Lab 3: VHDL Design and Optimization

Due: Tuesday, Feb 26, 2008 at Noon

In this third lab you will build a module to simulate data decryption. Your module (the Decrypter module) will interact with two other modules (provided) to simulate a complete decryption system. One module will be the generator module; this module will supply data for your module to decrypt. The second module is the checker module; this module will perform the check of your decrypter data to make sure it was decrypted correctly. Coordinating the interaction of all these modules is the testbench module (provided).

Your decrypter module will interact via the other modules with the port list shown below:

- clk : in std_ulogic;
- reset_n : in std_ulogic;
- databus : inout std_logic_vector(31 downto 0);
- controlbus : inout std_logic_vector(3 downto 0);

A description of these ports are:
- **reset_n**
  - if at any time your design sees the reset_n input transition to a LOW value, then your design should reset itself to a reset state and stay in the reset state until reset_n is no longer low

- **databus**
  - the databus is primary means of transferring data between the three modules, it is a shared bus and therefore putting information on the bus must be done correctly or else conflicts will arise.

- **controlbus**
  - the controlbus is the means to read and write different control commands to control the other modules’ execution.

The actual data to be decrypted consists of 256 bits. The data is referenced in 32 bit increments for decryption and transferred across the databus 32 bits at a time. The algorithm for decryption is shown below:

Data for decryption = 256 bits
Decrypt data in 32 bit chunks, therefore we can consider our data as an array of 8 positions each holding 32 bits of data.
We can represent this in VHDL as:

```
type dataMem is array (7 downto 0) of std_ulogic_vector(31 downto 0);
```
Decryption Algorithm:
For each index of our array 0 to 7 do
\[
dataArray(index) = (dataArray(index) \times 7 + invert(dataArray(index))) \ll \text{“5 least significant bits of dataArray(index)”}
\]
End loop;

where
\text{invert}: to invert each individual bit of our current index
\text{ex: } 00000000000000000000000000000000 \text{ become } 11111111111111111111111111111111

\ll : a CIRCULAR shift left, note that for our data type the maximum number of shifts that can occur is 32, we decide how much to shift by looking at the 5 least significant bits \((2^5 = 32)\) of our current array index data
\text{ex: } 11111110000000000000000000000000 \text{ shifted circularly left by 6 becomes } 10000000000000000000000000000000

To properly receive the data to decrypt and transfer the decrypted data your module must properly communicate with the other modules. The sequence of controlbus messages and their meanings for proper operation are shown below. By following this order shown below your module will be able to communicate with the other two modules:

<table>
<thead>
<tr>
<th>Control Bus Bit Sequence</th>
<th>Module sending message</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>Decrypter Module</td>
<td>I want the data to process</td>
</tr>
<tr>
<td>0010</td>
<td>Generator Module</td>
<td>I am ready to send data</td>
</tr>
<tr>
<td>0011</td>
<td>Decrypter Module</td>
<td>Send me the data</td>
</tr>
<tr>
<td>0100</td>
<td>Generator Module</td>
<td>I am sending data</td>
</tr>
<tr>
<td>0101</td>
<td>Generator Module</td>
<td>I am finished sending the data</td>
</tr>
<tr>
<td>0110</td>
<td>Decrypter Module</td>
<td>I am ready to send the decrypted data</td>
</tr>
<tr>
<td>0111</td>
<td>Checker Module</td>
<td>Send me the decrypted data</td>
</tr>
<tr>
<td>1000</td>
<td>Decrypter Module</td>
<td>I am sending decrypted data</td>
</tr>
<tr>
<td>1001</td>
<td>Decrypter Module</td>
<td>I am finished sending the decrypted data</td>
</tr>
<tr>
<td>1010</td>
<td>Checker Module</td>
<td>I am finished checking the decrypted data</td>
</tr>
</tbody>
</table>

\text{Synthesis:}
\text{NOTE: You only need to synthesize your decrypter module, DO NOT synthesize any of the other modules!!}
Hint and Tips:

1. Look at the code of the Generator and Checker modules to see how to structure the design of your module.
2. Use two separate processes, one which will handle all state machine transitions and the other to do the actual work depending on the current state that is active. Again see the other modules for an example.
3. To not write anything on the bus set it to high impedance which is represented in VHDL by ‘Z’ (ex. controlbus <= “ZZZZ”)
4. Write your decryption algorithm operators (<< and invert) in functions to make it easier to code and understand your design!
5. Since the number of cycles to complete decryption is low, try reducing your minimum clock period by making your datapath shorter; this can be done by using two cycles per decryption. With pipelining this two cycle latency can be made even less.
6. Try to start sending data as soon as it is decrypted instead of waiting for all of the decryption to complete.

Lab #3 Receivables:
1. Write a paragraph explaining your design and how it operates and reacts to the different input signals. Include a drawing of your state machine and its transitions.
2. Provide a copy of your decrypter module VHDL source code.
3. Provide a copy of the ModelSim transcript file which contains the output remarks for the supplied Test Bench file given on the lab website (this will test your design for correctness).
4. Provide a copy of the Synopsys synthesis script or a transcript of the commands you typed for synthesis, write a paragraph and give a table showing any intermediate synthesis results (area, timing).
5. Provide your final synthesis results – final clock period, final total area, final total number of cycles needed to complete processing (given from the test bench).
6. Provide a copy of the Synopsys area, timing reports (see the website for details on how to do this).

Lab #3 Grading Criteria:
Besides correctness of the design, part of your total Lab #3 score will be determined by how well in relation to your classmates the overall performance-area of your design is. So aim to optimize your design to reduce cycles to get a better performance number. Also, by tweaking your synthesis script you can also lower this number.

Performance Area = Area * Clock Period * clock cycles needed for completion

Points Breakdown:
60% - Your design works correctly
10% - You included all the relevant files
30% - Your relative performance area