Ihsan Dogramaci Bilkent University

EE-102 Lab 6 Report

Name: Emre Şaş

ID: 22102764

Date: 20.11.2023

Section: 2

Lab 6 : GCD (Greatest Common Divisor)

Questions:

1. **What was your algorithm to calculate the GCD?**
   a. The Euclidean algorithm is used (a method for finding the greatest common divisor of two numbers by iteratively applying the division remainder).
2. **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?**
   a. My module functions as a Finite State Machine (FSM), specifically adopting the Moore machine architecture due to the inherent connection between each step and the present state facilitated by the Euclidean algorithm. Comparing the implementation of the Greatest Common Divisor (GCD) as a combinational circuit versus an FSM, the former tends to be a more cost-effective choice. Combinational circuits are generally cheaper to implement and offer faster performance compared to FSMs, particularly when Moore machines are involved, as they introduce additional delays.While FSMs, including Moore machines, simplify the circuit design process, they don't always lead to the most economical or fastest solutions. The convenience in designing circuits with FSMs comes at the expense of increased delay and higher costs. The decision between a combinational circuit and an FSM for the GCD implementation requires a careful balance between design ease and optimization for cost and speed, taking into account the specific requirements and constraints of the application. In essence, while FSMs provide a streamlined design process, their drawbacks, such as increased delay and costs, should be weighed against the advantages to determine the most suitable approach for a given scenario.
3. **How many clock cycles did it take in your simulation to calculate the GCD? Do you think this can be optimized? How so?**
   a. The computation of the Greatest Common Divisor (GCD) for 140 and 12 required 296 clock cycles, highlighting the importance of minimizing clock-dependent processes for optimization purposes. In terms of speed, a Mealy machine would outpace a Moore machine, and furthermore, a combinational circuit stands out as the fastest option.

Methodology:

The approach used entails creating a Moore Machine with three discrete states: "hold," "replace," and "subtract," each of which has a particular purpose throughout execution. The function of the register will be explained in more depth in the section that follows with Design Specifications. The algorithm starts when the "initiate" button is pressed in tandem with the reset button being set to "0." Importantly, when the requirements are satisfied, the LED labeled as the "ready" status turns on.

The algorithm operates in a methodical three-step sequence: First, a comparison of the provided statistics is carried out. Second, the Greatest Common Divisor (GCD) is simply determined as the common value if the numbers turn out to be equal. Thirdly, the method performs a basic operation based on the Euclidean algorithm if the numbers are not equal. It does this by subtracting the smaller number from the bigger one until the equality requirement is met.

A key component of this approach is the Euclidean algorithm, which iteratively subtracts the smaller number from the bigger number until the two values equalize. This method systematically determines the GCD of two integers. This phase makes sure that the original numbers' common divisor is found quickly and iteratively. By using the Euclidean technique, the system's general architecture gains a level of mathematical rigor that makes it possible for the GCD to be determined methodically and accurately.

Design Specifications:

The system is designed with a comprehensive set of inputs and outputs, featuring five inputs and two outputs. The clock input is set at a stable 100 MHz, providing a synchronous timing reference for the entire system. The second input acts as a reset selection, allowing for the clearing of outputs when necessary. Meanwhile, the third and fourth inputs facilitate the selection of two 8-bit numbers. The "Initiate" input, serving as an enable signal, triggers the initiation of the algorithmic calculation process.

At the core of the system is the "gcd.vhd" module, which encapsulates the GCD algorithm implemented as a Moore machine. This machine is designed with three distinct states: "hold," "replace," and "subtract," each state playing a crucial role in executing the GCD calculation process.

Hold State: In this state, the system evaluates whether the two selected numbers are equal. If equality is confirmed, the system remains in the "hold" state. However, if the numbers are not equal, the system transitions to the "replace" state.

Replace State: This state handles scenarios where the second number is greater than the first. In such cases, a swap operation occurs between the first and second registers, and the system transitions to the "subtract" state. If the numbers are found to be equal during this process, the system reverts to the "hold" state.

Subtract State: The "subtract" state involves subtracting the first number from the second. Following this operation, the system transitions back to the "replace" state, where further evaluations can occur.

To facilitate the essential comparison and subtraction operations between two 8-bit numbers, the design employs two distinct modules. The "comp.vhd" module is responsible for comparing the two 8-bit inputs and producing a 3-bit result, indicating whether the numbers are equal, greater, or smaller. On the other hand, the "subt.vhd" module functions as a subtractor, taking two 8-bit numbers and employing 8 full subtractors to produce another 8-bit output.

Both the comparator and subtractor modules utilize an asynchronous clock to expedite the processing speed, contributing to the overall efficiency of the GCD calculation algorithm. Upon successful completion of the GCD calculation, the finite state machine (FSM) remains in the "hold" state. The system signals the completion of calculations through an associated LED designated as the "ready" output. Additionally, the calculated number is visually represented using LEDs, with each LED corresponding to one bit of the output, enhancing the system's transparency and user interface.

Results and Interpretations :

The test bench is created to determine the Greatest Common Divisor (GCD) of the given inputs: in_x = "10001100" (140) and in_y = "00001100" (12). The "initiate" signal is activated between 0ns and 100ns. The test bench produces an output result of "00000100" (4). The simulation outcome (Figure 1.1) and the rtl schmetaic(Figure 1.2) are presented in the figures below.

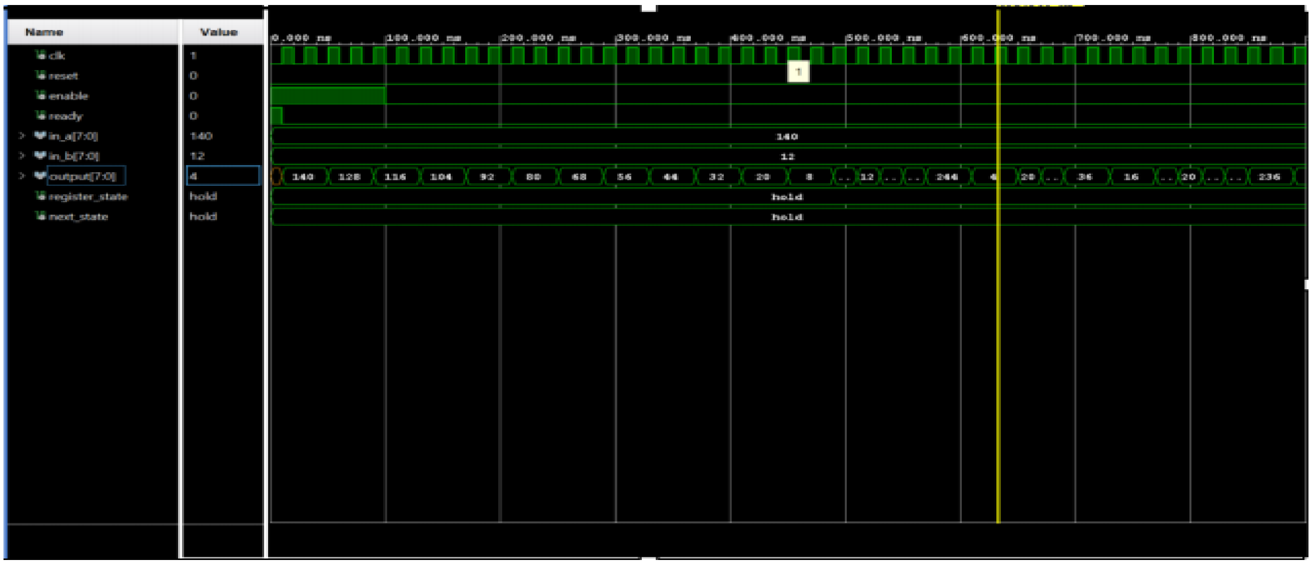


Figure 1.1 (Test bench is tested for the inputs first input "10001100" and second input "00001100". The enable is asserted between 0ns and 100ns.)

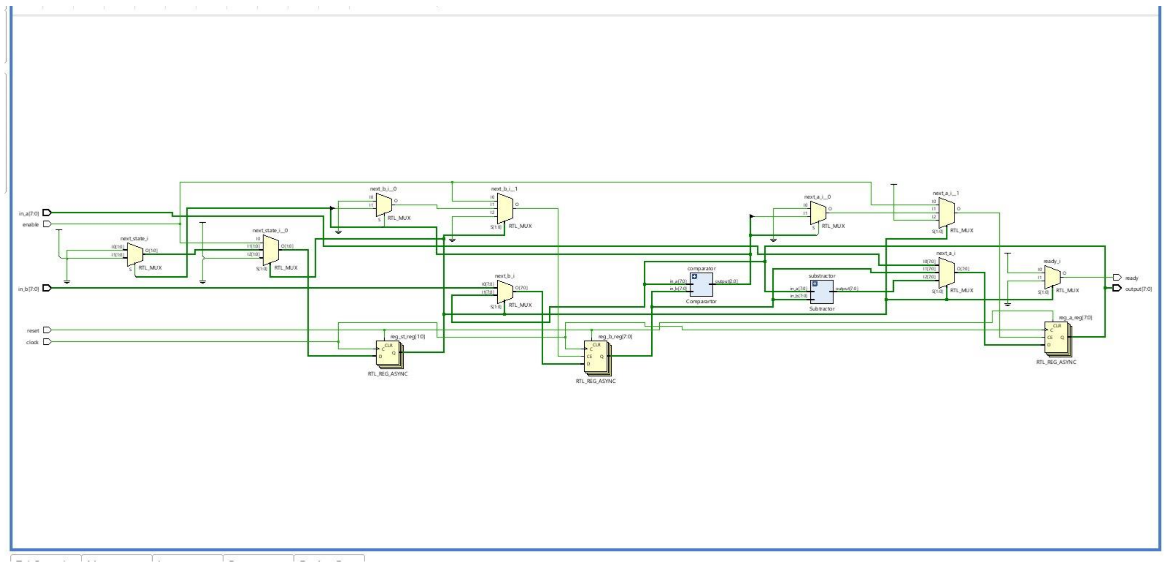
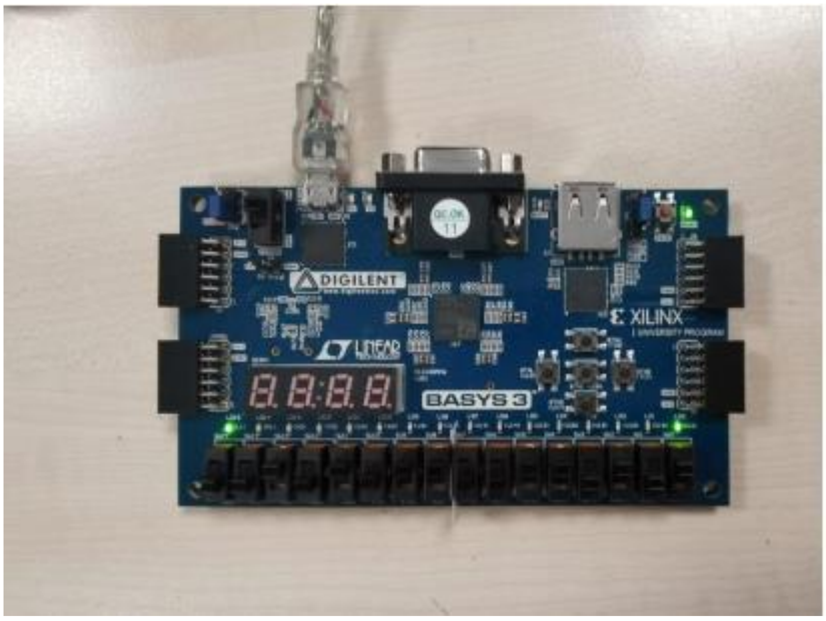
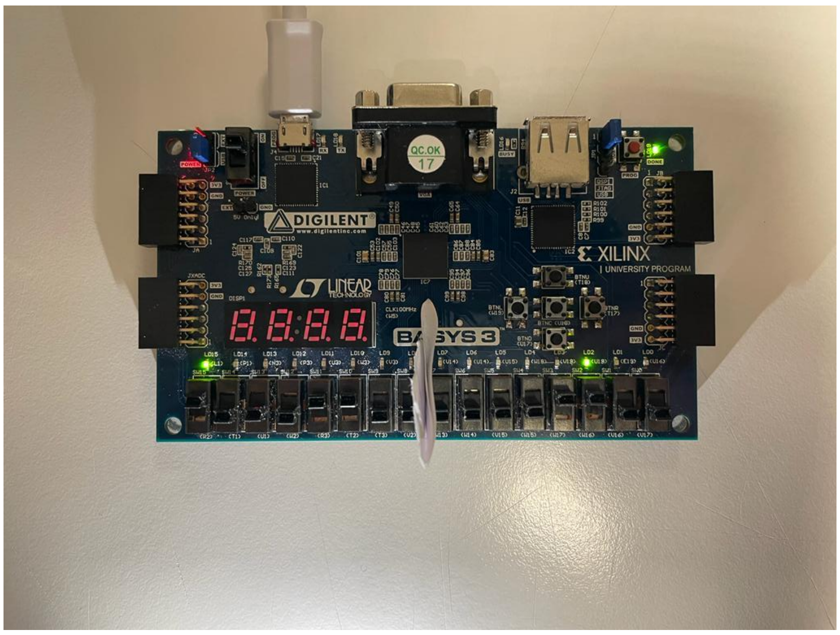

in_a[7:0]
enable
in_b[7:0]
reset
clock
next_state_i
next_state_i__0
next_b_i__0
next_b_i__1
next_b_i
reg_st_reg[1:0]
reg_b_reg[7:0]
comparator
Comparator
substractor
Subtractor
next_a_i__0
next_a_i__1
next_a_i
ready_i
reg_a_reg[7:0]
RTL_MUX
RTL_REG_ASYNC
ready
output[7:0]

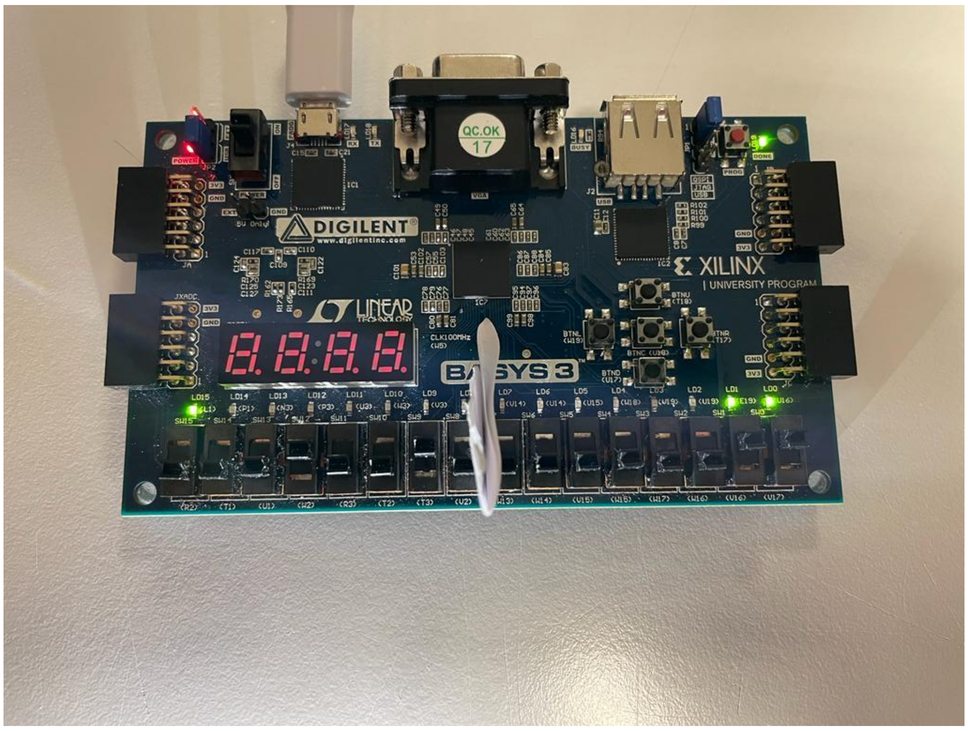
Figure (1.2)

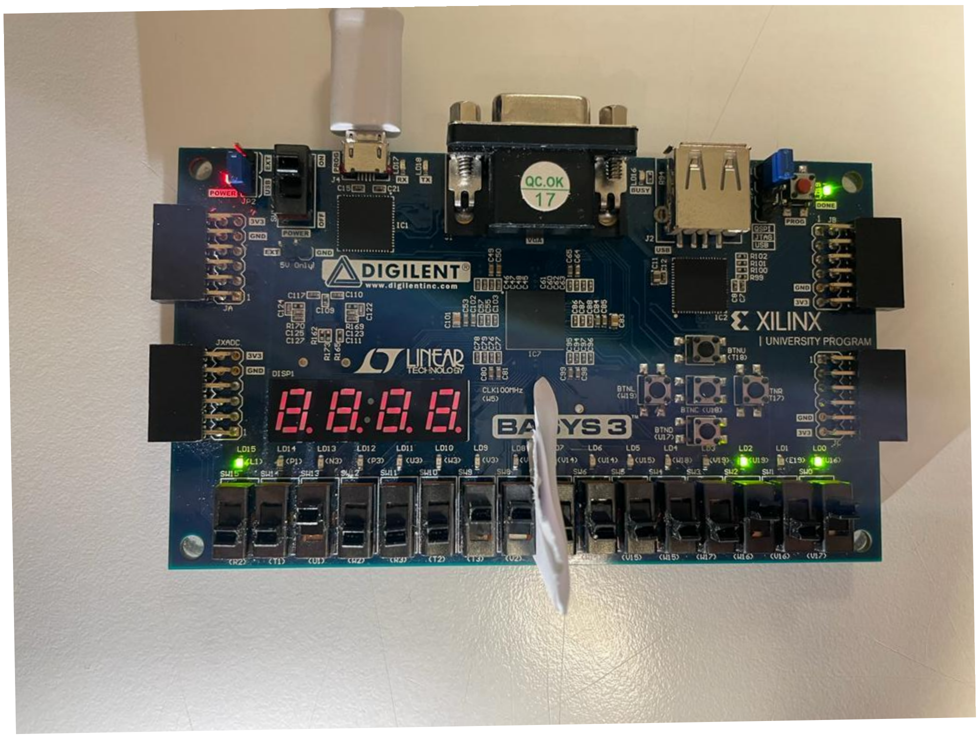


GCD of 43 and 53 (1)



GCD of 140 and 12 (4)


GCD of 18 and 3 (3)


GCD of 35 and 5 (5)

CONCLUSION:

Using the modulo and comparator modules, the greatest common divisor module has been built in VHDL. Many ideas, like how a Mealy machine functions, how to build and construct such a circuit, and how to fundamentally merge a combinational and a sequential circuit, have been reinforced throughout this lab work.

Appendices:

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer: emre_shash--
-- Create Date: 16.11.2023 11:08:24
-- Design Name:
-- Module Name:gcd- Behavioral
-- Project Name: Labbabalabla
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.05 - File Created
-- Additional Comments:
--
----------------------------------------------------------------
```

GCD.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
----------------------------------------------------------------
entity GCD is
port(
clk, rst: in std_logic;
enable: in std_logic;
Input_a, ib: in std_logic_vector(7 downto 0);
ready: out std_logic;
output: out std_logic_vector(7 downto 0)
);
end GCD ;
----------------------------------------------------------------
architecture behavioral of GCD is
type state_type is (hold, replace, substract);
signal regst, next_state : state_type;
signal rega, regb, na, nb: unsigned(7 downto 0);
signal ca,cb : unsigned(7 downto 0);
signal sb, sa, subsout: std_logic_vector(7 downto 0);
signal compout: std_logic_vector(2 downto 0);
begin
----------------------------------------------------------------
comp: entity work.Comp(rtl_comp);put=> compout);
substr: entity work.Subtr(rtl_Subt)
port map(input_a => sua, input_b => sb, output => subsout);
----------------------------------------------------------------
Process ( clk,rst)
begin
```

```
if rst='1' then;

regst <= hold;

rega <= (others=>'0');

regb <= (others=>'0');

elsif (rising_edge(clk)) then

regst <= next_state;

rega <= na;

regb <= nb;

end if;

end process;

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

process(regst,rega,regb,enable,input_a,input_b)

begin

na <= rega;

nb <= regb;

ca <= rega;

cb <= regb;

sa <= std_logic_vector(rega);

sb <= std_logic_vector(regb);

case regst is

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

when hold =>

if enable = '1' then

na<=unsigned(input_a);

neb<=unsigned(input_b);

nexstate<=replace;

else

nextstat<= hold;

end if;
```

```
---------------------------------------------------------------
when replace =>
if (compout(1) = '1') then
next_state <= hold;
else
if(compout(2) = '1') then
na <= regb;
nb <= rega;
end if;
next_state <= subst;
end if;
---------------------------------------------------------------
when substract =>
na <= unsigned(subs_out);
nextstate <= replace;
end case;
end process;
---------------------------------------------------------------
ready <= '1' when regst = hold else '0';
output <= std_logic_vector(rega);
---------------------------------------------------------------
end behavioral;
```

```
 --------------------------------------------------------------------------------
-- Company:
-- Engineer: emre_shash--
-- Create Date: 13.11.2023 13:45:41
```

```vhdl
-- Design Name:
-- Module Name:comp- Behavioral
-- Project Name: Labábalabla
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.05 - File Created
-- Additional Comments:
--
----------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Comp is
   port (
      input_a, input_b: in unsigned(7 downto 0);
      result_output: out std_logic_vector(2 downto 0)
   );
end entity;

architecture rtl_comp of Comp is
   signal equality_bits:std_logic_vector(7 downto 0);
   signal is_greater1,is_greater2: std_logic;
   signal result_bits:std_logic_vector(2 downto 0);
```

```
    signal is_equal, is_greater,is_smaller: std_logic;
begin


    equality_bits(0)<=(not input_a(0)) xnor (not input_b(0));
    equality_bits(1)<=(not input_a(1)) xnor (not input_b(1));
    equality_bits(2)<=(not input_a(2)) xnor (not input_b(2));
    equality_bits(3)<=(not input_a(3)) xnor (not input_b(3));
    equality_bits(4)<=(not input_a(4)) xnor (not input_b(4));
    equality_bits(5)<=(not input_a(5)) xnor (not input_b(5));
    equality_bits(6)<=(not input_a(6)) xnor (not input_b(6));
    equality_bits(7)<=(not input_a(7)) xnor (not input_b(7));
    is_equal<='1' when equality_bits = x"FF" else '0';


    is_greate<=(input_a(7) and not(input_b(7))) or
                (input_a(6) and not(input_b(6)) and equality_bits(7)) or
                (input_a(5) and not(input_b(5)) and equality_bits(7) and equality_bits(6)) or
                (input_a(4) and not(input_b(4)) and equality_bits(7) and equality_bits(6) and
equality_bits(5)) or
                (input_a(3) and not(input_b(3)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4)) or
                (input_a(2) and not(input_b(2)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3)) or
                (input_a(1) and not(input_b(1)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3) and equality_bits(2)) or
                (input_a(0) and not(input_b(0)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3) and equality_bits(2) and equality_bits(1));


    is_smaller<=(input_b(7) and not(input_a(7))) or
```

```
                (input_b(6) and not(input_a(6)) and equality_bits(7)) or
                (input_b(5) and not(input_a(5)) and equality_bits(7) and equality_bits(6)) or
                (input_b(4) and not(input_a(4)) and equality_bits(7) and equality_bits(6) and
equality_bits(5)) or
                (input_b(3) and not(input_a(3)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4)) or
                (input_b(2) and not(input_a(2)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3)) or
                (input_b(1) and not(input_a(1)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3) and equality_bits(2)) or
                (input_b(0) and not(input_a(0)) and equality_bits(7) and equality_bits(6) and
equality_bits(5) and equality_bits(4) and
                equality_bits(3) and equality_bits(2) and equality_bits(1));

   result_bits(2) <= is_smaller;
   result_bits(0) <= is_greater;
   result_bits(1) <= is_equal;
   result_output <= result_bits;

end rtl_comp;



library ieee;
use ieee.std_logic_1164.all;
------------------------------------------------------------------
entity testbench is
end testbench;
------------------------------------------------------------------
architecture rtl_tb of testbench is
```

```
signal clk, rst,enable,ready: std_logic;

signal input_a, input_b, output : std_logic_vector (7 downto 0);

type state_type is (hold, replace, subtract);

signal regst, next_state : state_type;

begin

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

dut: entity work.GCD(behavioral)

port map(clK,rSt,enable,input_a,input_b,ready,output);

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

clk_process: process begin

clk <= '0';

wait for 10ns;

clk <= '1';

wait for 10ns;

end process;

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

stim_process: process begin

enable <= '1';

input_a <= "10001100";

input_b <= "00001100";

rst <= '0';

wait for 100ns;

enable <= '0';

wait;

end process;

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

end rtl_tb;

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

configuration rtl_tb of testbench is
```

```
for rtl_tb
end for;
end rtl_tb;
```