ECE 501

Contemporary Digital Systems

Prof. Eric J. Balster

Homework #1

Aditya Sankaramanchi

101266059

**Introduction**

Boolean circuits in which all outputs at any given time depend on the inputs at that time are called as combinational logic circuits. The output for these circuits is a function of the present input. They are also called time-independent logic circuits. Examples of the combinational logic circuits are half adders, full adders, encoders and decoders.

To design and operate a digital logic circuit or an electronic circuit, we use a hardware description language (HDL). The two most widely used HDL’s in the industry are Verilog and VHDL. VHDL is a versatile and powerful programming language for designing and modelling digital hardware systems. It is inherently parallel in executing the commands which represent the logic gates. It also allows time specifications which act as gate delays to control the rate of processing data.

ModelSim is a powerful and widely used logic simulation tool that is used to simulate the behavior and performance of logic circuits. It can also be used for verification and debugging of digital circuits. This simulation tool allows the user to apply inputs to any digital circuit and to verify the outputs generated.

Quartus is a programmable logic device software tool from Altera. It is used for analysis and synthesis of HDL designs, compiling the design, perform timing analysis and view the digital logic RTL diagrams.

**Design**

The truth table considered for the combinational logic circuit is

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | D | F3 | F2 | F1 | F0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Design:

The architecture part of the VHDL code describes the internal operation of the design.

In VHDL there are four styles of coding:

1. Dataflow
2. Behavioral
3. Structural
4. Mixed Design

Dataflow:

For the dataflow design the circuit is described using logic gates like and, or, nor. The statements which are inside of the architecture are concurrent statements.

For the above truth table, considering A, B, C, D as the inputs to the circuit and F3, F2, F1, F0 are the outputs.

Karnaugh map:

K-map is a pictorial method used to simplify Boolean expressions. It helps in reducing the complexity of using Boolean algebra theorems and equation manipulations. K-map can also be called as a special version of truth table.

To find the Boolean expressions for the outputs F3, F2, F1, F0 we use the K-map technique.

The output expressions are:

The following architecture is used in VHDL code for the dataflow design:

architecture dataflow of homework\_1dataflow is

begin

F0 <= (not A and not B) or (not B and not C) or (not B and not D) or (A and B and C) or (not A and not C and D);

F1 <= (not C and D) or (not A and D) or (A and C and not D);

F2 <= (not B and not C) or (A and not C and not D) or (not A and not C and D) or (not A and not B and D) or (not A and B and C and not D) or (A and B and C and D);

F3 <= (not B and C) or ( B and not C and D) or (not A and B and D) or (A and C and not D) or (not A and not B and not D);

end dataflow;

Behavioral Design:

This type of design uses the sequential statements in which the output is described from the inputs directly.

The following process block in the architecture of the VHDL code describes the behavioral design:

architecture behavior of homework\_1behav is

hw1\_proc : process(inputs)

begin

case inputs is

when "0000" =>

outputs <= "1101";

when "0001" =>

outputs <= "0111";

when "0010" =>

outputs <= "1001";

.

.

end case;

end process hw1\_proc;

end behavior;

**Results:**

The simulation result for the dataflow design is

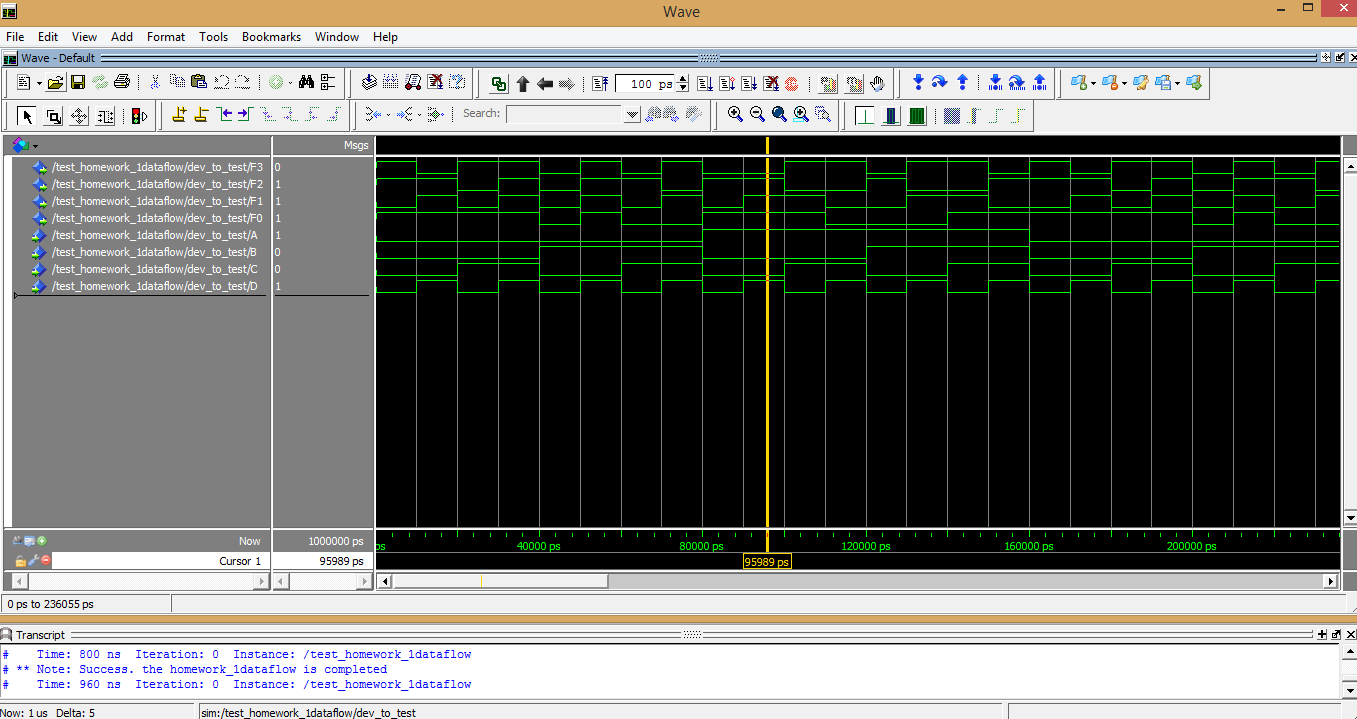


Fig. 1. Dataflow design result

From the above simulation we can observe that when the input ABCD = 1001 the output F3F2F1F0 = 0111. This result fulfills the truth table.

Quartus results:

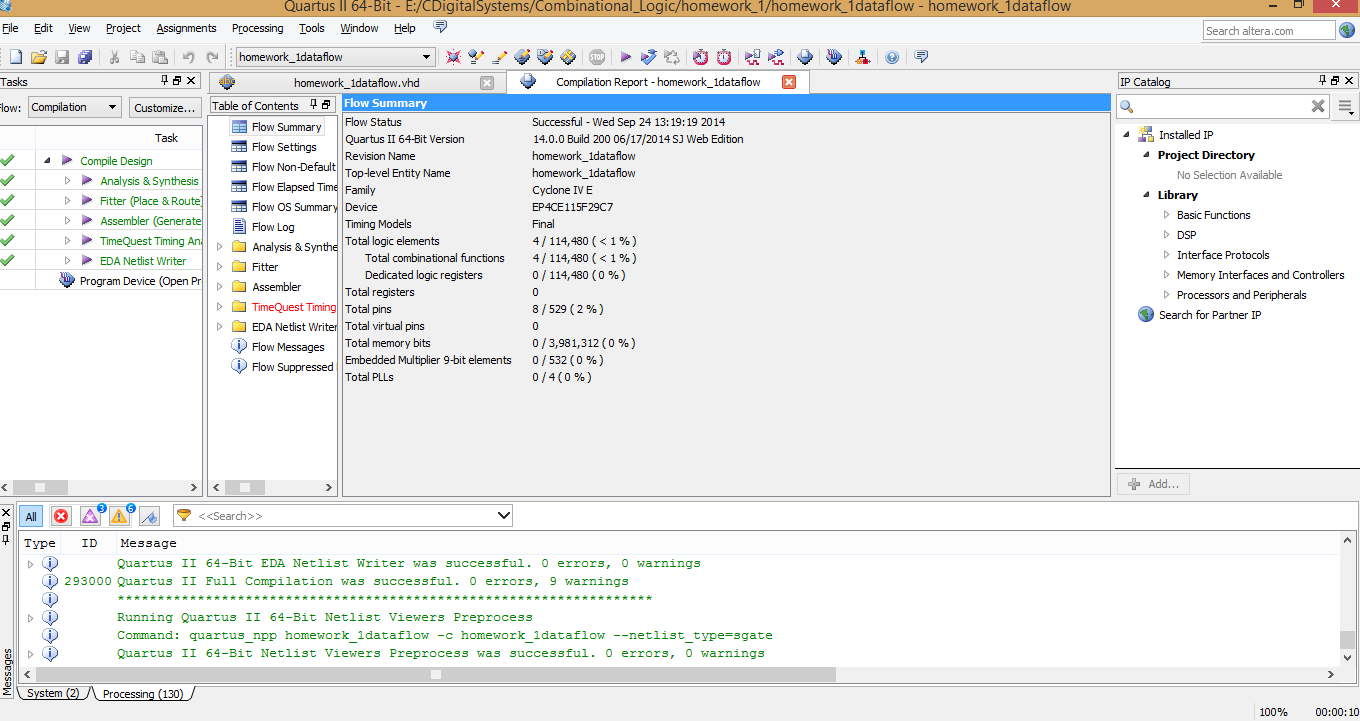


Fig.2. Quartus result for dataflow design

Description:

Total logic elements = 4

Total combinational functions = 4

Total pins = 8

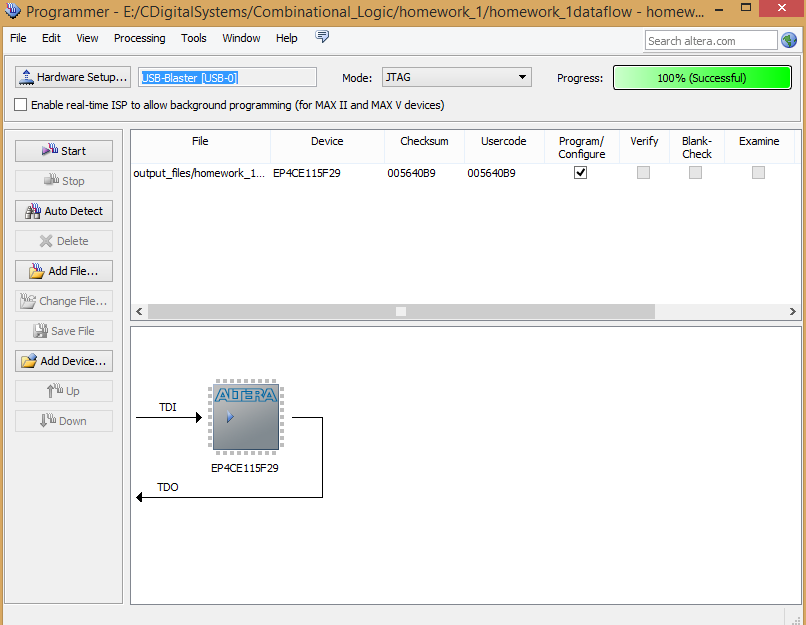


Fig.3. Quartus output with DE2-115 board connected

Now the DE2 board is used to test the functionality. The input switches considered are

1. PIN\_AB28
2. PIN\_AC28
3. PIN\_AC27
4. PIN\_AD27

The output LED’s are

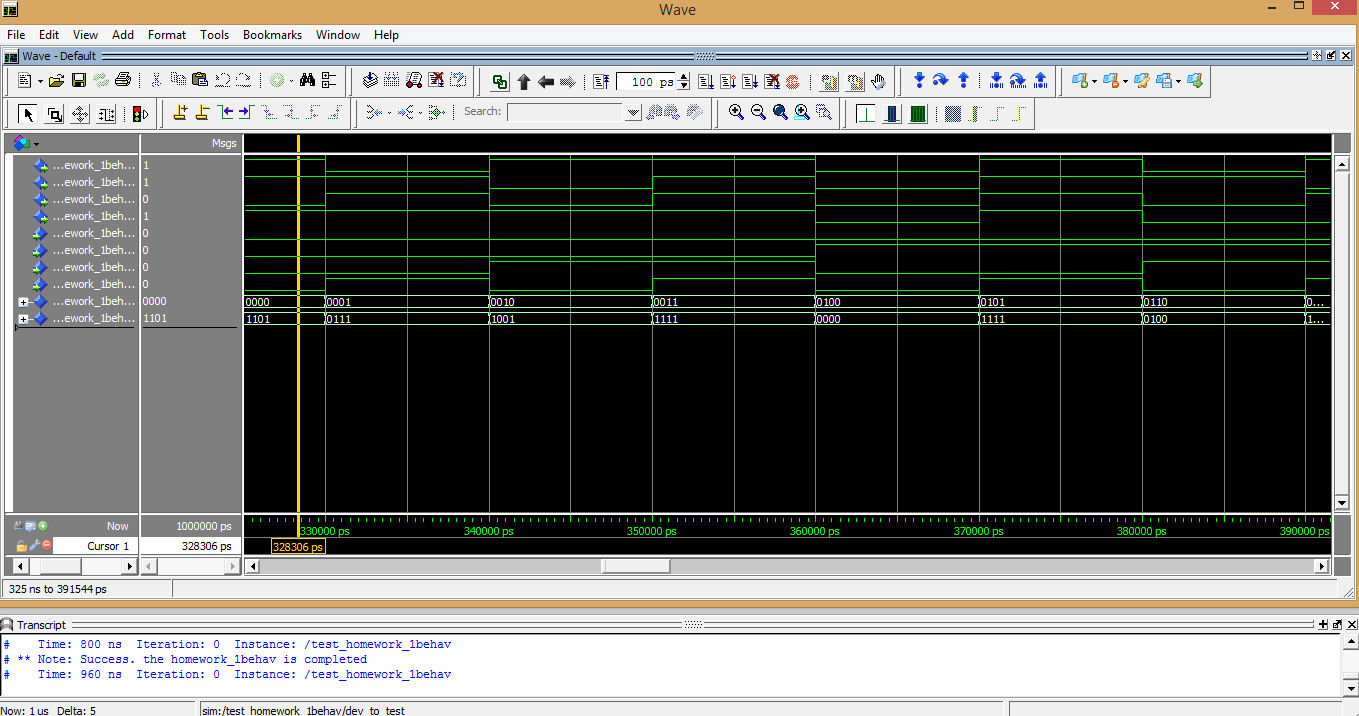
F3- PIN\_G19

F2-PIN\_F19

F1-PIN\_E19

F0-PIN\_F21

The simulation result for the behavioral design is:

 Fig.4. Behavioral design result

From the above simulation we can observe that when the input ABCD = 0000 the output F3F2F1F0 = 1101. This result fulfills the truth table.

Quartus result:

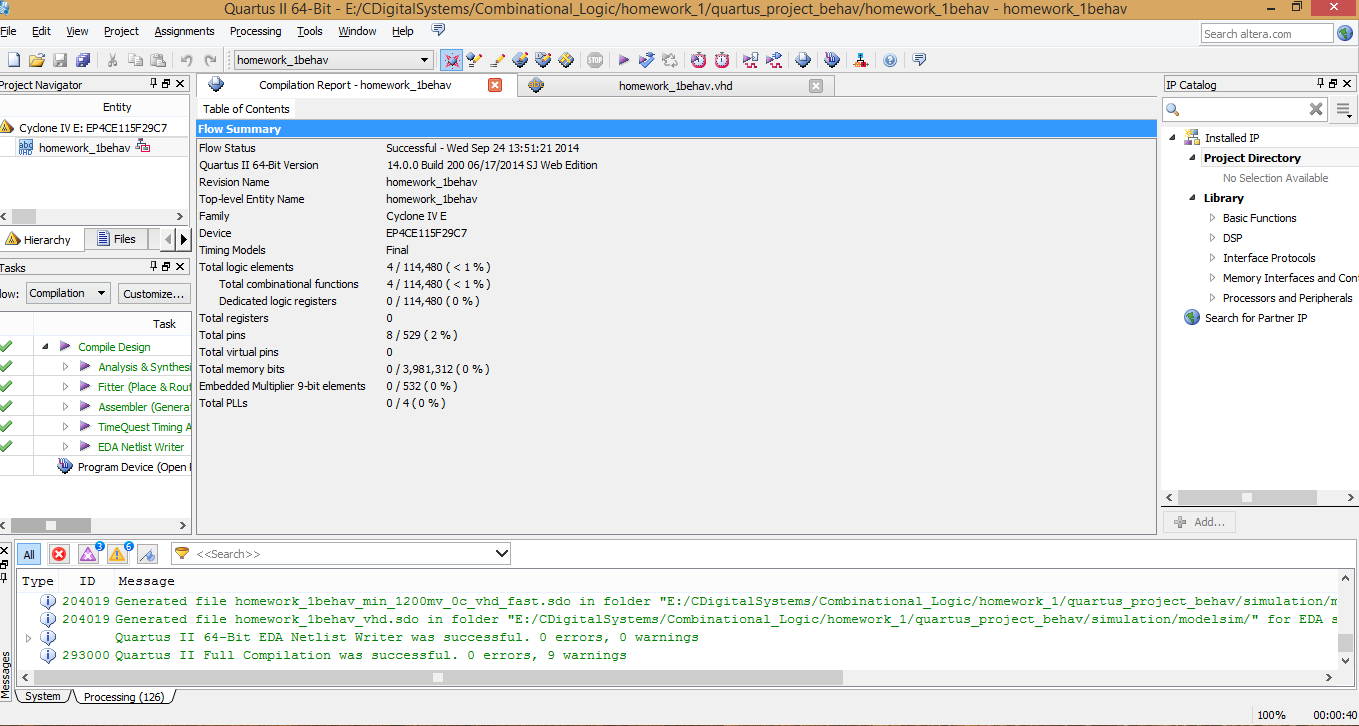


Fig.5. Quartus output for behavioral design

Description:

Total logic elements = 4

Total combinational functions = 4

Total pins = 8

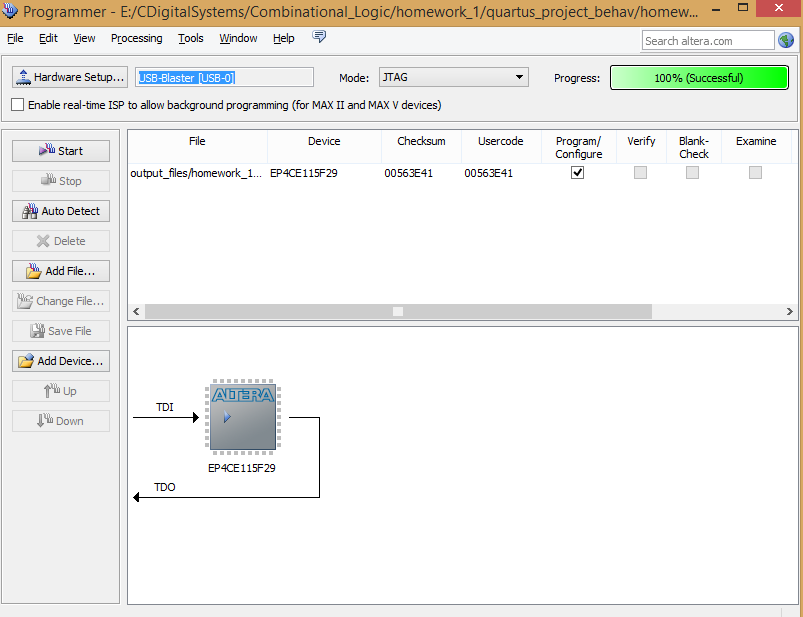


Fig.6. Quartus output with DE2-115 board connected

Now the DE2 board is used to test the functionality. The input switches considered are

1. PIN\_AB28
2. PIN\_AC28
3. PIN\_AC27
4. PIN\_AD27

The output LED’s are

F3- PIN\_G19

F2-PIN\_F19

F1-PIN\_E19

F0-PIN\_F21

**Conclusion:**

For the given truth table, the VHDL code is written in the dataflow designs and the behavioral design. For the dataflow design, the Boolean expressions are used in the architecture of the code. And for the behavioral design sequential statements are written in the process block. A test bench is used for every design to check for its correctness. The simulation is done using ModelSim and the output waveforms are obtained. The output satisfies the required functionality. Further, Quartus is used in developing logic design for the truth table and the DE2-115 board is used to test the functionality using input switches and output LEDs.

**Appendix- VHDL code:**

VHDL code for dataflow design:

homework\_1dataflow.vhd

-- data flow design

--

library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

use IEEE.numeric\_std.all;

entity homework\_1dataflow is

port (

F3 : out std\_logic; -- output

F2 : out std\_logic; -- output

F1 : out std\_logic; -- output

F0 : out std\_logic; -- output

A : in std\_logic; -- input

B : in std\_logic; -- input

C : in std\_logic; -- input

D : in std\_logic); -- input

end homework\_1dataflow;

architecture dataflow of homework\_1dataflow is

begin

F0 <= (not A and not B) or (not B and not C) or (not B and not D) or (A and B and C) or (not A and not C and D);

F1 <= (not C and D) or (not A and D) or (A and C and not D);

F2 <= (not B and not C) or (A and not C and not D) or (not A and not C and D) or (not A and not B and D)

or (not A and B and C and not D) or (A and B and C and D);

F3 <= (not B and C) or ( B and not C and D) or (not A and B and D) or (A and C and not D) or (not A and not B and not D);

end dataflow;

test\_homework\_1dataflow.vhd

-- testbench for data flow

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

use std.textio.all;

use ieee.std\_logic\_textio.all;

entity test\_homework\_1dataflow is

end;

architecture test of test\_homework\_1dataflow is

component homework\_1dataflow

port (

F3 : out std\_logic;

F2 : out std\_logic;

F1 : out std\_logic;

F0 : out std\_logic;

A : in std\_logic;

B : in std\_logic;

C : in std\_logic;

D : in std\_logic);

end component;

signal F3\_out : std\_logic;

signal F2\_out : std\_logic;

signal F1\_out : std\_logic;

signal F0\_out : std\_logic;

signal A\_in : std\_logic;

signal B\_in : std\_logic;

signal C\_in : std\_logic;

signal D\_in : std\_logic;

signal F3\_out\_expected : std\_logic;

signal F2\_out\_expected : std\_logic;

signal F1\_out\_expected : std\_logic;

signal F0\_out\_expected : std\_logic;

signal inputs : std\_logic\_vector(3 downto 0);

signal outputs : std\_logic\_vector(3 downto 0);

begin

dev\_to\_test : homework\_1dataflow

port map (

F3 => F3\_out, F2 => F2\_out, F1 => F1\_out, F0 => F0\_out,

A => A\_in, B => B\_in, C => C\_in, D => D\_in);

-- inputs

inputs <= A\_in & B\_in & C\_in & D\_in;

-- outputs

F3\_out\_expected <= outputs(3);

F2\_out\_expected <= outputs(2);

F1\_out\_expected <= outputs(1);

F0\_out\_expected <= outputs(0);

expected\_proc : process(inputs)

begin

case inputs is

when "0000" => outputs <= "1101";

when "0001" => outputs <= "0111";

when "0010" => outputs <= "1001";

when "0011" => outputs <= "1111";

when "0100" => outputs <= "0000";

when "0101" => outputs <= "1111";

when "0110" => outputs <= "0100";

when "0111" => outputs <= "1010";

when "1000" => outputs <= "0101";

when "1001" => outputs <= "0111";

when "1010" => outputs <= "1011";

when "1011" => outputs <= "1000";

when "1100" => outputs <= "0100";

when "1101" => outputs <= "1010";

when "1110" => outputs <= "1011";

when "1111" => outputs <= "0101";

when others => outputs <= (others => 'X');

end case;

end process expected\_proc;

stimulus : process

variable ErrCnt : integer := 0;

variable WriteBuf : line ;

begin

for i in std\_logic range '0' to '1' loop

A\_in <= i;

for j in std\_logic range '0' to '1' loop

B\_in <= j;

for k in std\_logic range '0' to '1' loop

C\_in <= k;

for l in std\_logic range '0' to '1' loop

D\_in <= l;

wait for 10 ns;

if (F3\_out\_expected /= F3\_out) then

write(WriteBuf, string'(" Error logic failed at F3\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F2\_out\_expected /= F2\_out) then

write(WriteBuf, string'(" Error logic failed at F2\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F1\_out\_expected /= F1\_out) then

write(WriteBuf, string'(" Error logic failed at F1\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F0\_out\_expected /= F0\_out) then

write(WriteBuf, string'(" Error logic failed at F0\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

end loop;

end loop;

end loop;

end loop;

if (ErrCnt = 0) then

report "Success. the homework\_1dataflow is completed";

else

report "homework\_1behav is broken " severity warning;

end if;

end process stimulus;

end test;

VHDL code for behavioral design:

homework\_1behav.vhd

-- homework 1 behavioral model

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

entity homework\_1behav is -- entity declaration

port (

F3 : out std\_logic;

F2 : out std\_logic;

F1 : out std\_logic;

F0 : out std\_logic;

A : in std\_logic;

B : in std\_logic;

C : in std\_logic;

D : in std\_logic

);

end homework\_1behav;

architecture behavior of homework\_1behav is

signal inputs : std\_logic\_vector(3 downto 0);

signal outputs : std\_logic\_vector(3 downto 0);

begin

inputs <= A & B & C & D;

F3 <= outputs(3);

F2 <= outputs(2);

F1 <= outputs(1);

F0 <= outputs(0);

hw1\_proc : process(inputs)

begin

case inputs is

when "0000" =>

outputs <= "1101";

when "0001" =>

outputs <= "0111";

when "0010" =>

outputs <= "1001";

when "0011" =>

outputs <= "1111";

when "0100" =>

outputs <= "0000";

when "0101" =>

outputs <= "1111";

when "0110" =>

outputs <= "0100";

when "0111" =>

outputs <= "1010";

when "1000" =>

outputs <= "0101";

when "1001" =>

outputs <= "0111";

when "1010" =>

outputs <= "1011";

when "1011" =>

outputs <= "1000";

when "1100" =>

outputs <= "0100";

when "1101" =>

outputs <= "1010";

when "1110" =>

outputs <= "1011";

when "1111" =>

outputs <= "0101";

when others =>

outputs <= "XXXX"; -- dont care combinations

end case;

end process hw1\_proc;

end behavior;

test\_homework\_1behav.vhd

-- test bench:::: for homework\_1behav.vhd

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.numeric\_std.all;

use std.textio.all;

use ieee.std\_logic\_textio.all;

entity test\_homework\_1behav is

end;

architecture test of test\_homework\_1behav is

component homework\_1behav

port (

F3 : out std\_logic;

F2 : out std\_logic;

F1 : out std\_logic;

F0 : out std\_logic;

A : in std\_logic;

B : in std\_logic;

C : in std\_logic;

D : in std\_logic);

end component;

signal F3\_out : std\_logic;

signal F2\_out : std\_logic;

signal F1\_out : std\_logic;

signal F0\_out : std\_logic;

signal A\_in : std\_logic;

signal B\_in : std\_logic;

signal C\_in : std\_logic;

signal D\_in : std\_logic;

signal F3\_out\_expected : std\_logic;

signal F2\_out\_expected : std\_logic;

signal F1\_out\_expected : std\_logic;

signal F0\_out\_expected : std\_logic;

signal inputs : std\_logic\_vector(3 downto 0);

signal outputs : std\_logic\_vector(3 downto 0);

begin

dev\_to\_test : homework\_1behav

port map (

F3 => F3\_out, F2 => F2\_out, F1 => F1\_out, F0 => F0\_out,

A => A\_in, B => B\_in, C => C\_in, D => D\_in);

-- inputs

inputs <= A\_in & B\_in & C\_in & D\_in;

-- outputs

F3\_out\_expected <= outputs(3);

F2\_out\_expected <= outputs(2);

F1\_out\_expected <= outputs(1);

F0\_out\_expected <= outputs(0);

expected\_proc : process(inputs)

begin

case inputs is

when "0000" => outputs <= "1101";

when "0001" => outputs <= "0111";

when "0010" => outputs <= "1001";

when "0011" => outputs <= "1111";

when "0100" => outputs <= "0000";

when "0101" => outputs <= "1111";

when "0110" => outputs <= "0100";

when "0111" => outputs <= "1010";

when "1000" => outputs <= "0101";

when "1001" => outputs <= "0111";

when "1010" => outputs <= "1011";

when "1011" => outputs <= "1000";

when "1100" => outputs <= "0100";

when "1101" => outputs <= "1010";

when "1110" => outputs <= "1011";

when "1111" => outputs <= "0101";

when others => outputs <= (others => 'X');

end case;

end process expected\_proc;

stimulus : process

variable ErrCnt : integer := 0;

variable WriteBuf : line ;

begin

for i in std\_logic range '0' to '1' loop

A\_in <= i;

for j in std\_logic range '0' to '1' loop

B\_in <= j;

for k in std\_logic range '0' to '1' loop

C\_in <= k;

for l in std\_logic range '0' to '1' loop

D\_in <= l;

wait for 10 ns;

if (F3\_out\_expected /= F3\_out) then

write(WriteBuf, string'(" Error logic failed at F3\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F2\_out\_expected /= F2\_out) then

write(WriteBuf, string'(" Error logic failed at F2\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F1\_out\_expected /= F1\_out) then

write(WriteBuf, string'(" Error logic failed at F1\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

if (F0\_out\_expected /= F0\_out) then

write(WriteBuf, string'(" Error logic failed at F0\_out : A ="));

write(WriteBuf, A\_in);

write(WriteBuf, string'(", B ="));

write(WriteBuf, B\_in);

write(WriteBuf, string'(", C ="));

write(WriteBuf, C\_in);

write(WriteBuf, string'(", D ="));

write(WriteBuf, D\_in);

writeline(output, WriteBuf);

ErrCnt := ErrCnt+1;

end if;

end loop;

end loop;

end loop;

end loop;

if (ErrCnt = 0) then

report "Success. the homework\_1behav is completed";

else

report "homework\_1behav is broken " severity warning;

end if;

end process stimulus;

end test;