Wednesday, April 6, 2005

SOLUTIONS!!!

Name (print): ________________________________________________________________

Signed: _____________________________________________________________________

I pledge on my honor that I have neither given nor received aid on this work.

GENERAL INSTRUCTIONS

1. Do not open this exam until instructed to do so.

2. This is a closed book, closed notes exam. Use a pencil to complete your work. Follow the instructions as given in the problem. Calculators ARE allowed for the exam.

3. Please do all of your work on the exam itself. You may use the backs of the pages, if necessary. Clearly indicate the continuation of your work on other pages.

4. Show work for maximum partial credit.

5. Please be as neat and as well-organized as possible.

6. Clearly indicate your answers.

<table>
<thead>
<tr>
<th></th>
<th>MAXIMUM</th>
<th>SCORE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>30</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>
**PROBLEM 1:** (45 pts)

A sequential circuit has two inputs, \( w \) and \( x \), and one output, \( z \). Its function is to compare the input sequences on the two inputs. If \( w \) equals \( x \) during any four consecutive clock cycles, the circuit produces \( z \) equal ‘1’; otherwise, \( z \) equals ‘0’. For example:

\[
\begin{align*}
  w & : 0 \ 1 \ 1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \\
  x & : 1 \ 1 \ 1 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \\
  z & : 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0
\end{align*}
\]

*Part A (5 pts)*

Does this finite state machine (FSM) correspond to a Mealy machine or a Moore machine? Give a reason why you chose your answer.

It corresponds to a Mealy machine. The FSM must assert the output signal in the same cycle in which it detects the four equivalent, consecutive inputs. A Moore machine would assert the output one cycle later.

*Part B (20 pts)*

Give a suitable state transition graph or state table that implements the described functionality from above. Label your states A, B, C, etc.

To compare individual bits, let \( k = w \ XOR \ x \). Then the corresponding state diagram is:
To compare individual bits, let \( k = w \ \text{XOR} \ x \). Then the corresponding state table is:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>Output z</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( k = 0 )</td>
<td>( k = 1 )</td>
</tr>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>A</td>
</tr>
</tbody>
</table>

**Part C (20 pts)**

Write the corresponding VHDL code for the architecture of your finite state machine (FSM) from **Part B**. The entity declaration is given for you below. Your FSM should operate on the falling edge of the clock and include an asynchronous reset signal that is active low. Use the following sheet of paper if necessary.

```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY Exam2_FSM IS
  PORT ( Clock  : IN STD_LOGIC;
         Resetn  : IN STD_LOGIC;
         w, x    : IN STD_LOGIC;
         z       : OUT STD_LOGIC);
END Exam2_FSM;
```
ARCHITECTURE Behavior OF Exam2_FSM IS

    TYPE State_type IS ( A, B, C, D ) ;

    SIGNAL y : State_type ;
    SIGNAL k : STD_LOGIC ;

BEGIN

    k <= w XOR x ;

    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Clock'EVENT AND Clock = '0') THEN
            CASE y IS
                WHEN A =>
                    IF k = '0' THEN y <= B ;
                    ELSE y <= A ;
                    END IF ;
                WHEN B =>
                    IF k = '0' THEN y <= C ;
                    ELSE y <= A ;
                    END IF ;
                WHEN C =>
                    IF k = '0' THEN y <= D ;
                    ELSE y <= A ;
                    END IF ;
                WHEN D =>
                    IF k = '0' THEN y <= D ;
                    ELSE y <= A ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS ;

    z <= '1' WHEN y = D AND k = '0' ELSE '0' ;

END Behavior ;
PROBLEM 2: (25 pts)

Part A (5 pts)
FPGAs have an abundance of registers and can take advantage of the one-hot encoding method. The following table presents an alternative method to encode the state variable. This method is very similar to a one-hot encoding scheme and is utilized by the synthesis tool for state assignment. What advantage would this pseudo-one-hot encoding scheme have over the regular one-hot encoding?

<table>
<thead>
<tr>
<th>State</th>
<th>One-Hot Encoding</th>
<th>Pseudo-One-Hot Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10000</td>
<td>00000</td>
</tr>
<tr>
<td>B</td>
<td>01000</td>
<td>11000</td>
</tr>
<tr>
<td>C</td>
<td>00100</td>
<td>10100</td>
</tr>
<tr>
<td>D</td>
<td>00010</td>
<td>10010</td>
</tr>
<tr>
<td>E</td>
<td>00001</td>
<td>10001</td>
</tr>
</tbody>
</table>

A finite state machine (FSM) will use a reset signal to place the machine in a known state. By using the pseudo-one-hot encoding scheme, this reset signal (or its complement) can be tied directly to the reset input of all the registers. In the above example, State A is the reset state. For the normal one-hot encoding scheme, this state would still require a logic circuit to be assigned.

Part B (5 pts)
Consider the VHDL code given below. What type of circuit does the code represent? Comment on whether or not the style of code used is a good choice for the circuit that it represents.

```vhdl
ARCHITECTURE behavior OF Problem2_B IS
BEGIN
    dout <= b WHEN (input = "001") ELSE
            c WHEN (input = "010") ELSE
            d WHEN (input = "011") ELSE
            e WHEN (input = "100") ELSE
            f WHEN (input = "101") ELSE
            g WHEN (input = "110") ELSE
            h WHEN (input = "111") ELSE
            a;
END behavior;
```

This VHDL code represents an 8:1 multiplexer. The code is not a good choice for a multiplexer. The WHEN-ELSE construct implies priority; however, a multiplexer is just a simple selection. Since WHEN-ELSE is a concurrent statement, you should use the SELECT statement for this circuit.
Part C (5 pts)
How does IEEE Standard 1149.1 (JTAG) improve the testability of devices?

The IEEE Standard 1149.1 (JTAG) provides a standard for scan path support. Scan paths connect all internal registers into a shift register to improve controllability and observability of circuitry within the device. JTAG can be used to support fault detection at the single chip level and detection of board level faults.

Part D (5 pts)
For the following circuit, give a test vector that will detect a “stuck at 1” fault on Node 2.

A “stuck at 1” fault means that the node has a value of ‘1’ even when the correct output for the preceding logic circuit should equal ‘0’. In order to observe the output of Node 2 at f, the input D must be set to ‘0’. By setting the input C to ‘0’, the AND gate should produce a ‘0’ on Node 2. The inputs A and B would be “don’t care” in this case:

\[ ABCD = \text{“xx00”} \]

Part E (5 pts)
Name a vendor of Field-Programmable Gate Arrays (FPGAs).

Altera, Xilinx, or Actel are examples of acceptable responses.
PROBLEM 3: (30 pts)

Design a 32-bit ALU for the MIPS datapath which implements the functionality in the table below. Unspecified opcodes result in assigning the A input directly to C. The ALU should also set the Zflag if the result equals zero. Give VHDL code for both the entity declaration and the architecture of your design. The library specification is given for you below. Use the following sheet of paper if necessary.

<table>
<thead>
<tr>
<th>ALU Opcode</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>AND</td>
</tr>
<tr>
<td>0001</td>
<td>OR</td>
</tr>
<tr>
<td>0010</td>
<td>add (A + B)</td>
</tr>
<tr>
<td>0110</td>
<td>subtract (A – B)</td>
</tr>
<tr>
<td>1100</td>
<td>NOR</td>
</tr>
</tbody>
</table>

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY Exam2_Problem3 IS
    PORT ( opcode : IN STD_LOGIC_VECTOR (3 DOWNTO 0);
           A, B : IN STD_LOGIC_VECTOR (31 DOWNTO 0);
           Zflag : OUT STD_LOGIC;
           C : OUT STD_LOGIC_VECTOR (31 DOWNTO 0) );
END Exam2_Problem3;
ARCHITECTURE Behavior OF Exam2_Problem3 IS

SIGNAL Result : STD_LOGIC_VECTOR (31 DOWNTO 0);

BEGIN

PROCESS ( A, B, opcode )
BEGIN
CASE opcode IS
    WHEN "0000" =>
        Result <= A AND B;
    WHEN "0001" =>
        Result <= A OR B;
    WHEN "0010" =>
        Result <= A + B;
    WHEN "0110" =>
        Result <= A - B;
    WHEN "1100" =>
        Result <= A NOR B;
    WHEN OTHERS =>
        Result <= A;
END CASE;
END PROCESS;

Zflag <= '1' WHEN Result = "000000000000000000000000000000000000000000" ELSE '0';

C <= Result;

END Behavior ;