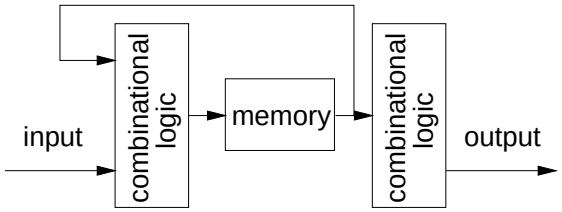
Sequential Logic Design

This lecture reviews the design of sequential logic.
After this lecture you should be able to design a state machine from an informal description of its operation.

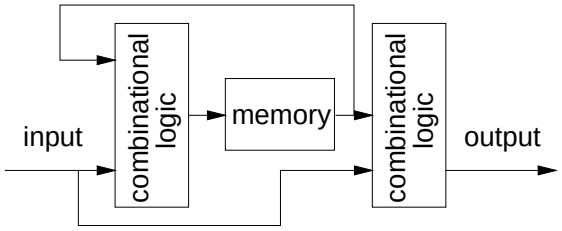
Sequential Logic and State Machines

Sequential logic circuits are circuits whose outputs are a function of their *state* as well as their current inputs. The state of a sequential circuit is defined as the contents of all of the memory devices in the circuit. Thus all sequential logic circuits have memory.

In theory, *any* sequential logic circuit, even the most complex CPU, can be described as a single state machine (also called a "finite" state machine or FSM). There are two basic types of state machines. In the *Moore* state machine the output is a function only of the current state:



whereas in the *Mealy* state machine the output is a function of the current state and the current inputs:



Moore state machines are simpler and are usually preferred because it's easier to ensure that they will behave correctly for all inputs. However, since their outputs only change on the clock edge they cannot respond as quickly to changes in the input.

Exercise: Which signal in the above diagrams indicates the current state?

Large sequential circuits such as microprocessors have too much state to be described as a single state machine.

Exercise: How may possible states are there for a CPU containing 10000 flip-flops?

A common approach is to split up the design of complex logic circuits into storage registers and relatively simple state machines. These state machines then control transfers between the registers. This type of design is called as *Register Transfer Level* (RTL[1]) design. In this lecture we will study the design of simple FSMs. In later lectures we will combine these simple state machines with registers to build relatively complex devices.

Common Sequential Logic Circuits

The *flip-flop* is the basic building block for designing sequential logic circuits. It's purpose is to store one bit of state. There are many types of flip-flops but the only one we will use is the D (delay) flip-flop. The rising edge of a clock input causes the flip-flop to store the value of the input (typically called "D" and makes it available on the output (typically "Q"). Thus the D flip-flop has a next-state input (D), a state output (Q) and a clock input. The D flip-flop state changes only on the clock edge.

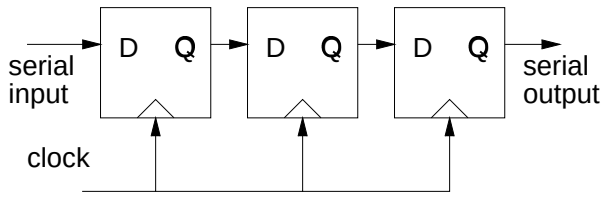
Usually all of the flip-flops in a circuit will have the same signal applied to their clock inputs. This *synchronous* operation guarantees that all flip-flops will change their states at the same time and makes it easy to estimate how fast a clock we can use and still have the circuit will operate properly. *Avoid using different signals for flip-flop clocks whenever possible!*

A *register* is several D flip-flops with their clocks tied together so that all the flip-flops are loaded simultaneously. A *latch* is a register that whose output follows the input (is *transparent*) when the clock is low.

[1] RTL can also mean Register Transfer Language and Register Transfer Logic

Exercise: What would be another name for a 1-bit register?

A *shift register* is a circuit of several flip-flops where the output of each flip-flop is connected to the input of the adjacent flip-flop:



On each clock pulse the state of each flip-flop is transfered to the next flip-flop. This allows the data shifted in at one "end" of the register to appear at the other end after a delay equal to the number of stages in the shift register. The flip-flops of a shift registers can often be accessed directly and this type of shift register can be used for converting between serial and parallel bit streams.

Exercise: Add the parallel outputs on the above diagram.

A *counter* is a circuit with an N-bit output whose value increases by 1 with each clock. A *synchronous counter* is a conventional state machine and uses combinational circuit (an adder) to select the next count based on the current count value. A *ripple counter* is a simpler circuit in which the the Q output of one flip-flop drives the clock input of the next counter stage.

Exercise: Draw block diagrams of two-bit synchronous and ripple counters showing the clock inputs to each flip-flop. Is a ripple counter also a synchronous state machine?

Design of State Machines

The first step in the design of a state machine is to specify the the inputs, the states, the outputs, and the conditions required to change states. It's important to ensure these items are identified correctly. If not, the remainder of the design effort will be wasted.

We then choose enough memory elements (typically flip-flops) to represent all 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). In some cases we can simplify the design of the state machine by using more than the minimum number of flip-flops.

Exercise: If we used 8-bits of state information, how many states could be represented? What if we used 8 bits of state but added the condition that exactly one bit had to be set at any given time (a so-called "one-hot encoding")?

Although it's possible to arbitrarily encode states into combinations of flip-flop values, sometimes a particular encoding of states can simplify the design. For example, in the case of a Moore state machine we may be able to eliminate the combinational circuit that generates the output by choosing an appropriate encoding and possibly using more than the minimum number of flip-flops.

As with combinational logic, the simplest description of a sequential circuit is as a table with one line for each possible combination of state and inputs. After the inputs and the state encodings have been determined, the next step is to exhaustively enumerate all the possible combinations of state and input. Then, based on the design's requirements, we determine the required output and next state for each line. In the case of a Moore state machine there is only one possible output for each state.

The final step is to design the two blocks of combinational logic that determine the next state and the output. The design of these combinational circuits proceeds as described previously.

We also need to apply a clock signal to the clock inputs of the flip-flops. The sequential circuit will change state on every rising edge of this clock signal. Practical circuits will also require some means to initialize (reset) the circuit when power is first applied.

Example: Synchronous 2-bit Counter

A two-bit counter will have four states. Two flip-flops are sufficient to implement four states. In this case there are no inputs, the circuit merely counts up

at each clock signal. The transition conditions are simply to unconditionally go from one state (count) to the next state (next higher count).

If we use the variables Q0 and Q1 to represent the state of the system, and Q0' and Q1' as the subsequent state, the tabular representation would be as follows:

Q1	Q0	Q1'	Q0'
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

This example is particularly simple since there are no inputs and the outputs are the same as the values of the state variables. The combinational circuit only needs to determine the next state based on the current state. We can obtain the following sum-of-products expressions for these equations:

$$Q1' = \overline{Q1}Q0 + Q1\overline{Q0}$$

$$Q0' = \overline{Q1}\,\overline{Q0} + Q1\overline{Q0}$$

Exercise: Write the tabular description of a counter with an up/$\overline{\text{down}}$ input that controls the count direction. Add an *enable* input that prevents the count from incrementing unless it is asserted.