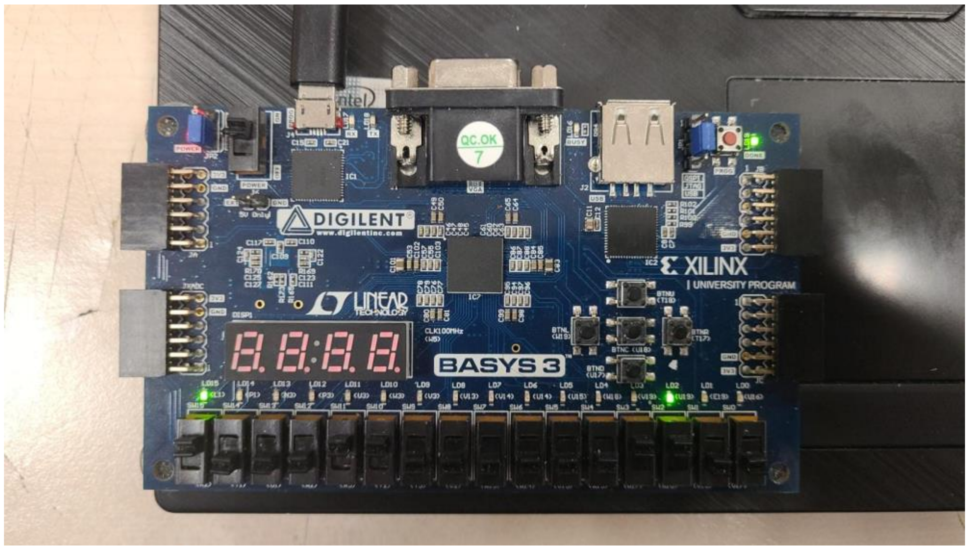

Figure 4: Case 140 and 12

Reset: 0

Ready: 1

A: 00001100 (12)

B: 10001100(140)

Outp: 00000100(4)

Example 2: GCD of 28 and 7

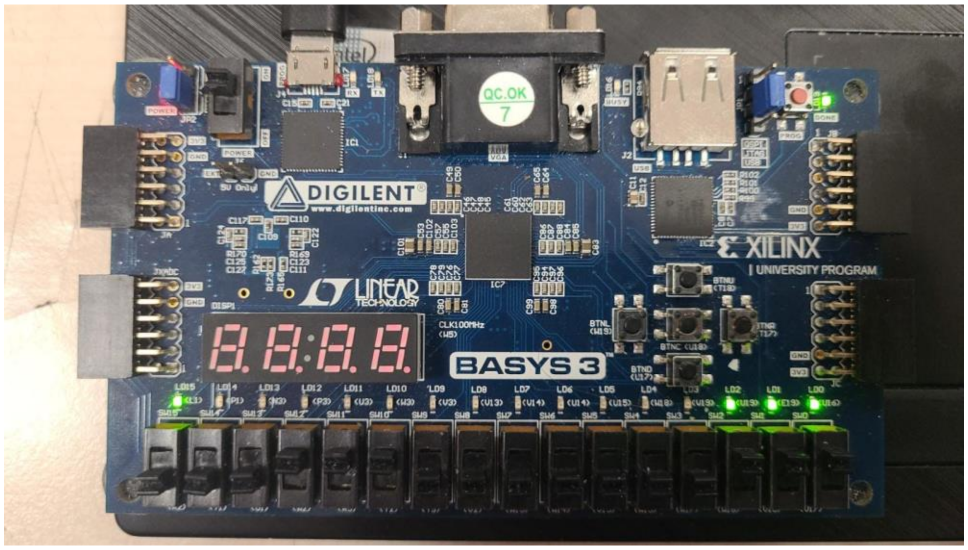

Figure 5: Case 28 and 7

Reset: 0

Ready: 1

A: 00000111(7)

B: 00011100 (28)

Outp: 00000111

In both cases the values the LED outp displayed aligned, after I pressed the start button, it's greatest common divisor which meant that the Function I had written was working appropriately.

Conclusion: Through this lab I figured out how to implement FSM as a VHDL code to the BASYS3 and find the greatest common divisor of two 8-bit numbers. One of the challenges I had during this lab was to figure out how I was going to switch between states and when I was going to remain in the state. Figuring out the comparator function and subtract function was quite easy as I have done the same two functions during Lab 4 in which I used an Arithmetic Logic Unit.

As I didn’t use the package from the IEEE library and it was more simpler to write to code in a modular fashion I had to resort to a lot of signal usages throughout this lab which sometimes did get confusing to understand whether I need a signal or not to assign a value to a variable. However, it is still more reasonable to write this code in modular way as one big chunk of VHDL code would be quite confusing even when it’s not complex.

References:

http://quitoart.blogspot.com/2017/11/vhdl-greatest-common-divisor-gcd.html

https://en.wikipedia.org/wiki/Euclidean_algorithm#:~:text=In%20mathematics%2C%20the%20Euclidean%20algorithm,them%20both%20without%20a%20remainder.

Appendixes:

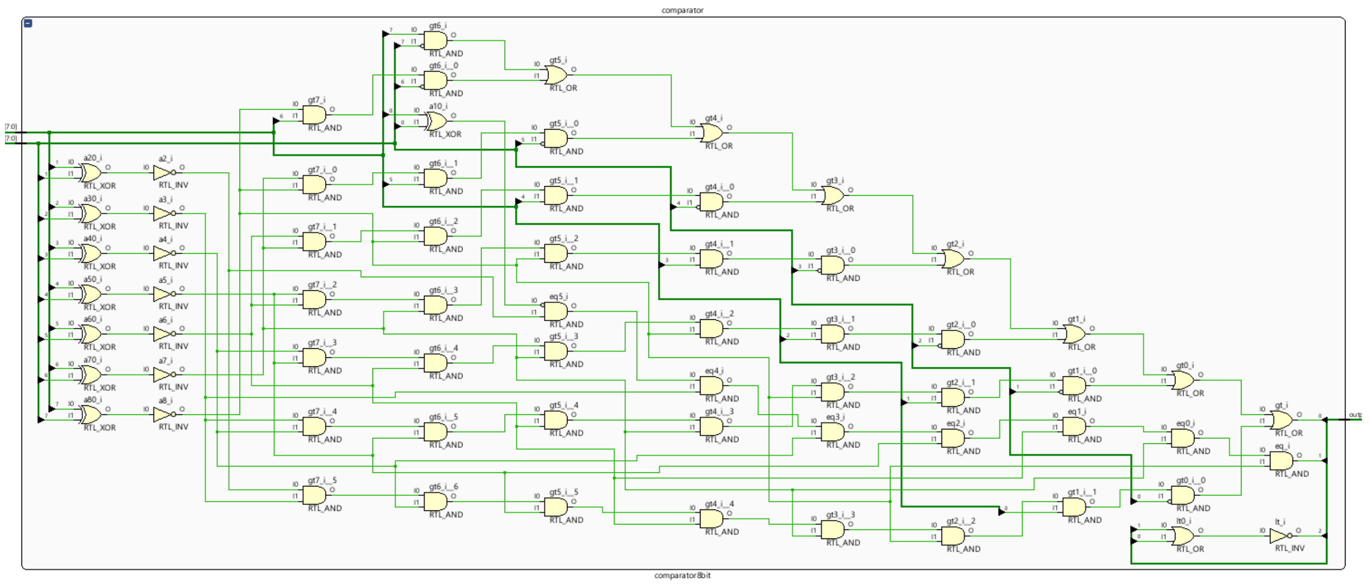

Figure 6: RTL Schematic of Comparator

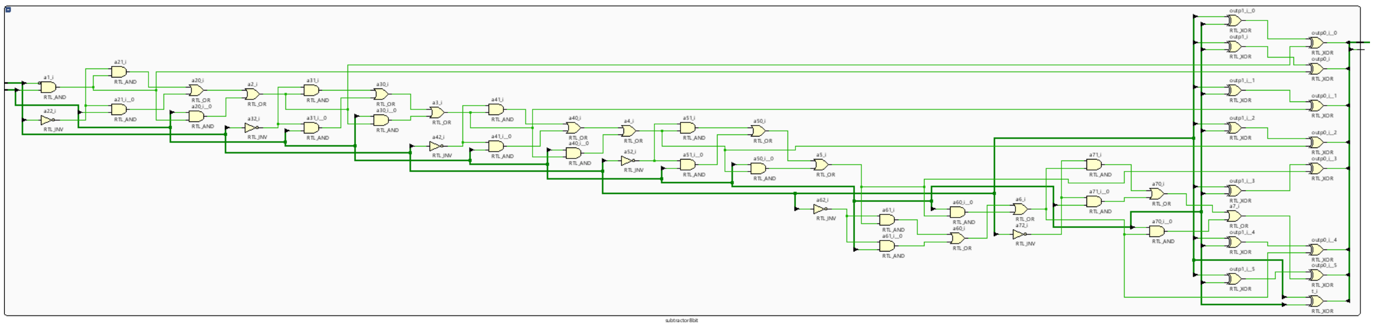
Figure 7: RTL Schematic of Subtractor

GCD.VHDL

```
library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

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

entity gcd_main is

port(

clock, reset: in std_logic;

start: in std_logic;

A, B: in std_logic_vector(7 downto 0);

ready: out std_logic;

outp: out std_logic_vector(7 downto 0)

);

end gcd_main ;

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

architecture gcdmain of gcd_main is

type state_type is (hold, swap, substract);

signal reg_state, next_s : state_type;


signal reg_a, reg_b, next_a, next_b: unsigned(7 downto 0);

signal compare_a,compare_b : unsigned(7 downto 0);
signal sub_b, sub_a, sub_res: std_logic_vector(7 downto 0);
signal compare_res: std_logic_vector(2 downto 0);

begin
------------------------------------------------------------
comparator: entity work.comparator8bit(rtlcomp)
port map(A => compare_a, B=> compare_b, outp=> compare_res);
substractor: entity work.subtractor8bit(rtlsubt)
port map(A => sub_a, B => sub_b, outp => sub_res);
------------------------------------------------------------
process(clock,reset)
begin
if reset='1' then
reg_state <= hold;
reg_a <= (others=>'0');
reg_b <= (others=>'0');
elsif (rising_edge(clock)) then
reg_state <= next_s;
reg_a <= next_a;
reg_b <= next_b;
end if;
end process;
------------------------------------------------------------
process(reg_state,reg_a,reg_b,start,A,B)
begin
next_a <= reg_a;
```

```
next_b <= reg_b;

compare_a <= reg_a;

compare_b <= reg_b;

sub_a <= std_logic_vector(reg_a);

sub_b <= std_logic_vector(reg_b);


case reg_state is

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

when hold =>

if start = '1' then

next_a <= unsigned(A);

next_b <= unsigned(B);

next_s <= swap;

else

next_s <= hold;

end if;

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

when swap =>

if (compare_res(1) = '1') then

next_s <= hold;


else

if(compare_res(2) = '1') then

next_a <= reg_b;

next_b <= reg_a;

end if;

next_s <= substract;
```

```
end if;

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

when substract =>

next_a <= unsigned(sub_res);

next_s <= swap;

end case;

end process;

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

ready <= '1' when reg_state = hold else '0';

  outp <= std_logic_vector(reg_a);

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

end gcdmain;
```

Comparator8bit.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use ieee.numeric_std.all;



entity comparator8bit is

__Port ( A : in unsigned(7 downto 0);

_____B : in unsigned(7 downto 0);

_____outp : out STD_LOGIC_VECTOR (2 downto 0));

end comparator8bit;


architecture rtlcomp of comparator8bit is
```

```
signal a1, a2,a3,a4,a5,a6,a7,a8,gt,lt,eq: std_logic ;

signal result: std_logic_vector (2 downto 0) ;

begin

a1 <= not (a(0) xor b(0));

a2 <= not (a(1) xor b(1));

a3 <= not (a(2) xor b(2));

a4 <= not (a(3) xor b(3));

a5 <= not (a(4) xor b(4));

a6 <= not (a(5) xor b(5));

a7 <= not (a(6) xor b(6));

a8 <= not (a(7) xor b(7));

eq <= a1 and a2 and a3 and a4 and a5 and a6 and a7 and a8;

gt <=(a(7) and (not b(7))) or

__(a8 and a(6) and (not b(6))) or

__(a7 and a8 and a(5) and (not b(5))) or

__(a6 and a7 and a8 and a(4) and (not b(4))) or

__(a5 and a6 and a7 and a8 and a(3) and (not b(3))) or

__(a4 and a5 and a6 and a7 and a8 and a(2) and (not b(2))) or

__(a3 and a4 and a5 and a6 and a7 and a8 and a(1) and (not b(1))) or

__(a2 and a3 and a4 and a5 and a6 and a7 and a8 and a(0) and (not b(0)));


lt <= not(eq or gt);

result (1) <= eq;

result (0) <= gt;

result (2) <= lt;

outp <= result;
```

```
end rtlcomp;
```

Subtractor8bit.vhd

```
library ieee;
use ieee.std_logic_1164.all;
entity subtractor8bit is
port (A, B: IN STD_LOGIC_VECTOR ( 7 DOWNTO 0);
outp : OUT STD_LOGIC_VECTOR ( 7 DOWNTO 0);
t: out std_logic
);
end subtractor8bit;
architecture rtlsubt of subtractor8bit is
signal a1,a2,a3,a4,a5,a6,a7,a8: std_logic;
BEGIN
t <= A(0) XOR B(0) xor '0';
outp(0) <= A(0) XOR B(0) xor '0';
a1 <= (not(A(0)) and '0') or (not(A(0)) and B(0)) or (B(0) and '0');
outp(1) <= A(1) XOR B(1) xor a1;
a2 <= (not(A(1)) and a1) or (not(A(1)) and B(1)) or (B(1) and a1) ;
outp(2) <= A(2) XOR B(2) xor a2;
a3 <= (not(A(2)) and a2) or (not(A(2)) and B(2)) or (B(2) and a2);
outp(3) <= A(3) XOR B(3) xor a3;
a4 <= (not(A(3)) and a3) or (not(A(3)) and B(3)) or (B(3) and a3);
outp(4) <= A(4) XOR B(4) xor a4;
a5 <= (not(A(4)) and a4) or (not(A(4)) and B(4)) or (B(4) and a4);
outp(5) <= A(5) XOR B(5) xor a5;
a6 <= (not(A(5)) and a5) or (not(A(5)) and B(5)) or (B(5) and a5);
```

```
outp(6) <= A(6) XOR B(6) xor a6;

a7 <= (not(A(6)) and a6) or (not(A(6)) and B(6)) or (B(6) and a6);

outp(7) <= A(7) XOR B(7) xor a7;

END rtlsubt;
```

MainTB

```
entity mainTB is

end mainTB;

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

architecture rtl_tb of mainTB is

signal clock, reset,start,ready: std_logic;

signal a, b,outp : std_logic_vector (7 downto 0);

type state_type is (idle, swap, sub);

signal reg_s, next_s : state_type;

begin

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

dut: entity work.gcd(slow_arch)

port map(clock,reset,start,a,b,ready,outp);

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

clock_process: process begin

clock <= '0';

wait for 10ns;

clock <= '1';

wait for 10ns;

end process;

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

stim_process: process begin
```

```
start  <= '1';

a <= "10001100";

b <= "00001100";

reset <= '0';

wait for 100ns;

start <= '0';

wait;

end process;

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

end rtl_tb;

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

configuration rtl_tb of mainTB is

for rtl_tb

end for;

end rtl_tb;
```