ECSE-323
Digital System Design

Lab #4 – *VHDL for Sequential Circuit Design* Fall 2008

Introduction

In this lab you will learn how to use VHDL to describe sequential logic circuits using process blocks.

Learning Outcomes .

After completing this lab you should know how to:

- Use process blocks in VHDL descriptions.
- Write VHDL descriptions of sequential circuits.

Table of Contents

This lab consists of the following stages:

1. VHDL design of the binary to base-100 converter
2. Simulation and testing of the converter on the Altera board.
3. Design of a circuit to implement an Nth root operation.
4. Simulation and testing of the Nth root circuit on the Altera board.
5. Writeup of the lab report

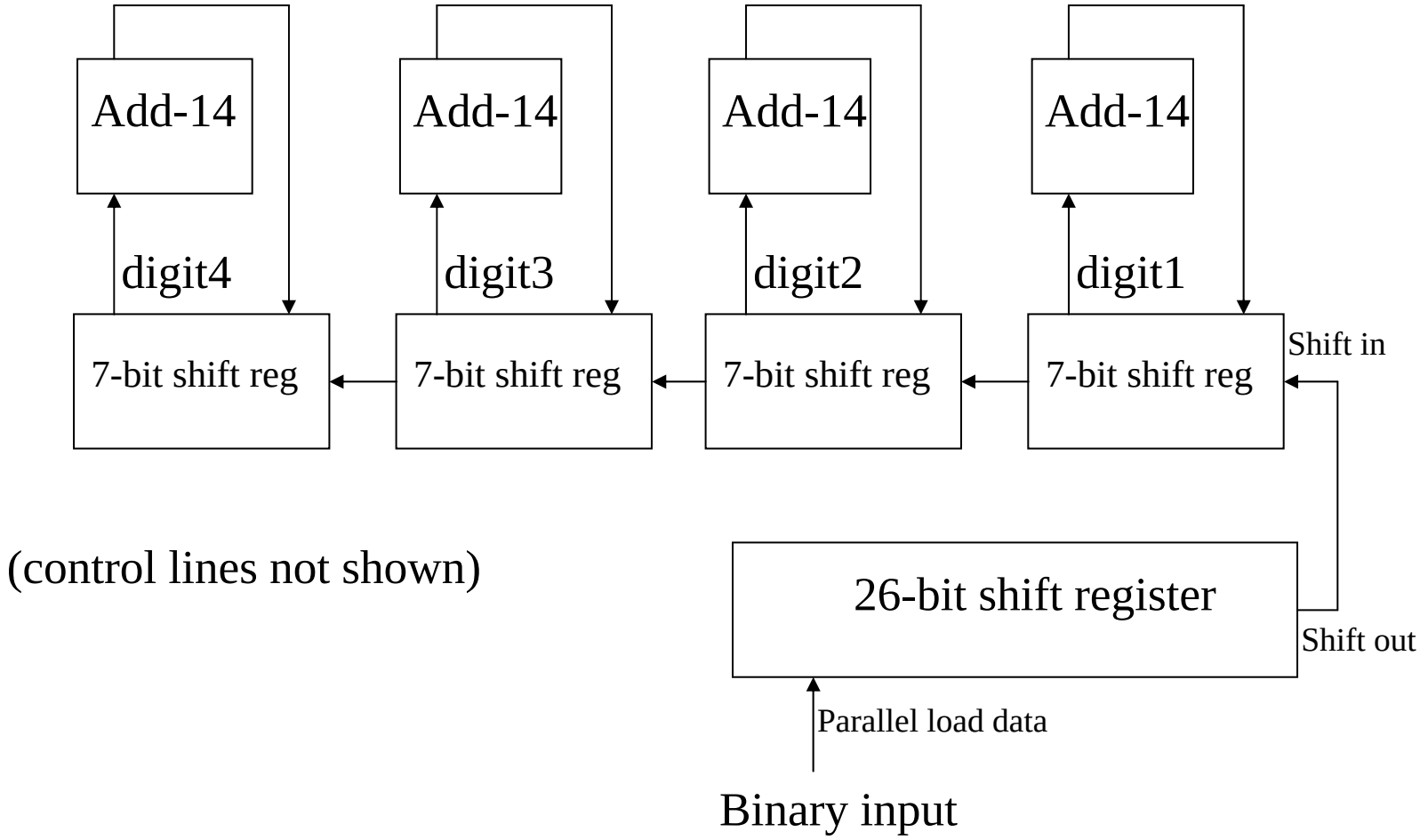
1. Design of a 26-bit Binary to Base-100 Converter .

For your project, you will need to display various binary values. But your alien bosses do not understand binary! So you will need to display your values in base-100 digits.

The following serial algorithm can be used to convert a binary number into a set of base-100 digits. It is called the “Add-14" algorithm:

1. Wait for START input to go low. When it does go low, go to step 2.
2. Wait for START input to go high. When it does go high, go to step 3.
3. Initialize all four base-100 digit registers to zero. Reset the cycle counter to zero (or to 25, if you want to count down). Set DONE output to low.
4. Load the input binary number into a (26-bit) shift register.
5. If any of the four base-100 digits has a value greater than 49, add 14 to that digit (ignoring any carry out), and load it back into the digit register.
6. Shift the shift register left by one place. The bit shifted out from the 26-bit register is shifted into the first base-100 digit register, the bit shifted out from the first base-100 digit register is shifted into the second base-100 digit register, etc.
7. Go to step 6 when all data has been shifted out of the 26-bit shift register (i.e. after 26 shift cycles). Otherwise loop back to step 4.
8. Set DONE output to high. Go to step 1.

1. Design of a 26-bit Binary to Base-100 Converter .



1. Design of a 26-bit Binary to Base-100 converter .

Use VHDL to describe the conversion circuit, using multiple process blocks as needed. The use of an FSM is recommended for controlling the operation of the circuit. The entity declaration for the circuit should be:

```
entity gNN_binary_to_base100 is
 port (  binary       : in std_logic_vector(25 downto 0);
         clk, reset   : in std_logic;
         START  : in std_logic;
         DONE   : out std_logic;
         DIGIT1 : out std_logic_vector(6 downto 0);
         DIGIT2 : out std_logic_vector(6 downto 0);
         DIGIT3 : out std_logic_vector(6 downto 0);
         DIGIT4 : out std_logic_vector(6 downto 0));
end gNN_binary_to_base100;
```

Show the TA your VHDL description of the circuit, explaining how it works.

[*You might want to think a little bit about why the add-14 algorithm works. A hint is that conversion to an arbitrary even base N requires adding (2^M-N)/2 (where M is the smallest integer such that 2^M>N) whenever the digit value is greater than or equal to N/2. This can be seen in the general Matlab code on the next page.*]

Matlab program for conversion of binary to an arbitrary even base

[Note: This is given for information only – you don't have to implement it!]

```
function bc=binary_convert(bin,BS)
%
% conversion from base-2 to base-BS representation
% BS must be an even number
% (c) James Clark, October 2008
%
NI = 32; % number of bits for input word (assume 32 bits)
NB = ceil(log2(BS)); % number of bits to represent the base
ND = ceil(NI/log2(BS)); % number of base-BS digits needed
%
bc = zeros(ND,1); % bc is an ND element array, initially cleared
for i = 1:NI, % shift bits out of bin 1 bit at a time
    for j = 1:ND,
       if  bc(j) > BS/2-1
           bc(j) = bc(j)+(2^NB-BS)/2;
       end
    end
    bctmp = bc;
    bc=bitshift(bc,1,NB);
    for j=2:ND,
        bc(j)=bitset(bc(j),1,bitget(bctmp(j-1),NB));
    end
    bc(1)=bitset(bc(1),1,bitget(bin,NI));
    bin=bitshift(bin,1,NI);
end
```

2. Simulation of the binary to base-100 converter .

Perform a timing simulation of the converter. Use a clock period of 500nsec for the initial simulation. Show the TA the results of your simulation. You should test at least 4 different input values, including all ones and all zeros.

How fast could you clock the circuit without modification? (hint: the clock rate will mainly be limited by the propagation delay of the add-14 operations).

For the overall system we will want to use the fastest clock available, which on the DE1 board runs at 50MHz. If this is too fast for your circuit you will have to modify the circuit to handle the faster clock. If you are using an FSM the easiest way to do this is to add in wait states (that do no operations, just take time) before the bit shifting states.

Re-run the simulations using the 50MHz clock on your modified circuit. Show the results to the TA.

TIME CHECK

You should be this far at the end of your *first* 2-hour lab period!

Interlude: Fixed Point Representations and Arithmetic .

The calculator system may need to deal with fractional values, and not just integer numbers. Fractions can be represented in binary fashion in the same way as integer numbers – as sums of powers of 2. In the case of fractions, the powers will be negative.

$$V = \ldots + v_{N-2}2^{N-2} + \ldots + v_2 2^2 + v_1 2^1 + v_0 2^0 + v_{-1}2^{-1} + v_{-2}2^{-2} + \ldots$$

For example the fractional number 16543.5625 is represented as:
16544.5625 = 1000000010100000.1001 since 0.5625 = 1/2+ 1/16 and
16544 = 2^14+2^7+2^5

Note that just because a fraction expressed in decimal (base-10) form has a finite number of decimal digits, this does not mean that the binary form always has a finite number of binary digits (bits) as well. For example, 0.1 in binary is 0.000110011001…

Interlude: Fixed Point Representations and Arithmetic .

You add and multiply fractional numbers just as you would with integers, but you have to keep track of where the "binary point" is. The binary point is located where the power of two in the base 2 expansion of the value goes from 0 to -1.

Adding two *N*-bit fractional numbers each with *M* fractional bits gives a result with *N+1* bits, still with the *M* fractional bits.

For addition we can treat the input operands as integers by multiplying the fraction values by 2^M where M is the number of fractional bits and then dividing the result by 2^M to shift it back to the proper value.

For example : 5.3125+3.8125 = 9.125
101.0101+ 011.1101 = (1010101+0111101)/2^4 = 1001.0010

Interlude: Fixed Point Representations and Arithmetic .

Multiplying two *N*-bit fractional numbers each with *M* fractional bits gives a result with 2*N* bits, but now with 2*M* fractional bits.

As in addition we can treat the input operands as integers by multiplying each of them by 2^M, where M is the number of fractional bits, and then dividing the result by 2^(2M) to shift it back to the proper value.

For example : 5.3125*3.8125 = 20.25390625
101.0101* 011.1101 = (1010101*0111101)/2^8 = 010100.01000001

Now, you don't actually have to do this shifting, you just need to keep track of where to place the binary point after the computation (e.g. for display purposes).

Often you will want to use "***fixed-point***" representation, where you always keep a constant number of fractional bits. In the above multiplication example, we would only keep 4 of the fractional bits, to give 010100.0100 = 20.25

Converting Fractional Binary to Base-100 .

Conversion of fractional numbers (numbers strictly less than one) to Base-B is actually easier than converting integer numbers. To extract the most-significant digit, one needs to multiply the number by B (e.g. B=100 for base-100 numbers) and take the integral part of the result. Then, to get the next most-significant base-B digit, take the fractional part of the result and multiply by B again, taking the integral part of the result as the next digit. Continue this until you have enough fractional base-B digits.

If you have a number with both an integer part and a fractional part you can combine the above procedure with the integer conversion described earlier. But this will be rather cumbersome to control, since you have two different operations going on.

A more convenient (but slower) approach to conversion of a general number is to multiply the number by B^M, where M is the number of fraction digits you want to obtain. After multiplication, discard all of the fractional bits of the result. This will leave just an integer, which you can convert using the approach described earlier.

For example, to convert 10100.01000001 to base-10, keeping 3 decimal digits, we would multiply by 10^3=1111101000 to give 100111100011101 = 20253

4. Testing of the binary to base-100 converter .

You will find, in general, that just because a system works in simulation it is not guaranteed to work when downloaded to the Altera board. This is especially true for sequential systems, as your design may have timing issues that prevent it from working properly.

So, you should design a testbed for the binary to base-100 converter circuit. This should use the 4 LEDs to display the 4 base-100 digits. Use the dipswitches on the Altera board to provide the input to the converter circuit. *Use your ingenuity in figuring out how to use the 10 switches available on the DE1 board to load in a 26-bit binary number!*

Use VHDL to describe the testbed circuit, using components to connect the various modules.

Compile and download the design to the Altera board, and demonstrate its functioning to the TA, switching between different input patterns.

TIME CHECK

You should be this far at the end of your *second* 2-hour lab period!

4. Design of an Nth root Calculator .

For your alien financial calculator system a circuit that computes the Nth root of an *M*-bit number will be needed.

$$Y = \sqrt[N]{X}$$

To compute this, you can use the following binary search algorithm:

1. Initialize the bit counter (*BC*) to *M* and set *DONE = 1 (reset state)*
2. Wait for *START* to go low
3. Wait for *START* to go high
4. Set *DONE = 0*, initialize *Y* to zero
5. set bit *BC* of *Y* to 1 *(bit 1 is the LSB, bit M is the MSB)*
6. compute *Z=Y^N*, by repeated multiplication. If *Z>X*, *at any time*, reset bit *BC* of *Y* to 0 and go to step 7
7. *BC = BC-1*
8. If *BC > 0* go back to step 5, else go back to step 1

Make the following assumptions:

- $1 \leq N \leq 63$ (6 bits are needed to represent N)
- X is an integer between 0 and 99. (so only 7 bits are needed for X)

Any other values for N and X should result in the ERROR output being set high.

Y will not be an integer in general, so provide extra bits to represent its fractional part. The integer part of Y will be at most 99 so only 7 bits are needed for the integer part. Use 14 bits for the fractional part, for a total of 21 bits. When displayed Y will have one base-100 digit for the integer part and two base-100 digits for the fractional part.

It may seem that we would need, in the worst case, 7*63+14=455 bits to represent Z, because of the repeated multiplications in step-6. But, because of the check to see if $Z>X$, the repeated multiplications will be interrupted long before Z grows to be that large. So, you can use multipliers that have 21 bits for the input operands (14 of these fractional) and 42 bits for the output (14 for the integer part and 28 for the fractional part). You can then discard the lower 14 bits and the upper 7 bits.

Design the nth_root circuit using the Datapath/Controller approach, and use VHDL to describe it. The circuit should be called *gNN_Nth_root* and have the following inputs: *X, N, START, clock, reset*; and the following outputs: *Y, DONE*.

Remember to add in wait states where necessary. For example, the multipliers will probably have propagation delays longer than the clock period, so you need to wait until sufficient time has passed before making use of the multiplier outputs. The comparison operation ($Z>X$) may also be slow. You can determine the propagation delay of your datapath elements by simulation (just simulate the modules by themselves).

Once you have finished your VHDL description of the Nth_root circuit, show it to the TA and explain its design.

TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *third* 2-hour lab period!

5. Simulation of the Nth_root Circuit .

Next, compile the circuit and create a symbol for it, then insert the symbol into a new empty schematic. Then perform a *timing* simulation of the circuit.

Test the circuit by simulating it a number of different input values (trying various values of X and N). Compare your results to those provided by a calculator.

Observe how many clock cycles pass before the computation finishes. Does this number depend on the input values? What is the worst case time for completion of the operation?

Show your simulations to the TA. 

6. Testing of the Nth_root Circuit on the DE1 Board .

Create a testbed for the Nth_root circuit which displays its output (in base-100 digits) on the 7-segment LEDs of the DE1 board. Use the switches on the DE-1 board to set the test input values. Turn on the decimal point indicator of one of the LEDs to indicate the location of the decimal point in front of the two fractional digits.

Note that, as mentioned earlier in this document, in order to properly display the two fractional base-100 digits on the 7-segment LEDs, you will have to multiply the output of the Nth_root circuit by 10,000 (100^2) before feeding it into the binary_to_base100 circuit.

Compile the testbed circuit and download the programming file to the DE-1board.

Demonstrate to the TA the operation of the Nth_root circuit on the DE-1 board. You may have difficulties getting the circuit to operate properly. If so, the most likely reason is timing-related, probably because you did not provide enough time between successive multiplication or comparison operations. Or you may have the wrong sequence of events being generated by your controller.

TIME CHECK

You should be this far (i.e. have completed the lab) at the end of your *fourth* 2-hour lab period!

7. Writeup of the Lab Reports .

Write up the reports for the *gNN_binary_to_base100* and *gNN_nth_root* circuits.

The reports must include the following items:

- A header listing the group number (and company name if you gave it one), the names and student numbers of each group member.
- A title, giving the name (e.g. ***gNN_Nth_root***) and function of the circuit.
- A description of the circuit's function, listing the inputs and outputs. Provide a pinout or symbol diagram.
- The VHDL description of the circuit (don’t put this in the report itself, but provide the .vhd files).
- A discussion of how the circuit was tested, giving details of the testbed and showing representative simulation plots.
- A summary of the timing performance of the circuit, giving the timing analysis and the simulated propagation delays.
- A summary of the FPGA resource utilization (from the Compilation Report's Flow Summary).

The reports should be done in html or pdf (preferred), or in Microsoft Word, and submitted to the WebCT site .

Make sure that you have uploaded ***all*** of the design files (e.g. .bdf and .vhd files) used in your project. These should be included with your reports in a single zip file.

The reports are due one week after the last day of the lab period, or Friday, November 21.

Grade Sheet for Lab #4

Fall 2008.

Group Number: _____.

Group Member Name: _____. Student Number: _____.

Group Member Name: _____. Student Number: _____.

Marks			
	1.	VHDL description of the binary to base-100 converter circuit	.
	2.	Simulation of the binary to base-100 converter circuit	.
	3.	Testing of the binary to base-100 converter circuit	.
	4.	VHDL description of the Nth root circuit	.
	5.	Simulation of the Nth root circuit	.
	6.	Testing of the Nth root circuit on the Altera board	.
			TA Signatures

Each part should be demonstrated to one of the TAs who will then give a grade and sign the grade sheet. Grades for each part will be either 0, 1, or 2. A mark of 2 will be given if everything is done correctly. A grade of 1 will be given if there are significant problems, but an attempt was made. A grade of 0 will be given for parts that were not done at all, or for which there is no TA signature.