Digital Logic Design

This lecture reviews combinational and sequential logic design and shows how state machines can be implemented as digital logic circuits.
After this lecture you should be able to design a combinational or sequential logic circuit from a description of its behaviour.

Applications for Logic Circuits

Some control applications may require the design of custom digital logic circuits. In this course we will only cover the design of relatively simple circuits. Some examples of situations where the design of logic circuits is required are:

- when the controller is too simple for a microcomputer implementation. For example, a simple alarm system may only requires a flip-flop and a few logic gates.
- when it is necessary to implement operations that cannot be done fast enough under computer control. For example, a computer could not count events happening at a rate of 100 MHz.
- when "glue" logic is required to interface a peripheral IC chip to the microprocessor. For example, address decoders and buffers are usually required to attach an analog-to-digital converter chip to a microprocessor's data bus.

Combinational Logic

A combinational logic circuit is one where the output depends only on the current input and not on past inputs. Therefore we can determine the output simply by considering the current input. We will learn three ways to represent combinational circuits and how to convert between them.

The first type of description is a truth table. A truth table is a table that enumerates all of the possible inputs and the corresponding outputs.

The second representation is as an equation using boolean variables and operators to define the value of each output variable as a function of the input variables. We can also use boolean algebra to simplify the resulting equations.

The final type of description is a circuit diagram (typically called a "schematic") that shows the interconnection of hardware logic gates. A gate is a circuit that implements a boolean logic function such as "and" or "nor".

Truth Tables

A truth table is simply a table showing the value of each output for each possible combination of the input variables.

For example, the truth table for a circuit with 3-inputs (labelled a, b, c), and two outputs (x and y) might begin as follows:

a	b	c	x	y
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0

This particular truth table is for a device that outputs (in binary) the number of '1's in the input.

Exercise 31: Complete the table.

Boolean Algebra

It is also possible to write an algebraic expression for each output variable as a function of the input variables. In boolean algebra we use variables which can take on values of true or false. Typically true is represented as the value one or a high voltage and false as zero or a low voltage. However the opposite convention ("active low") logic is also common.

The logical AND function is expressed as implied multiplication (sometimes as a dot ($a \cdot b$) and logical OR as addition ($a + b$). The notation $\overline{\text{a}}$ is used for the logical complement (NOT) of a.

Sum of Products Form

From the truth table for a combinational circuit we can write an expression for an output as a function of the input variables.

One way to do this is as a "sum of products" (an "OR" of "AND"s). There will be one term in this sum for each line of the truth table in which the output variable has the value one. Each term is the product of each of the input variables – either the input variable (if that variable is 1 in that row) or its complement (if that variable is 0).

For example, the variable x above takes on a value of 1 in two lines (the fourth and sixth lines) so there would be two terms. The first term corresponds to the case where the input variables are $a = 0$, $b = 1$ and $c = 1$. So the term is $\overline{\text{a}}bc$. Note that this product will only be true when a, b and c have the desired values, that is, only for the combination of inputs on the fourth line.

If we form the other terms in the sum in a similar way (one term for each of the other lines where the desired output variable takes on the value one) we will have an expression that evaluates to '1' only for those lines and will evaluate to zero in all other cases.

Exercise 32: Write out the expressions for the two variables in the table above (assume the output is zero for the input conditions that are not shown).

Logic Identities

Having written down the expression for the desired output we can use a number of boolean logic identities to simplify the expression. This is normally used to simplify the resulting hardware implementation.

There are a number of basic identities that can be obtained by inspection:

$$ABC = (AB)C = A(BC)$$
$$AB = BA$$
$$AA = A$$
$$A1 = A$$
$$A0 = 0$$
$$A(B+C) = AB + AC$$
$$A + AB = A$$
$$A + BC = (A+B)(A+C)$$
$$A + B + C = (A+B) + C = A + (B+C)$$
$$A + B = B + A$$
$$A + A = A$$
$$A + 1 = 1$$
$$A + 0 = A$$
$$\overline{1} = 0$$
$$\overline{0} = 1$$
$$A + \overline{\text{A}} = 1$$
$$A\overline{\text{A}} = 0$$
$$\overline{\overline{\text{A}}} = A$$
$$A + \overline{\text{A}}B = A + B$$

There are also two useful relations called DeMorgan's theorem:

$$\overline{\text{(A+B)}} = \overline{\text{A}}\overline{\text{B}}$$
$$\overline{\text{AB}} = \overline{\text{A}} + \overline{\text{B}}$$

In practice, computer software is used to perform these minimizations in all but the simplest cases.

Exercise 33: Simplify the logic expressions for x and y.

Logic Gates

Having simplified the algebraic form of the combinational logic circuit we can then proceed to draw a circuit diagram using logic gates that will implement the desired function. These logic gates are available in the form of integrated circuits. Chips are available to implement all of the common boolean logic operations (AND, OR, NAND, NOR, XOR, NOT, etc.). *Programmable logic devices* (PLDs) are chips which can be configured ("programmed") after manufacturing to perform complex logic functions. Modern practice is to use PLDs for all but the simplest logic functions.

It is useful to know that it is possible to synthesize any of the logic functions using only NAND gates or only NOR gates.

Exercise 34: Design a logic circuit called a full adder. It should have three one-bit inputs (two addends and a carry input) and should generate a one-bit result plus a carry-out. Individual full adders can be chained together to create multiple-bit addition circuits.

Sequential Logic

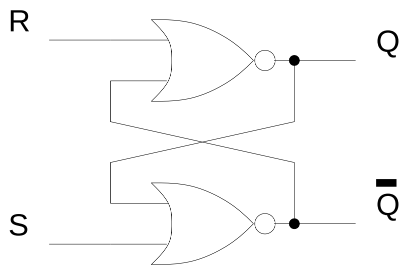
A sequential logic circuit is one where the output depends not only on the current input but also on the past inputs. These circuits are thus said to have memory.

We will start by showing how a sequential logic circuit that “remembers” its past input can be put together. This type of circuit is called a flip-flop. Then we will describe the operation of one of the most common types of flip-flops, the synchronous D (delay) flip-flop.

Sequential logic circuits are state machines. Once the operation of the required circuit is specified as a state machine, the design of the circuit can proceed in a straightforward fashion. We will see how to do this and a design a simple controller as an example.

The RS latch

The schematic of an RS latch is as follows:



The circuit has two inputs R (reset) and S (set) and two outputs Q (the state of the flip-flop) and $\overline{Q}$ (the complement of Q).

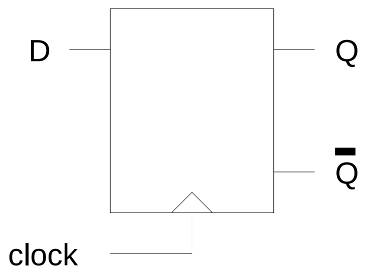
If R=1 and S=0, then we can show that the only possible output values are Q=0 and $\overline{Q}$=1. Similarly if R=0 and S=1 we can show that Q=1 and $\overline{Q}$=0.

When both R=0 and S=0 the outputs are functions only of themselves and retain their values. Thus when both inputs are low the circuit remembers the previously set state.

When both R=1 and S=1 both outputs go low and this violates the condition that Q and $\overline{Q}$ have complementary values so this set of inputs is not allowed. A circuit that uses this RS flip-flop should therefore be designed so that these conditions don’t happen (such as by using additional logic at the inputs).

The D flip-flop

The D (delay) flip-flop has a two inputs, the next-state input (D) and a clock input (usually labelled with a triangle on the schematic symbol).



The D flip-flop has the property that the state only changes when the clock signal changes state from low to high. In the truth table for the flip-flop this is shown by an arrow.

D	clock	Q	$\overline{Q}$
1	↑	1	0
0	↑	0	1
X	0	Q(t-)	$\overline{Q}$(t-)

Usually many (or all) of the flip-flops in a circuit will have the same signal applied to their clock inputs. This *synchronous* operation guarantees that their states will change at the same time.

D flip-flops often have additional inputs that can preset the state to zero or one.

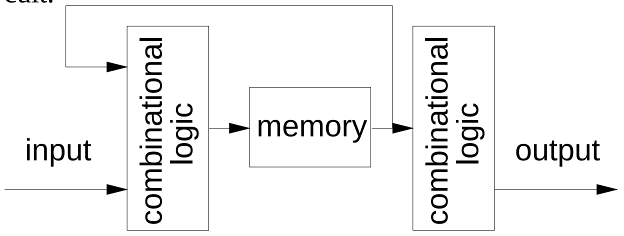
Design of Sequential Logic Circuits

The first step in the design of a sequential logic circuit is to specify the inputs, outputs, states, and transition conditions just as in the design of any other state machine.

Next we choose a sufficient number of flip-flops to represent all of the possible states. *n* flip-flops can be used to represent up to 2^n states (e.g. 3 flip-flops can encode up to 8 states).

We then build a table similar to the tabular form of the state machine representation that has one line for for each possible combination of inputs and flip-flop states. On each line we also show the flip-flop inputs required to move to the desired next state. We also write out a table showing the required outputs for each state.

The last step is to design two combinational logic circuits to implement the two truth tables. The inputs to the first circuit are the outputs of the flip-flops (representing the current state) and any inputs to the sequential circuit. This circuit's output is fed back to the inputs of the flip-flops (representing the next state). The second circuit's input are the current state and its outputs are the outputs of the sequential circuit.



We also need to apply a clock signal to the clock inputs of the flip-flops. The sequential circuit will change state (although perhaps to the same state) at every positive edge of this clock signal.

Example

We need to design a controller for a simple drilling machine. The machine is designed to drill holes in wooden boards that are passing by the machine on a conveyor belt.

The controller has two inputs: a sensor that tells us that a board is in position (board=1) and a sensor on the drill that tells us that the hole has been drilled through (done=1). There are two outputs: a signal to drive an electrically-driven hook that grabs the next board coming along the belt, preventing it from continuing and holding it in place for drilling (hold=1) and a signal that turns on the drill (drill=1).

The controller must actuate the board "capture and hold" mechanism and then wait until a board is in position (state = WAITING). Then it must run the drill until the hole is drilled (state = DRILLING). Then it must release the board and wait until the board leaves the drilling station (state = RELEASING) and go back to wait for the next board to come along. We will ignore error conditions in this example.

From the design specification we can identify three states and draw the state diagram.

Exercise 35: Draw the state transition diagram.

We can arbitrarily assign the controller states to three of the four possible combinations of flip-flop states.

Then we build the tabular form of the state diagram and include the following columns:

- the current state name
- the flip-flop values for the current state
- one line for each possible set of input conditions for each state
- the output values for each of these lines
- the next state name for each of these lines
- the flip-flop values for these states

Exercise 36: Write out the state transition table showing the binary encoding of the states.

The final step is to develop the combinational circuits for each of the flip-flop inputs and each of the controller outputs. These will be functions of the current flip-flop outputs (the current state) and the inputs. These functions can be written in sum of products form by inspection of the table.

Exercise 37: Obtain the boolean equations for the state flip-flop inputs.

These functions can then be simplified if possible and the schematic diagram drawn from these expressions.