Efe Tarhan

22002840

EE102-02

08.12.2021

LAB 6: GREATEST COMMON DIVISOR

A) PURPOSE

Aim of this experiment is to understand and implement registers in VHDL projects. Objective of the lab is designing an GCD unit which takes to numbers as inputs and gives back the greatest common divisor of the inputs. The process is controlled with 2 buttons. GO button initates the the procedure and reset button resets the calculation by assessing 0 to both of the numbers.

B) DESIGN METHODOLOGY

In the design of the GCD unit a multiplexer component, a register component, a datapath component and a state component has created.

In the design the multiplexer assigns the register inputs according to their selection inputs. This extra component has been made to guarantee the controlled process of the design. There are two multiplexers for two numbers in the design.

Registers are another component of this design. There are 3 registers for two for the input numbers and one for the greatest common divisor of these two numbers. Those registers assign values to their outputs depending on the state that they are currently in.

In the state component there are 7 states for the algorithm. The start state is the initial state of the FSM where the process waits the GO input to start. The input state is actually an unnecessary state because it doesn’t change any input or output but it generates a

delay to provide enough time for the registers and multiplexers to prepare the signals ready for operation. The test states checks the inequality of the two numbers and outputs two inputs depending on the relation between the signals. In the update state the outputs are updated by the Euclidian Algorithm and return back to the test state. This loop is continued until the numbers became equal to each other.

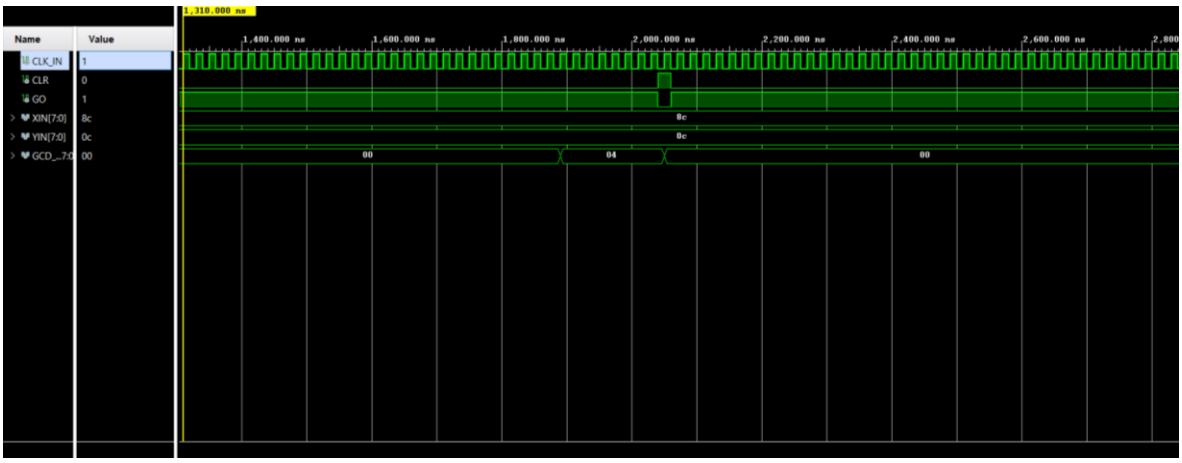
This FSM is a mealy machine since the CLR is an asynchronous input of the system which sets the next state to the start state.

The FSM design is cheaper than a possible combinational circuit model but since its dependence on the clock it will be slower than it. The combinational circuit that can operate this process requires many and, or, nand, nor ,etc gates. That is the reason why it will be more expensive than the FSM design.

C) RESULTS

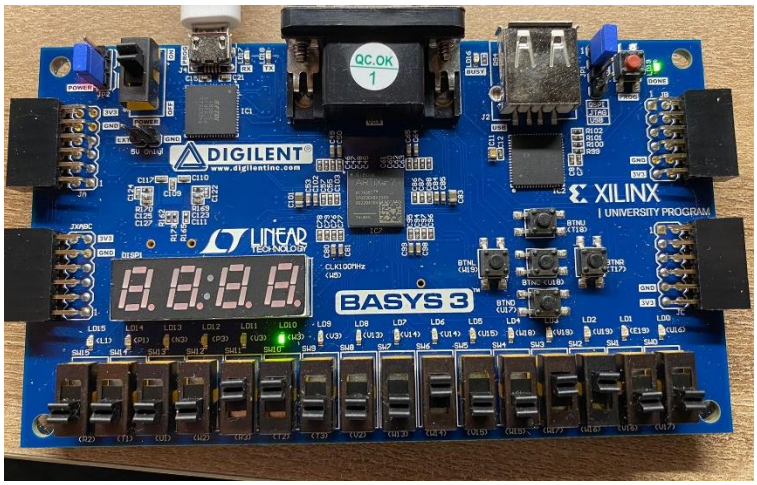
After completing the design step a simulation has been written to test if the code is working. First number is set to 140 and the second number is set to 12, the simulation clock is generated with period 20ns. The simulation result fits with the expected behaviour of the design. The waveform graph of the simulation can be seen below.

In the simulation where the GCD of 140 and 12 is calculated. It takes 42 clock cycles for this FSM to find the result. This value can be decreased by elimininating several states. For example the input state might be changed with an alternative algorithm to take one clock cycle out or the test1 and test2 states can be reduced to a single state to optimize the time further more.

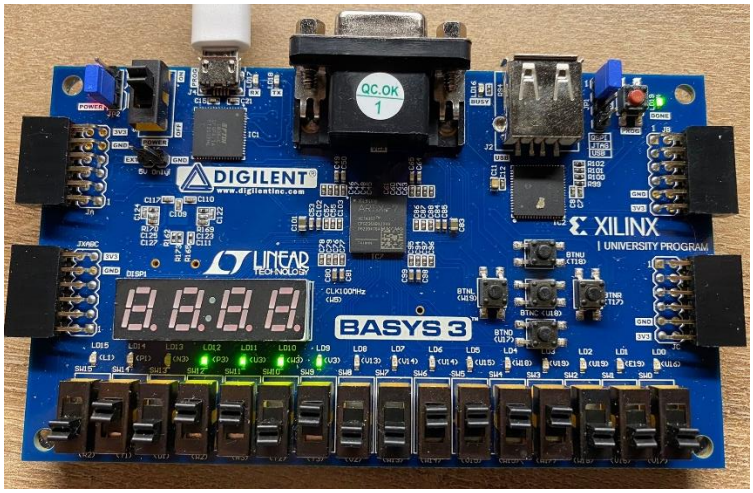


(Image 1: Waveform graph of the simulation applied on the design)

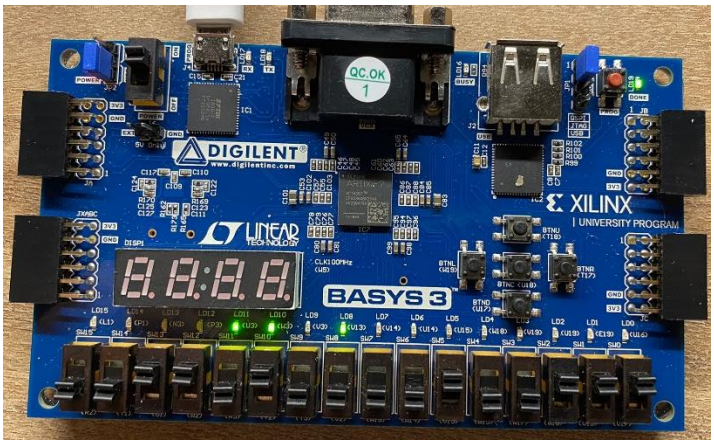
After writing the constraint file and programming the device few tests has been made. The first 8 switches represent the first number, other 8 switches represent the second number and first 8 leds represent the greatest common divisor of the numbers.



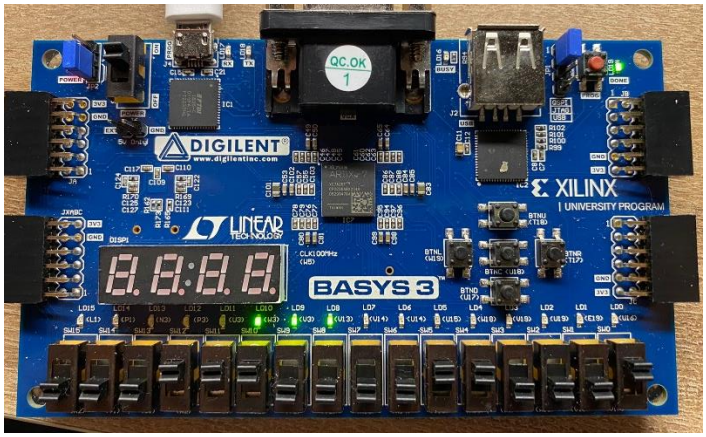
(Image 2: When NUM1 = “00001100” and NUM2 = “01001100” the GCD = “00000100”)



(Image 3: When NUM1 = “01011010” and NUM2 = “01111000” the GCD = “00011110”)



(Image 4: When NUM1 = “00110100” and NUM2 = “00100111” the GCD = “00001101”)



(Image 5: When NUM1 = “00011100” and NUM2 = “00110001” the GCD = “00000100”)

D) CONCLUSION

This experiment was particularly helpful in understanding rergisters and controlling them by using clock ticks. In the first parts of the experiments I tried several algorithms by myself but because of some errors I decided to investigate the sources on the internet and got help from various sources. This experiment also helped me to understand and design better state machines which will be helpful in the term project demo.

E)APPENDIX

Design Code:

MULTIPLEXER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity MUX_GCD is
port(
NUM1: in std_logic_vector(7 downto 0);
NUM2:in std_logic_vector(7 downto 0);
SEL: in std_logic;
OUTNUM:out std_logic_vector(7 downto 0));
end MUX_GCD;

architecture MUX_GCD of MUX_GCD is

begin
with SEL select OUTNUM<=
NUM1 when '0',
NUM2 when '1';

end MUX_GCD;
```

REGISTER

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;

entity REG_GCD is
port(
LOAD_REG: in std_logic;
REG_IN: in std_logic_vector(7 downto 0);
CLK_IN:in std_logic;
CLR: in std_logic;
REG_OUT:out std_logic_vector(7 downto 0));
end REG_GCD;

architecture REG_GCD of REG_GCD is

begin
process(CLK_IN)
begin
if rising_edge(CLK_IN) then
if CLR = '1' then
REG_OUT<=(others=>'0');
else
if LOAD_REG='1' then
REG_OUT <= REG_IN;
end if;
end if;
end if;
end process;

end REG_GCD;
```

STATES

```
library IEEE;
```

```
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity STATES_GCD is
Port (
CLK_IN: in std_logic;
CLR: in std_logic;
GO:in std_logic;
EQUAL: in std_logic;
LITTLE: in std_logic;
SLCT_X: out std_logic;
SLCT_Y: out std_logic;
X_LOAD: out std_logic;
Y_LOAD: out std_logic;
G_LOAD: out std_logic);
end STATES_GCD;

architecture STATES_GCD of STATES_GCD is
type state_type is(START, INPUT, F_TEST,S_TEST,F_UPD,S_UPD,DONE);
signal PRE_STATE, NEX_STATE: state_type;
begin

SREG: process(CLK_IN, CLR)
begin
if CLR = '1' then
PRE_STATE<=START;
elsif RISING_EDGE(CLK_IN)then
PRE_STATE<=NEX_STATE;
end if;
end process;
```

```
C1: process(PRE_STATE,GO,EQUAL,LITTLE)
begin
case PRE_STATE is
when START=>
if GO='1' then
NEX_STATE<=INPUT;
else
NEX_STATE<=START;
end if;

when INPUT=>
NEX_STATE<=F_TEST;

when F_TEST=>
if EQUAL = '1' then
NEX_STATE<=DONE;
else
NEX_STATE<=S_TEST;
end if;

when S_TEST=>
if LITTLE='1' then
NEX_STATE<= F_UPD;
else
NEX_STATE<=S_UPD;
end if;

when F_UPD=>
NEX_STATE<=F_TEST;

when S_UPD=>
```

```
NEX_STATE<=F_TEST;

when DONE=>
NEX_STATE<=DONE;

WHEN OTHERS =>
NULL;

end case;
end process;

C2: process(PRE_STATE)
begin
X_LOAD<='0';Y_LOAD<='0';G_LOAD<='0';SLCT_X<='0';SLCT_Y<='0';

case PRE_STATE is

when INPUT=>
X_LOAD<='1';Y_LOAD<='1';
SLCT_X<='1';SLCT_Y<='1';

when F_UPD=>
Y_LOAD<='1';

when S_UPD=>
X_LOAD<='1';

when DONE=>
G_LOAD<='1';

WHEN OTHERS =>
```

```
NULL;

end case;
end process;
end STATES_GCD;
```

ALGORITHM PATH

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;

entity GCD_BEHAV is
Port (
CLK_IN:in std_logic;
CLR: in std_logic;
SLCT_X: in std_logic;
SLCT_Y: in std_logic;
X_LOAD: in std_logic;
Y_LOAD: in std_logic;
G_LOAD: in std_logic;
XIN: in std_logic_vector(7 downto 0);
YIN: in std_logic_vector(7 downto 0);
GCD: out std_logic_vector(7 downto 0);
EQUAL: out std_logic;
LITTLE: out std_logic);
end GCD_BEHAV;


architecture Behavioral of GCD_BEHAV is
component MUX_GCD
port(
```

```
NUM1: in std_logic_vector(7 downto 0);
NUM2:in std_logic_vector(7 downto 0);
SEL: in std_logic;
OUTNUM:out std_logic_vector(7 downto 0));
end component;

component REG_GCD
port(
LOAD_REG: in std_logic;
REG_IN: in std_logic_vector(7 downto 0);
CLK_IN:in std_logic;
CLR: in std_logic;
REG_OUT:out std_logic_vector(7 downto 0));
end component;

signal X,OUTNUM,F_X,F_Y,X_MINUS_Y,Y_MINUS_X: std_logic_vector(7 downto 0);

begin
X_MINUS_Y<=X-OUTNUM;
Y_MINUS_X<=OUTNUM-X;

EQ: process(X,OUTNUM)
begin
if X=OUTNUM then
EQUAL<= '1';
else
EQUAL<='0';
end if;
end process;

LT: process(X,OUTNUM)
```

```
begin
if X<OUTNUM then
LITTLE<='1';
else
LITTLE<='0';
end if;
end process;

MUX1: MUX_GCD
port map(NUM1=>X_MINUS_Y, NUM2=>XIN,SEL=>SLCT_X, OUTNUM=>F_X);

MUX2: MUX_GCD
port map(NUM1=>Y_MINUS_X, NUM2=>YIN, SEL=>SLCT_Y, OUTNUM=>F_Y);

REG1: REG_GCD
port map(LOAD_REG=>X_LOAD, REG_IN=>F_X, CLK_IN=>CLK_IN, CLR=>CLR,
REG_OUT=>X);

REG2: REG_GCD
port map(LOAD_REG=>Y_LOAD, REG_IN=>F_Y, CLK_IN=>CLK_IN, CLR=>CLR,
REG_OUT=>OUTNUM);

REG3: REG_GCD
port map(LOAD_REG=>G_LOAD, REG_IN=>X, CLK_IN=>CLK_IN, CLR=>CLR,
REG_OUT=>GCD);

end Behavioral;
```

TOP MODULE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity MOD_GCD is
Port (
GO:in std_logic;
XIN:in std_logic_vector(7 downto 0);
YIN: in std_logic_vector(7 downto 0);
CLK_IN: in std_logic;
CLR: in std_logic;
GCD_OUT: out std_logic_vector(7 downto 0));
end MOD_GCD;

architecture Behavioral of MOD_GCD is
component GCD_BEHAV
Port (
CLK_IN:in std_logic;
CLR: in std_logic;
SLCT_X: in std_logic;
SLCT_Y: in std_logic;
X_LOAD: in std_logic;
Y_LOAD: in std_logic;
G_LOAD: in std_logic;
XIN: in std_logic_vector(7 downto 0);
YIN: in std_logic_vector(7 downto 0);
GCD: out std_logic_vector(7 downto 0);
EQUAL: out std_logic;
LITTLE: out std_logic);
end component;

component STATES_GCD
Port(
CLK_IN: in std_logic;
CLR: in std_logic;
```

```
GO:in std_logic;
EQUAL: in std_logic;
LITTLE: in std_logic;
SLCT_X: out std_logic;
SLCT_Y: out std_logic;
X_LOAD: out std_logic;
Y_LOAD: out std_logic;
G_LOAD: out std_logic);
end component;

signal EQUAL, LITTLE, SLCT_X,SLCT_Y: std_logic;
signal X_LOAD,Y_LOAD,G_LOAD: std_logic;

begin

SIG1:GCD_BEHAV port
map(CLK_IN=>CLK_IN,CLR=>CLR,SLCT_X=>SLCT_X,SLCT_Y=>SLCT_Y,X_LOAD=>X_LOAD
,Y_LOAD=>Y_LOAD,
G_LOAD=>G_LOAD,XIN=>XIN,YIN=>YIN,GCD=>GCD_OUT,EQUAL=>EQUAL,LITTLE=>LITT
LE);

SIG2:STATES_GCD port
map(CLK_IN=>CLK_IN,CLR=>CLR,GO=>GO,EQUAL=>EQUAL,LITTLE=>LITTLE,
SLCT_X=>SLCT_X,SLCT_Y=>SLCT_Y,X_LOAD=>X_LOAD,Y_LOAD=>Y_LOAD,G_LOAD=>G_
LOAD);
end Behavioral;
```

Test Bench Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity GCD_SIM is
```

```
end GCD_SIM;

architecture Behavioral of GCD_SIM is


COMPONENT MOD_GCD IS
PORT(
CLK_IN: in std_logic;
CLR: in std_logic;
GO:in std_logic;
XIN:in std_logic_vector(7 downto 0);
YIN: in std_logic_vector(7 downto 0);
GCD_OUT: out std_logic_vector(7 downto 0));
END COMPONENT;

SIGNAL CLK_IN,CLR,GO : STD_LOGIC;
SIGNAL XIN, YIN, GCD_OUT: STD_LOGIC_VECTOR(7 DOWNTO 0);

begin

SIM_GCD : MOD_GCD PORT MAP(
CLK_IN => CLK_IN,
CLR => CLR,
GO => GO,
XIN => XIN,
YIN => YIN,
GCD_OUT => GCD_OUT
);


CLOCK : PROCESS
```

```
BEGIN
WAIT FOR 10NS;
CLK_IN <= '1';
WAIT FOR 10NS;
CLK_IN <= '0';
END PROCESS;

TESTBENCH1 : PROCESS
BEGIN



CLR <= '1';
WAIT FOR 20NS;

CLR <= '0';
WAIT FOR 1000NS;

XIN <= "10001100";
YIN <= "00001100";
END PROCESS;

TESTBENCH2 : PROCESS
BEGIN
GO <= '0';
WAIT FOR 20NS;

GO <= '1';
WAIT FOR 1000NS;
END PROCESS;

end Behavioral;
```

F) INDEX

LBEbooks “Lesson 96 - Example 64: GCD Algorithm-2 - Datapath Control.” YouTube, 22 Nov. 2012, https://www.youtube.com/watch?v=IxPWTMEv5dI.