cs281: Computer Organization

Lab3

1 Combinational Logic Design

Combinational Logic Design refers to the use of logic gates that, when combined, perform some boolean function. The boolean function results in output(s) that are dependent only on the input values and are independent of previous input values or states. In other words, the current inputs completely determine the output(s).

The major design objective is usually to minimize the cost of the hardware needed to implement a logic function. That cost can usually be expressed in terms of the number of gates, although for technologies such as programmable logic, there are other limitations, such as the number of terms that may be implemented.

The design process generally proceeds through the following steps:

1. We begin with a description of the boolean function. This description should clearly identify the boolean input variables and the set of outputs, each of which must be a single boolean variable. The description may take the form of an informal English description, or could use some more formal specification.

2. From the description, the next step is to generate a truth table that enumerates all possible values of the input variables. So if there are n boolean inputs, the complete truth table will have 2^n rows, and these are normally written in ascending binary value order. With this standard ordering, we can unambiguously refer to the i^{th} row of the truth table. Sometimes, the process begins at this step by specifying the boolean function in terms of its truth table.

3. From the truth table, we can, for each output, express the boolean output variable as equal to a boolean algebra expression in one of two canonical forms: the *sum of products* form, or the *product of sums* form.

4. From the canonical boolean expression form, one option to yield a lower cost hardware implementation is to use boolean identities to simplify the boolean expression. However, this method can be difficult, slow, and error-prone.

5. Alternatively, we can rewrite the truth table (for functions where the number of boolean inputs ranges from 2 through 5) into an equivalent form called a *Karnaugh-map* (or *K-map* for short). Using this graphical method, we can determine equivalent simpler terms for either the sum-of-products form or the product-of-sums form.

6. From the simplified equation for each output, we can then translate the boolean expression into **and**, **or**, and **not** logic gates.

7. In many technologies, implementation of **nand** gates or **nor** gates is easier than that of **and** and **or** gates. Any logic function can be realized using only **nand** gates or only **nor** gates, and conversion from **and**/**or** gates to these alternative forms is straightforward. This last step will generally not be pursued in this class.

Assume, consistent with the above design description, that we are designing a piece of conbinational logic with a number of input variables and a single output.

A *minterm* is defined as a boolean **and** function containing exactly one instance of each input variable or its inverse. A *maxterm* is defined as a boolean **or** function with exactly one instance of each variable or its inverse. For a combinational logic circuit with n input variables, there are 2^n possible minterms and 2^n possible maxterms. If the logic function is *true* (has value 1) at row i of the standard truth table, the i^{th} minterm exists and is designated by m_i. If the logic function is *false* (has value 0) at row i of the standard truth table, the i^{th} maxterm exists and is designated by M_i. For example, the table below defines a logic function with inputs A, B, and C and output Z. The final column in the table shows the minterms and the maxterms for this particular function.

row	A	B	C	Z	min/max
0	0	0	0	1	m_0
1	0	0	1	1	m_1
2	0	1	0	0	M_2
3	0	1	1	0	M_3
4	1	0	0	0	M_4
5	1	0	1	1	m_5
6	1	1	0	0	M_6
7	1	1	1	1	m_7

The logic function may be described by the logical OR or its minterms:

$$Z = m_0 + m_1 + m_5 + m_7$$

A function expressed as a logical OR of distinct minterms is in *sum-of-products* (SOP) form, so expanding the minterms, we get

$$Z = \overline{A} \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot \overline{B} \cdot C + A \cdot B \cdot C$$

Each variable is inverted if there is a corresponding 0 in the truth table and not inverted if there is a 1. The shorthand notation for the sum-of-products (i.e. the sum of minterms) is $Z = \sum m(0, 1, 5, 7)$.

Similarly, the logic function may be described by the logical AND of its maxterms:

$$Z = M_2 \cdot M_3 \cdot M_4 \cdot M_6$$

A function expressed as a logical AND of distinct maxterms is in *product-of-sums* (POS) form, so expanding the maxterms, we get

$$Z = (A + \overline{B} + C) \cdot (A + \overline{B} + \overline{C}) \cdot (\overline{A} + B + C) \cdot (\overline{A} + \overline{B} + C)$$

Each variable is inverted if there is a corresponding 1 in the truth table and not inverted if there is a 0. The shorthand notation for the product of sums (i.e. the product of maxterms) is $Z = \prod M(2, 3, 4, 6)$.

We define an *implicant* as a single term that covers at least one true value and no false values of a function. For example, the function $Z = A + \overline{A} \cdot \overline{B}$ is shown in the table below:

row	A	B	Z	min/max
0	0	0	1	m_0
1	0	1	0	M_1
2	1	0	1	m_2
3	1	1	1	m_3

The implicants of this function are $A \cdot B$, A, $\overline{B}$, $\overline{A} \cdot \overline{B}$, and $A \cdot \overline{B}$. The non-implicants are $\overline{A}$, B, $\overline{A} \cdot B$.

A *prime implicant* is an implicant that covers one or more minterms of a function, such that the minterms are not all covered by another single implicant. In the example above, A and $\overline{B}$ are prime implicants. The other implicants are all covered by one of the two prime implicants. An *essential prime implicant* is a prime implicant that covers an implicant not covered by any other prime implicant. Thus, A and $\overline{B}$ are essential prime implicants.

1.1 Karnaugh Maps

A Karnaugh map is effectively another way to write a truth table. See your instructor for a discussion of the creation of K-maps.

2 Problems

For each of the following boolean logic functions, use K-maps (by hand) to design a minimal logic circuit to realize the function. Deliverables for each include (a) the truth table, if not already given, (b) the K-map in standard form with all prime implicants circled, (c) a simplified boolean expression for the output in terms of the essential prime implicants, and (d) a realization of the circuit using LogiSim.

1. In general, a *multiplexor* (mux) is a device that performs *multiplexing*; it selects one of multiple input lines and forwards the selected input along to the output. The simplest multiplexor has two input lines and one line called the *selector* that selects which input line should have its value as the output. So a digital 2-1 mux defines a logic function with three inputs, A and B for the two input lines and S for the selector. When the value of S is 0, the output of the function is the same as the current value of the A input. When the value of S is 1, the output of the function is the same as the current value of the B input.

2. Given two 2-bit words A and B (consisting of a_1a_0 and b_1b_0) as input, the boolean function f should yield 1 whenever A is strictly less than B when the words are interpreted as unsigned binary integers and should yield 0 otherwise.

3. Use the following truth table to define the boolean function:

row	A	B	C	D	Z
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1