In the last lecture we learned how to build combinational logic circuits given a functional specification that told us how output values were related to the current values of the inputs.

But here's a simple device we can't build with combinational logic.

The device has a light that serves as the output and push button that serves as the input.

If the light is off and we push the button, the light turns on.

If the light is on and we push the button, the light turns off.

What makes this circuit different from the combinational circuits we've discussed so far?

The biggest difference is that the device's output is not function of the device's *current* input value.

The behavior when the button is pushed depends on what has happened in the past: odd numbered pushes turn the light on, even numbered pushes turn the light off.

The device is "remembering" whether the last push was an odd push or an even push so it will behave according to the specification when the next button push comes along.

Devices that remember something about the history of their inputs are said to have state.

The second difference is more subtle.

The push of the button marks an event in time: we speak of the state before the push ("the light is on") and state after the push ("the light is off").

It's the transition of the button from un-pushed to pushed that we're interested in, not the whether the button is currently pushed or not.

The device's internal state is what allows it to produce different outputs even though it receives the same input.

A combinational device can't exhibit this behavior since its outputs depends only on the current values of the input.

Let's see how we'll incorporate the notion of device state into our circuitry.

We'll introduce a new abstraction of a memory component that will store the current state of the digital system we want to build.

The memory component stores one or more bits that encode the current state of the system.

These bits are available as digital values on the memory component's outputs, shown here as the wire marked "Current State".

The current state, along with the current input values, are the inputs to a block of combinational logic that produces two sets of outputs.

One set of outputs is the next state of the device, encoded using the same number of bits as the current state.

The other set of outputs are the signals that serve as the outputs of the digital system.

The functional specification for the combinational logic (perhaps a truth table, or maybe a set of Boolean equations) specifies how the next state and system outputs are related to the current state and current inputs.

The memory component has two inputs: a LOAD control signal that indicates when to replace the current state with the next state, and a data input that specifies what the next state should be.

Our plan is to periodically trigger the LOAD control, which will produce a sequence of values for the current state.

Each state in the sequence is determined from the previous state and the inputs at the time the LOAD was triggered.

Circuits that include both combinational logic and memory components are called sequential logic.

The memory component has a specific capacity measured in bits.

If the memory component stores K bits, that puts an upper bound of 2^K on the number of possible states since the state of the device is encoded using the K bits of memory.

So, we'll need to figure out how to build a memory component that can loaded with new values now and then.

That's the subject of this chapter.

We'll also need a systematic way of designing sequential logic to achieve the desired sequence of actions.

That's the subject of the next chapter.

We've been representing bits as voltages, so we might consider using a capacitor to store a particular voltage.

The capacitor is passive two-terminal device.

The terminals are connected to parallel conducting plates separated by insulator.

Adding charge Q to one plate of the capacitor generates a voltage difference V between the two plate terminals.

Q and V are related by the capacitance C of the capacitor: Q = CV.

When we add charge to a capacitor by hooking a plate terminal to higher voltage, that's called "charging the capacitor".

And when we take away charge by connecting the plate terminal to a lower voltage, that's called "discharging the capacitor".

So here's how a capacitor-based memory device might work.

One terminal of the capacitor is hooked to some stable reference voltage.

We'll use an NFET switch to connect the other plate of the capacitor to a wire called the bit line.

The gate of the NFET switch is connected to a wire called the word line.

To write a bit of information into our memory device, drive the bit line to the desired voltage (i.e., a digital 0 or a digital 1).

Then set the word line HIGH, turning on the NFET switch.

The capacitor will then charge or discharge until it has the same voltage as the bit line.

At this point, set the word line LOW, turning off the NFET switch and isolating the capacitor's charge on the internal plate.

In a perfect world, the charge would remain on the capacitor's plate indefinitely.

At some later time, to access the stored information, we first charge the bit line to some intermediate voltage.

Then set the word line HIGH, turning on the NFET switch, which connects the charge on the bit line to the charge on the capacitor.

The charge sharing between the bit line and capacitor will have some small effect on the charge on the bit line and hence its voltage.

If the capacitor was storing a digital 1 and hence was at a higher voltage, charge will flow from the capacitor into the bit line, raising the voltage of the bit line.

If the capacitor was storing a digital 0 and was at lower voltage, charge will flow from the bit line into the capacitor, lowering the voltage of the bit line.

The change in the bit line's voltage depends on the ratio of the bit line capacitance to C, the storage capacitor's capacitance, but is usually quite small.

A very sensitive amplifier, called a sense amp, is used to detect that small change and produce a legal digital voltage as the value read from the memory cell.

Whew!

Reading and writing require a whole sequence of operations, along with carefully designed analog electronics.

The good news is that the individual storage capacitors are quite small.

In modern integrated circuits we can fit billions of bits of storage on relatively inexpensive chips called dynamic random-access memories, or DRAMs for short.

DRAMs have a very low cost per bit of storage.

The bad news is that the complex sequence of operations required for reading and writing takes a while, so access times are relatively slow.

And we have to worry about carefully maintaining the charge on the storage capacitor in the face of external electrical noise.

The really bad news is that the NFET switch isn't perfect and there's a tiny amount leakage current across the switch even when it's officially off.

Over time that leakage current can have a noticeable impact on the stored charge, so we have to periodically refresh the memory by reading and re-writing the stored value before the leakage has corrupted the stored information.

In current technologies, this has to be done every 10ms or so.

Hmm.

Maybe we can get around the drawbacks of capacitive storage by designing a circuit that uses feedback to provide a continual refresh of the stored information… Here's a circuit using combinational inverters hooked in a positive feedback loop.

If we set the input of one of the inverters to a digital 0, it will produce a digital 1 on its output.

The second inverter will then a produce a digital 0 on its output, which is connected back around to the original input.

This is a stable system and these digital values will be maintained, even in the presence of noise, as long as this circuitry is connected to power and ground.

And, of course, it's also stable if we flip the digital values on the two wires.

The result is a system that has two stable configurations, called a bi-stable storage element.

Here's the voltage transfer characteristic showing how V_OUT and V_IN of the two-inverter system are related.

The effect of connecting the system's output to its input is shown by the added constraint that V_IN equal V_OUT.

We can then graphically solve for values of V_IN and V_OUT that satisfy both constraints.

There are three possible solutions where the two curves intersect.

The two points of intersection at either end of the VTC are stable in the sense that small changes in V_IN (due, say, to electrical noise), have no effect on V_OUT.

So the system will return to its stable state despite small perturbations.

The middle point of intersection is what we call metastable.

In theory the system could balance at this particular V_IN/V_OUT voltage forever, but the smallest perturbation will cause the voltages to quickly transition to one of the stable solutions.

Since we're planing to use this bi-stable storage element as our memory component, we'll need to figure out how to avoid getting the system into this metastable state.

More on this in the next chapter.

Now let's figure out how to load new values into our bi-stable storage element.