CONCORDIA UNIVERSITY
Department of Electrical and Computer Engineering
COEN 313 - Digital Systems Design II - Winter 2020

Lab 3: A combinational sign-magnitude to two's complement converter

Objectives

• To become familiar with using VHDL combinational processes to design combinational logic.
• To become acquainted with some useful command line options to the vcom command.

Introduction

In this lab, a combinational VHDL process will be used to design a combinational circuit which converts a 4 bit input representing a signed integer in sign-magnitude notation into its equivalent representation in 4 bit two's complement representation. Recall that in sign magnitude representation, the most signficant bit represents the sign of the number (0 meaning a positive number and 1 meaning a negative number) while the remaining bits represent the magnitude of the number. For positive numbers, a given number has the same representation in both sign-magnitude and two's complement representation. Negative numbers have different representations. Table 1 gives the 16 possible inputs and the numbers these patterns represent both in sign-magnitude notation and in two's complement notation. The first and last columns of Table 1 may thus be considered the inputs and the outputs of the converter circuit.

Table 1: Sign-Magnitude and 2's complement representation

Input	Value is sign-magnitude	Value in 2's complement	Output
0000	+0	+0	0000
0001	+1	+1	0001
0010	+2	+2	0010
0011	+3	+3	0011
0100	+4	+4	0100
0101	+5	+5	0101
0110	+6	+6	0110
0111	+7	+7	0111
1000	-0	-8	1000

Table 1: Sign-Magnitude and 2's complement representation

Input	Value is sign-magnitude	Value in 2's complement	Output
1001	-1	-7	1111
1010	-2	-6	1110
1011	-3	-5	1101
1100	-4	-4	1100
1101	-5	-3	1011
1110	-6	-2	1010
1111	-7	-1	1001

The following algorithm may be used to convert from sign-magnitude into two's complement representation:

```
if  (the input is a positive number) then
{
  the output is the same as the input
}
else
{
  negate the magnitude bits of the input;
  add "001" to the negated magnitude bits to obtain
  the low order bits of the output ;
  set the high order bit of the output to the high order bit of the input
}
```

The combinational register-transfer-level hardware shown in Figure 1 implements this conversation algorithm. It consists of a 3-bit wide inverter, a 3-bit parallel adder, and a 2-1 3-way multiplexer with the sign bit of the input selecting one of the multiplexer inputs.

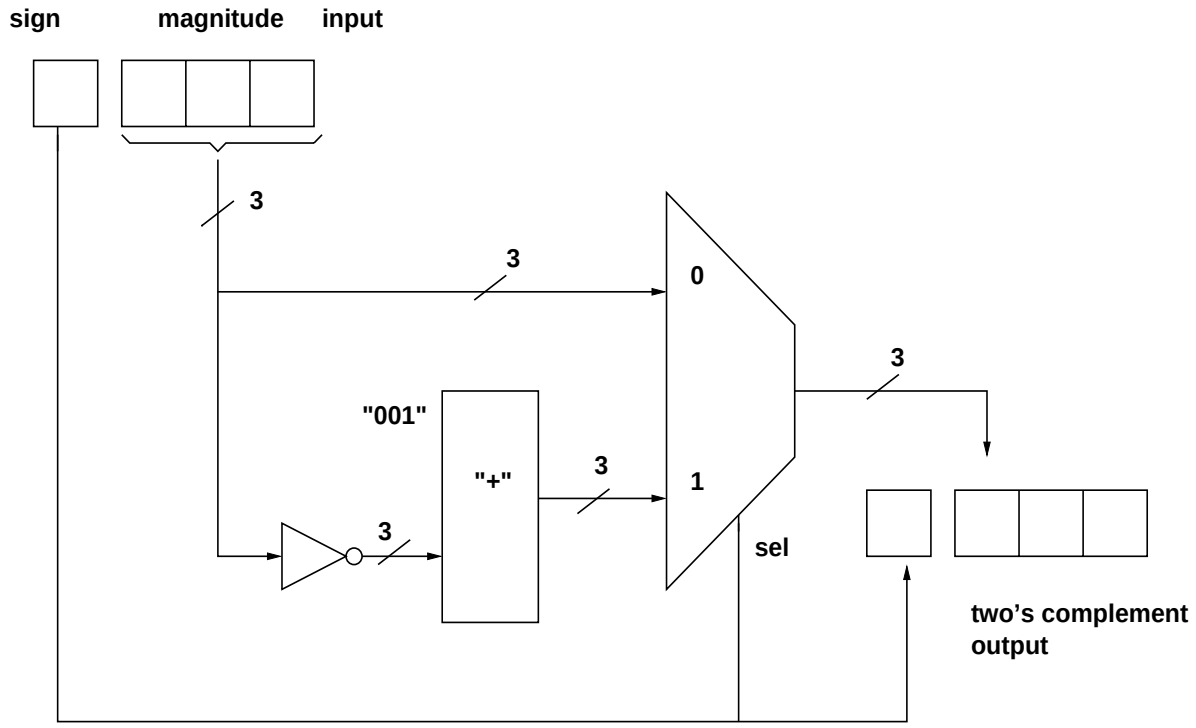

Figure 1: Sign-magnitude to two's complement converter circuit.

Design using a combinational VHDL process a sign magnitude to two's complement converter using the given algorithm and hardware description. Use the following VHDL entity specification:

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity converter           is
port( sign_mag  : in std_logic_vector(3 downto 0) ;
      twos_comp : out std_logic_vector(3 downto 0) );
end;
```

Some Hints

• Make use vector slices (array manipulation) and the concatenation operator to gain access to the magnitude bits and to form the output signal.

• You are free to make use of variables or to write the process making use of only signals.

Options to the `vcom` command

Most software tools used in engineering design are fairly complex and may be controlled by the user through the use of command line options. For example, the Modelsim simulator command vcom contains many such options (the first two labs did not make use of any of them). To obtain a list of all the available options, one may use the -help option as in:

```
ted@deadflowers Code 12:26pm >vcom -help
Model Technology ModelSim SE-64 vcom 6.6g Compiler 2012.05 May 23
2012
Usage: vcom [options] files
Options:
  -help              Print this message
  -time              Print the compilation wall clock time
  -version           Print the version of the compiler
  -32                Run in 32-bit mode
  -64                Run in 64-bit mode
  -work <path>       Specify library WORK
  -error <msgNumber>[,<msgNumber>...]
                     Report the listed messages as errors
  -warning <msgNumber>[,<msgNumber>...]
                     Report the listed messages as warnings
  -note <msgNumber>[,<msgNumber>...]
                     Report the listed message as notes
  -suppress <msgNumber>[,<msgNumber>...]
                     Suppress the listed messages
  -87                Enable support for VHDL 1076-1987 only
  -93                Enable support for VHDL 1076-1993
  -ams               Enable AMS wreal extensions
  -2002              Enable support for VHDL 1076-2002
  -check_synthesis   Check for compliance to some synthesis rules
```

(not all options are shown above as the list is long)

A helpful option is the `-check_synthesis` which checks that the VHDL code complies with synthesis rules. Writing VHDL meant for simulation purposes is different from writing VHDL code which is crafted to produce working hardware by a synthesis tool. Being aware of the differences between the two is the hallmark of a good digital designer. To paraphrase a well known song, "you can’t always synthesize what you simulate". For example, in a combinational process, all signals which are being read from within the process should be listed in the sensitivity list. The `-check_synthesis` option will detect the existence of any signals which are read within a process but do not appear in the process sensitivity list:

```
ted@deadflowers Code 4:11pm >vcom -check_synthesis
lab3_with_signal_bad.vhd
Model Technology ModelSim SE-64 vcom 6.6g Compiler 2012.05 May 23
2012
-- Loading package standard
-- Loading package std_logic_1164
-- Loading package std_logic_arith
-- Loading package std_logic_unsigned
-- Compiling entity converter
-- Compiling architecture using_signal of converter
** Warning: lab3_with_signal_bad.vhd(35): (vcom-1400) Synthesis
Warning: Signal "not_magnitude" is read in the process but is not
in the sensitivity list.
** Warning: lab3_with_signal_bad.vhd(40): (vcom-1400) Synthesis
Warning: Signal "adder_out" is read in the process but is not in
the sensitivity list.
```

In addition to simulation tool warnings, a designer should also be aware of any messages/warnings/errors generated by the logic synthesis tool. The Xilinx Vivado tools create several log files in the project directory. The GUI provides a method to view these reports within the GUI, alternatively the reports can be viewed (or printed) from the Linux command line. There are reports created during the synthesis and implementation stages. They are stored in the files:

`/home/t/ted/VIVADO/Lab3/bad/bad.runs/synth_1/is_this_good_ou_mauvaise.vds`

`/home/t/ted/VIVADO/Lab3/bad/bad.runs/impl_1/is_this_good_ou_mauvaise.vdi`

Note: The above are meant as examples. Your files would be stored in your home directories, within your Vivado project directories. In the above, `bad` was my chosen Project name, and `is_this_good_ou_mauvaise` was the name of the VHDL entity.

One should always refer to these report files and check for any informative messages, warnings, or errors. Typical informative messages include ones relating to RTL component inferencing such as the following:

```
Detailed RTL Component Info :
+---Adders :
           2 Input      3 Bit       Adders := 1
+---Registers :
                        3 Bit    Registers := 1
```

Some messages may be related to warnings which appear to be innocuous, but may lead to mismatch between simulation and hardware:

```
WARNING: [Synth 8-614] signal ’stone’ is read in the process but
is not in the sensitivity list
```

Error messages should be closely investigated. The tool often provides suggestions to correct the error and other possbilities:

```
ERROR: [DRC LUTLP-1] Combinatorial Loop Alert: 1 LUT cells form a
combinatorial loop. This can create a race condition. Timing
analysis may not be accurate. The preferred resolution is to mod-
ify the design to remove combinatorial logic loops. If the loop is
known and understood, this DRC can be bypassed by acknowledging
the condition and setting the following XDC constraint on any one
of the nets in the loop: ’set_property ALLOW_COMBINATORIAL_LOOPS
TRUE [get_nets <myHier/myNet>]’. One net in the loop is
keith_OBUF. Please evaluate your design. The cells in the loop
are: keith_OBUF_inst_i_1.
```

One should always be attentive of any warning/error messages and investigate the reason for their appearances.

Requirements

1. Modelsim simulation results for the design. Test your design for all possible input values.

2. RTL schematic diagrams of the elaborated and implemented circuit .

3. The synthesis and implementaion log files as produced by Vivado highlighting any messages/ warnings. Discuss the importance of any generated message or warning.

4. The VHDL code for your designs.

5. Download to the FPGA board and demonstrate to your TA the operation of your circuit.

Questions

1. What will result (during synthesis) if a signal appears on both sides of the signal assignment operator (<=) within a combinational VHDL process such as:

```
library IEEE;
use IEEE.std_logic_1164.all;

entity is_this_good_ou_mauvaise is
port( mick : in std_logic;
      keith : out std_logic);
end;
```

```
architecture rtl of is_this_good_ou_mauvaise is

signal stone : std_logic;

begin


process(mick)
begin
  stone <= mick and stone;
end process ;

keith <= stone ; -- tout le monde sais que Keith == stone

end ;
```

2. What will happen during simulation if a signal is read from within a combinational process but does not appear in the process sensitivity list?

3. If you made use of variable in your combinational process, rewrite the VHDL code such that the process makes use of only signals. If you originally made use of only signals, rewrite your VHDL code such that it makes use of variable(s). Simulate your new VHDL code to show that it gives the same simulation results. You do not have to re-synthesize. Comment on the salient differences between the code which uses only signals and the code which makes use of a variable.

T. Obuchowicz
Feb. 16, 2018

Revised: Feb. 14, 2020 for Vivado log files.