Experiment 2 : 4 Bit Divider

Badal Varshney, Roll Number 19D070015

EE-214, WEL, IIT Bombay

March 10, 2021

## Overview of the experiment:

|  |
| --- |
| The purpose of the experiment was to design a system capable of divide two four bit numbers by Behavioral Description. The design of the project was done in Quartus Prime and a generic testbench was used which reads data from a trace file containing the test cases to check whether the expected answers match the outputs. |

## Approach to the experiment:

|  |
| --- |
| The 4 bit divider was implemented using Behavioral Description in which we use generic port in entity declaration, use ‘for’ loop statement and variable declaration statement like in c programming language  And uses sequential statements in process for calculate the quotient and remainder. Testbench was compiled which read data from the trace file whose first column was the 8 bit input, the second column was the expected output and the last was the mask bit to determine whether to check the corresponding bit or not( checked if set to 1). The trace file was generated using python code. |

## Design document and VHDL code if relevant:

|  |
| --- |
| **Code of divider**-  library ieee;  use ieee.std\_logic\_1164.all;  entity divi is  generic(  N : integer:=4; -- operand width  NN : integer:=8 -- result width  );  port (  Nu: in std\_logic\_vector(N-1 downto 0);-- Nu (read numerator) is dividend  D: in std\_logic\_vector(N-1 downto 0);-- D (read Denominator) is divisor  RQ: out std\_logic\_vector((NN)-1 downto 0)--upper N bits of RQ will have remainder and  --lower N bits will have quotient  ) ;  end divi;  architecture beh of divi is  -- unconstrained 1D x 1D array  type pr\_type is array (natural range<>) of std\_logic\_vector(NN-1 downto 0);  -- subtractor function. [Usage: var := sub(X,Y) where var is a variable  -- and X,Y are two 4-bit inputs for subtractor]  function sub(A: in std\_logic\_vector; B: in std\_logic\_vector)  return std\_logic\_vector is  -- variable declaration  variable W : integer := A'length;  variable diff : std\_logic\_vector(W downto 0):= (others=>'0');  variable borrow : std\_logic\_vector(W downto 0):= (0 => '1', others=>'0');  variable B\_sign: std\_logic\_vector(A'length-1 downto 0):=(others=>'0');  begin  B\_sign(B'length-1 downto 0) := not B;  for i in 0 to W-1 loop  diff(i) := A(i) xor B\_sign(i) xor borrow(i);  borrow(i+1) := (A(i) and B\_sign(i)) or (borrow(i) and (A(i) xor B\_sign(i)));  end loop;  diff(W) := not borrow(W);  return diff;  end sub;    begin  division : process(Nu, D)  -- Here Nu (read numerator) is dividend and D (read denominator) is divisor  -- variable k holds length of dividend  variable k : integer := Nu'length;  variable rem\_hold: pr\_type(0 to N) := (others=>(others=>'0'));  -- 1D x 1D array should be used, instead of reading and writing to same variable  -- (This is a limitation of VHDL synthesizer)    -- declare variable to hold partial remainder for subsequent iteration  variable part\_remd : std\_logic\_vector(N downto 0) :=(others=>'0');    -- declare variable to hold difference from subtractor  variable diff\_hold : std\_logic\_vector(NN-1 downto 0) := (others=>'0');    -- declare temporary variable to hold prior partial product in case difference is negative  variable pre\_part : std\_logic\_vector(N-1 downto 0) := (others=>'0');      begin  rem\_hold(0)(N-1 downto 0):= Nu;      -- sequential statements to calculate quotient and remainder      for i in 1 to N loop  for k in 0 to NN-2 loop  rem\_hold(i)(k+1) :=rem\_hold(i-1)(k); --//SHIFTING LEFT  end loop;    diff\_hold:= rem\_hold(i); --storing the value  pre\_part:=rem\_hold(i)(NN-1 downto N);  part\_remd:= sub(pre\_part, D); --subtracting the prior partial product and Divisor  if part\_remd(N)='0' then --checking  rem\_hold(i)(NN-1 downto N) := part\_remd(N-1 downto 0);  rem\_hold(i)(0) := '1';  else  rem\_hold(i) := diff\_hold;  rem\_hold(i)(0) := '0';  end if;  end loop;  RQ <= rem\_hold(4);-- final result assignment  end process ; -- division  end beh ; -- beh  **Code of DUT file-**  -- A DUT entity is used to wrap your design.  -- This example shows how you can do this for the    library ieee;  use ieee.std\_logic\_1164.all;      entity DUT is  port(input\_vector: in std\_logic\_vector(7 downto 0);  output\_vector: out std\_logic\_vector(7 downto 0));  end entity;        architecture DutWrap of DUT is  component divi is  generic(  N : integer:=4; -- operand width  NN : integer:=8 -- result width  );  port (  Nu: in std\_logic\_vector(N-1 downto 0);-- Nu (read numerator) is dividend  D: in std\_logic\_vector(N-1 downto 0);-- D (read Denominator) is divisor  RQ: out std\_logic\_vector((NN)-1 downto 0)--upper N bits of RQ will have remainder and  --lower N bits will have quotient  ) ;  end component;  begin    -- input/output vector element ordering is critical,  -- and must match the ordering in the trace file!  add\_instance: divi  port map (  -- order of inputs Cin B A    Nu(3) => input\_vector(7),  Nu(2) => input\_vector(6),  Nu(1) => input\_vector(5),  Nu(0) => input\_vector(4),  D(3) => input\_vector(3),  D(2) => input\_vector(2),  D(1) => input\_vector(1),  D(0) => input\_vector(0),    -- order of outputs S Cout  RQ(7) => output\_vector(7),  RQ(6) => output\_vector(6),  RQ(5) => output\_vector(5),  RQ(4) => output\_vector(4),  RQ(3) => output\_vector(3),  RQ(2) => output\_vector(2),  RQ(1) => output\_vector(1),  RQ(0) => output\_vector(0));  end DutWrap; |

## RTL View:

|  |
| --- |
| Attach screen-shot of the RTL view generated by Quartus. |

## DUT Input/Output Format:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mention the format (LSB/MSB of input and output) and few test cases from trace-file.   |  |  |  | | --- | --- | --- | | Input bits | Output bits | Mask bits | | 01011111 | 01010000 | 11111111 | | 01100000 | 01010000 | 00000000 | |

## RTL Simulation:

|  |
| --- |
| Attach the clearly visible screen-shot of RTL simulation waveforms. |

## Gate-level Simulation:

|  |
| --- |
| Attach the clearly visible screen-shot of Gate-level Simulation. |

## Krypton board\*:

|  |
| --- |
| Map the logic circuit to the Krypton board and attach the images of the pin assignment and output observed on the board (switches/LEDs). |

## Observations\*:

|  |
| --- |
| You must summarize your observations, either in words, using figures and/or tables. |

## References:

|  |
| --- |
| You may include the references if any. |

\* To be submitted after the tutorial on ”Using Krypton.