So today we are going to talk about another process of lumping Do you see where the problem This is my forbidden region or another process of discretization what will lead to the digital abstraction. So today's lecture is titled "Go Digital". So let me begin with a usual review. And so in the first lecture we started out by looking at elements and lumping them.

For example, we took an element and said for the purpose of analyzing electrical properties let's lump this element into a single lumped value called a resistor, R. And this led to the lumped circuit abstraction. The lumped circuit abstraction says let's take these elements, connect them with wires and analyze the properties of these using a sort of analysis technique.

So a set of a methods. We've looked at the KVL, KCL method. Another example of a method we looked at was the node method. And of this category there is one method you should remember, which you can apply to every single circuit and it will simply work, is the node method.

For linear circuits other methods also apply, and these include superposition, Thevenin method, and in recitation or in your course notes you would have looked at the Norton method.

So that's what we did so far. So this is a toolkit.

So now you have a utility belt with a bunch of tools in it, and you can draw from those tools.

And, just like any good carpenter, you know, the carpenter has to cut a piece of wood.

He could use a chisel. He could use a saw.

He could use an electric saw. And the reason you pay carpenters $80 an hour in the Boston region is because they know which tool to use for what job.

So what we'll learn today is, so this was one process of discretization. We discretized matter.

This gave us the discipline here that we decided to follow, lumped matter discipline, that moved us from Maxwell's equations into this new playground called EECS.

Where all elements looked like these rinky-dinky little values like resistors and voltage sources and so on.

What we'll do today, if that wasn't simple enough, let's simplify our lives even further.

What we're going to do is lump some more.

So what else can we lump? We've lumped matter, so all matter is taken care of. So what can we lump to make life even easier? When in doubt, if things are complicated, discretize it or lump it, right? So what do you think?

What we will do today is lump signal values.

So we'll just deal with lumped values.

And this will lead to the digital abstraction.

And the related reading is Chapter 5 of the course notes.

So before we do this kind of lumping, let me motivate why we do this. One reason is to simplify our lives, but there is no need to just go around simplifying things just because we can. Let's try to see if there are other reasons motivating the digital abstraction.

So what I would like to start with is a simple example of a analog processing circuit that you should now be able to analyze. So I'm going to be motivating digital. So let's start with an analog circuit that looks like this, two resistors, R1 and R2. And what I'm going to do is apply a voltage source here, V1, apply another one here, V2, and make this connection. And let me call this voltage V nought and call this my output. This voltage with respect to ground node, rather than drawing this wire here, I often times draw a ground here and simply throw ground wherever I want. This symbol simply refers to the fact that the other terminal is taken at the ground node.

So here is my V nought. Now, let's go and analyze this and see what it gives us. In this example, V1 and V2 may be outputs of two sensors, maybe heat sensors or something like that. This is a heat sensor on that side of the room and this is a heat sensor on this side of the room. And I pass their signals through two resistors and I look at the voltage there.

So by now you should be able to write the answer V nought, or the value V nought almost by inspection.

Just to show you, let me use superposition.

When you see multiple sources, the first thing you should think about is can I use superposition to simplify my life? And let me do that.

V nought here is the sum of two voltages, one due to V1 acting alone and one due to V2 acting alone.

So what's the voltage here due to V1 acting alone?

To find out that I short this voltage, I zero out this voltage and look at the effect of V1. So the effect of V1, if this were shorted out, is simply V1 x R2 / R1 + R2.

This is now a voltage divider, right?

A voltage V applied across two resistors and the output taken across one resistor. So that's this value.

Then I could do the second part.

To look at the effect of V2, what I will do is short this voltage and look at the effect of this.

Now, this voltage is across this resistor divider.

And so I get R1 / (R1 + R2) here.

So you'll notice that for something like this, if I had applied KVL and KCL of the node method I would have gotten a bunch of equations, but here I wrote it just by inspection. You should be able to look at circuit patterns like this and write the answers down very quickly. Let's say if I chose R1 to be equal to R2 then V nought would simply be (V1 + V2) / 2.

So if these two values were equal, I simply get the output, the average of the two voltages.

So this guy is an adder circuit.

It adds up these two voltages. But more precisely it's an averaging circuit. It takes two voltages and gives me the average value. Now, if you have two sensors in the room, you might think of why you want to take that average value to control the temperature of the room.

But suffice it to say that V nought is the average of the two values. So let me show you a quick demo of this example and then look at what the problems are with this example. So let's say, as one example, I applied a square wave at V1, which is the top curve, the green curve, and I applied a triangular wave at V2, that's the second one.

As you expect, the output is going to be the sum of the two voltages scaled appropriately.

So notice that I have a square wave with a superimposed triangular wave on top. And I can play around.

What I could do is change the amplitude of my wave form here.

And, as you notice, the amplitude of the output component also changes accordingly.

So this is one simple example of an adder circuit, and the two wave forms get summed up and I get the output.

So I'll switch to Page 3. Let me just draw a little sketch for you here. Here, what I showed you was I had a triangular wave coming on one of these inputs and I had a square wave on the other one, and the output looks something like this.

OK? No surprise here.

This is a simple analog signal processing circuit which gives me the average of two wave forms.

Now, let me do the following. Often times I may need to look at this value some distance away.

So let's say this person here wants to look at the value.

So I bring this wire here. And I also bring the ground connection and I look at it. I look at this value here.

And when I have a long wire I can get noise added onto the circuit. So let's say a bunch of noise gets added into the signal there.

And what I end up seeing here is not something that looks like this but something that looks like that.

That's not unusual. And the problem with this is now when I look at this, if I'm looking to distinguish between, say, a 3.9 and a 3.8, it's really hard to do that because my noise is overwhelming my signal. I have a real problem, a real problem here. Noise is a fact of life.

So what do we do? This is so fundamental.

Large bodies of courses in electrical engineering are devoted to how do I carefully analyze signals in the presence of noise? You'll take courses in speech processing that look at clever techniques to recognize speech in the presence of noise and so on and so forth.

One technique we adopt that we'll talk about here, which is fundamental to EECS, is using the digital abstraction. Let me show you how it can really help with the noise problem.

So the idea is value lumping or value discretization.

Much like we lumped matter, we've discretized matter into discrete chunks, let's discretize value into two chunks. Let's simply say that now I'm going to deal with two values and I can, say, call them high, low.

I have a bunch of choices here. I may call it 5 volts and 0 volts. I may call it true and false.

What I'm doing is I'm just restricting my universe to deal with just two values, zero and one.

This is like dealing with a number system with only two digits. And these are zero and one.

So what I've now done is I'm saying that rather than dealing with all possible continuous values, 0.1, 3.9999

recurring and so on and so forth, what I'm going to do is simply deal with a high and a low. Dealing with this whole continuum of numbers is really complicated.

Let me simplify my life and just postulate that I am going to be looking at high and low. Whenever I see something I'll look at it and say high or low, is it black or white, period. There's no choice here, just two individual values. So that sounds simple, and nice and so on, but what's the point?

What do we get by doing that? Let's take our example.

Let's take what might be a digital system.

Let's take a digital system and let's say I have a sender.

Much like I sent a signal value a long distance, let me have a sender, and I have a ground as well and here is a receiver. This symbol simply says that both of them share a ground wire.

So the sender and a receiver. And what I'm interested in doing, the sender is interested in sending a signal to the receiver. And in the digital system, the way I would send a digital signal is all I can use is ones and zeros, OK? So let's say the sender sends something like this. The sender wants to send a value. This is my time axis and this is 2.5 volts, this is 0 volts and this is 5 volts. My sender has some agreement with the receiver and says I'm just going to be sending to you low values and high values. And this signal here would correspond to "0" "1" "0". It's a symbol.

That's why I have input zero in quotes there.

We'll go into this in much more detail later, but for now suffice it to say that I'm sending a set of signals here "0" "1" "0". This simplistic scheme will not work in many situations but go along with this for a few seconds. So I send the signal sequence "0" "1" "0" out here. And notice that there is a high and a low. And the agreement the sender and the receiver have is that, look, if you see a value that's higher than 2.5 volts that's a high.

If you see a value below 2.5 volts in the wire that's a low.

And I'm going to send a 0 volt and a 5 volt from here.

So now at the sending site let's say I don't have any noise in this system. Let's say this is my Vn, some noise being added. And let's say Vn is 0.

Then in that case I will receive exactly what is sent "0" "0" 5, 2.5, 0 volts. And this is time.

Nothing fancy here, right?

My receiver receives a "0" "1" "0".

Now, the beauty of this is that now suppose I were to impose noise much like I had noise out there and Vn was not 0.

Rather Vn was some noise voltage, let's say 0.2 volts peak to peak. Let's say that simply got superposed on the signal. In which case what do I get?

What I end up here with is a signal that looks like this.

So the receiver gets that signal because a noise is added into my signal and that's what I get.

But guess what? No problem.

The receiver says oh, yeah, this is a 0 because the values are less than 2.5, this is a 1 and this is a 0.

"0" "1" "0". So here my receiver was able to receive the signal and correctly interpret it without any problems. So because I used this value discretization and because I had this agreement with the receiver, I had better noise immunity.

Consequently, I had what is called a noise margin. Noise margin says how much noise can I tolerate? And in this situation, because the sender sends 5 volts and 0 volts, the 5 volts can creep all the way down to 2.5, I'll still be OK. Similarly, 0 could go all the way up to 2.5, I'd still be OK.

So in this case I have a noise margin of 2.5 volts for a 1 and similarly 2.5 volts for a 0, because there are 2.5 volts between a 0 volt and 2.5. So notice that I have a nice little noise margin here, which simply is the English meaning of the term there is a margin for noise.

And even though I can change the signal value by up to 2.5 volts, the receiver will still correctly interpret the signal.

So I've decided to discretize values into highs and lows.

And because of that, if all I wanted to do in life is send highs and lows I can send them very effectively.

There are many complications, but if all I care about is sending highs and lows I can send it with a lot of tolerance to noise. So many of you are saying but what about this, but what about that?

There are lots of buts here. And let's take a look at some of them. If you look up there.

What I ended up doing was creating a design space that looked like this. This is on Page 6.

What I did was I said with a range of values from 0 to 5, what I'm going to do is at 2.5 I drew a line and I said as a sender if you wanted to send a 0 then you would send a value here. And if you wanted to send a 1 you would send a value here. Similarly, for a receiver.

And if the sender sent a value all the way up in 5 volts that was the best thing, but technically the sender could send any value between 2.5 and 5.

And if there was no noise then the receiver could correctly interpret a 1 if it was above this and 0 if it was below this.

The problem with this approach really is that if I allow the sender to send any value above 2.5 all the way to 5 then there really is no noise margin in this situation.

OK? Because if I allowed the sender to send any value between 2.5 and 5 then what if I have a value 2.5 for a 1? Then I may end up getting very little noise margin on the other side.

Worse yet, what if I get a value 2.5?

That's a much worse situation. What if the receiver receives a value of 2.5? Now what?

What does the receiver do? The receiver cannot tell whether it's a 1 or a 0. The receiver gets hopelessly confused. So to deal with that, I'm going to fix this, what I'm going to do is the following. Switch to Page 7.

What I'll do here is to prevent the receiver from getting confused, if the receiver saw 2.5, what I'm going to do is define what is called "no man's land".

I'm going to define the region of my voltage space called the forbidden region. And what I'm going to do is, say, let's say I defined it as 2 volts, 3 volts and 5 volts, 0, 2, 3 and 5. With my forbidden region, if I have a sender then I tell the sender you can send any value between 3 and 5 for a 1. And you can send any value between 2 and 0 for a 0. To send the symbol 0, I can send any voltage between 0 and 2, and similarly for 1.

At the receiving side, if I see any value between 3 and 5, I read that as a 0, and any value between 0 and 2 I read that as 2 volts. So I may label this value VH and label this threshold VL, so there's a high threshold and a low threshold. So this solves one problem.

Now the receiver can never see a value in the forbidden region.

Now, I can stand her and pontificate and say, oops, that's a forbidden region, thou shalt not go there.

But what if I get some noise and a value goes in there?

In real systems values may enter there.

But what I'm saying, so this is the beauty of using a discipline. Let me use my playground analogy. This is my playground.

We got into this playground using the discrete matter of discipline, the playground of EECS, but in that playground some region of that playground deals with just high and low values. I further restrict the playground and I say I'm only going to focus on that playground in which all signal values have a forbidden region.

All senders and receivers adhere to a forbidden region.

And if there is any signal in this space, in the forbidden space then my behavior is undefined.

I don't care. You want to go there?

Sure. I don't know what's going to happen to you. Now, we're engineers, right? So we've disciplined ourselves to play in this playground. It's like I tell my 9-year-old, don't go there, right?

And of course he wants to go there.

He says what will happen if I go there?

And the answer here will be undefined, OK?

Something really bad could happen to you.

I don't know what it is but something really bad, you know, a lightening bolt or who knows what, but something really bad. And you as a designer of a circuit can, let's say you were Intel.

Intel designs its chips. And let's say Intel decides to play in this playground and there is a forbidden region.

So Intel says oh, it's really easy for me if in the forbidden region the chip simply burns up and catches fire, we'll sell more chips. That's fine.

Whatever you want. The key here is that all I'm saying is that I am going to discipline myself into playing in this playground and that's where I will define my rules, and you stay within the boundaries and all the rules will apply. It's called a "discipline." You're disciplining yourselves to stay within it.

There's no logic to it. It's just a discipline.

Just do it and you'll be OK. When we look at practical circuits and so on, we have to address the issue of what happens when things go in there.

But let's postpone that discussion.

For now I've solved one of my problems, which is, the previous problem was what does a receiver do if it saw a 2.5? Now it can't see a 2.5.

But then the receiver asks, Agarwal, but what if I see a 2.5? I can tell the receiver you can do whatever you want to do. You can stomp it.

You can squish it. You can burn it.

You can chuck it. Whatever you want.

It's up to you. Do whatever you want.

You won't see a value. If you do, do whatever you want. It's undefined.

That works. So you, as the receiver designer can do whatever you want when you see a 2.5.

You can say yeah, I'll just put out a 1 if I see a 2.5 or a 2.6. I'll just do something.

No one cares. So this is pretty good.

This is pretty good. We still have a problem, though. Do people see the problem here?

This still doesn't quite work. If Intel did this, instead of your laptops failing and blue-screening every hour they'd be doing it every millisecond.

So the problem is this discipline have allowed the sender to send any value between 3 and 5 as a 1.

And any value between 3 and 5 at the receiver is treated as a is? Yes?

The sender sends a 1.99 and the noise pumps it into forbidden region. Exactly.

So the sender says it's legitimate, I'm Intel.

They've told me stick to 0 and And Intel parts will be sending to values between 0 and 2.

And Motorola parts, which are receivers, you know they have to receive 0 and 2.

So Intel can send the value, They can because it's 1.9 out of 2.

It's legal. This way I can make really cheap parts. But now the problem is that even the smallest amount of noise will bump it into the forbidden region, and so therefore this one has a problem. And the problem is that this one offers zero noise margin. There is no noise margin.

There is no margin for noise in the discipline.

All right, back to the drawing board, folks.

Switch to Page 8. Let's get rid of all this stuff and go back to the drawing board.

OK, so what do we do now? How about the following?

How, about as before I say, as a receiver, if you see a value between 3 and 5 you treat that as a 1 and a value between 0 and 2 you treat that as a 0.

No difference. So as a receiver same as before. But now what I do is I hold the sender to tougher standards. I hold the feet of the sender to the fire and say you have to adhere to tougher standards.

So what I'm going to do is hold the sender to tougher standards, maybe four walls. That is tell the sender that if you want to send to 0 or a 1, for a 1 you have to send a value between 4 and 5, and for a 0 a value between 0 and 1. Sender is now held to tougher standards. This is what my chart looks like. So now I do have some noise margin. Can someone tell me what is the noise margin here for a 1? 1 volt.

And the reason is that the lowest voltage a sender can send is 4 volts, OK? If the 4 leaks down to 2.99 that's in the forbidden region, I'm in trouble.

here. And 2.99 is in the forbidden region. I'm in trouble.

So notice that the lowest value that the receiver can receive is 3 volts. So if I sent the 4 and sent this over a long cable to you, the value can be beaten up by noise to such an extent that you may begin receiving 3s but nothing lower than a 3. So this is a noise margin, 1 volt. Similarly, for a 0 the noise margin is also 1 volt. So let me label these.

There are four important thresholds here.

This threshold is called VOL. V output low.

These have special meanings. This threshold here is called VOH, V output high. This threshold here is called V input high and this threshold here is called V input low.

So VOH simply says that senders must send voltages higher than VOH. Receivers must receive values higher than VIH as a 1. So these four thresholds together give you your threshold.

For the sender gets 2.5, what does sender do?

It could do that. So, in that case, you can do that. If all you want to do is have one value here then what you have is an infinitesimal value here for the forbidden region. That's fine.

It's up to you to design it that way.

You can. But it turns out that when you design circuits, when we see some examples in the next lecture it turns out to be fairly practical and easy to do it this way. But, again, these are design choices. If I'm Intel, Intel wants all its parts to work together.

So parts that follow a common discipline can work together, right? Because senders will send values, receivers will receive these values here, so it will simply work. So the noise margin for a 1 here is simply VOH minus VIH and the noise margin for a 0 is VIL minus VOL. VIL minus VOL is the noise margin for a 0. So what do we have here?

What we have here is a discipline that we've agreed to follow where senders are held to a tough standard and receivers are held to a different standard so that I allow myself some margin for error. And it's up to you as a designer to choose ranges for the forbidden region.

Now, you may say that I want to make my forbidden region as small as possible. But you will see in practical circuits it's very hard to achieve that.

Practical devices that you get, they have a natural region that gets very, very hard to break apart, and that tends to establish what that region looks like.

So to continue with an example here, I may have the following voltage wave form for a sender. So I have some sender, I have a sender here.

I have VOL, VIL, VIH, VOH and some other high voltage. And then, as a sender, if I want to send a "0" "1" "0" then I send a 0.

I have to be within this band. And then for a 1 I have to be within this band. So this is an example of, say, "0" "1" "0" "1". And at the receiver -- Let's have VOL, VIL, VIH, VOH.

So at the receiver, I interpret any signal below VIL as a 0. So I may get some signal that looks like this.

And I'll still interpret that as a "0" "1" "0" "1".

So to summarize here, this discipline that forms the foundations of digital systems is called "a static discipline".

The static discipline says if inputs meet input thresholds -- So if an input to a digital system meets the input thresholds then outputs will meet, or the digital system should ensure that the outputs -- Output thresholds. So this means that if I have a system like this then if I give it good inputs.

And by giving it good inputs I mean for 1s I have signal values that are greater than VIH and for 0s signal values which are less than VIL. These are valid inputs.

So if my inputs are valid, that is below VIL for a 0 and above VIH for a 1 then this digital system D will produce corresponding outputs that follow output thresholds.

For a 1 it will produce outputs that are greater than VOH and if it needs to produce a 0 it will produce outputs that are less than VOL. So notice that there is this tough requirement in digital systems that for the inputs, I should recognize as a 1 anything higher than a VIH.

But if I want to produce a 1, I have to produce a tough 1 like a 4-volt 1. So there is a discipline that all my digital systems must follow, and that discipline is called a static discipline. So static discipline encodes the thresholds, encodes four thresholds that all digital systems must follow so that they can talk to each other. So if Intel and Motorola want to make parts that are compatible with, say, Pentium 4 devices then they will all talk over the phone or something and agree on a static discipline.

We will say that, all right, all my peripherals will follow a static discipline with the following volted thresholds. And this way parts made by different manufacturers can interoperate and still provide immunity to noise. Yes.

Question?

Absolutely. There are many constraints on how you as a designer choose the noise margin.

As a designer you want to make your noise margin as large as possible. The larger the noise margin the better you can tolerate noise which is why, how many people have heard of some devices called rad hard devices, radiation hard devices? Some of you have.

There are a bunch of devices. Different manufacturers make different kinds of devices for different markets.

For consumer markets they use parts which may have relatively poor noise margins because consumers can tolerate more faults. But if you're building devices for, say, the medical industry or for spaceships and so on, you need to be held to a much, much tougher standard.

So for those devices you may end up having much, much tighter bands in which you have to operate so you have a tougher noise margin. So that leads us to, given these sort of voltage thresholds, we now move into the digital world. And in the digital world we can build a bunch of digital devices.

The first device we will look at is called a combinational gate.

A combinational gate is a device that adheres to the static discipline, Page 11, and this is a device whose outputs are a function of inputs alone.

So I can build little boxes which take some inputs, produces an output where the outputs are a function of the existing inputs. And this kind of a device is called a combinational gate. And I can analyze such devices for the kinds of things that I would like to do.

Before I go into the kinds of devices I'd like to build, let's spend a few minutes talking about how to process signals. How to process digital signals, Page 10. So notice that you have two values, 0 and a 1. So devices like my combinational gate, for example, can only deal with 0s and 1s. So I have to come up with some kind of a mathematics or some kind of a set of processing that can work with 0,1 values. So 0,1 map completely natural to the logic true and false. So I can borrow from logic and use true and false to do my processing of signals.

So if all I care about is processing logic values, 0s and 1s, trues and falses then that's all I need.

I can also use numbers. How do I represent a number?

3.9 which is 0s and 1s. It turns out that this is a whole field in itself. You'll hear more about this in recitation. Let me also point you to the last section of the course notes, Chapter 5.6 I believe, that talks about how to represent numbers.

The basic insight is much like you can represent arbitrary long numbers with the digits 0 through 9 in the same way, but concatenating digits you can represent arbitrary long numbers with 0-1-1-1-0-0 and so on.

So you can have a whole sequence of digits and you can build a binary number system. So you can read A&L Section 5.6, I believe. It's the last section for numbers. And you will also discuss this in your recitation tomorrow. Let me spend some more time talking about Boolean logic, two-valued logic, and how to process these systems.

So one way of processing it is using logic statements of the following form. If X is true and Y is true then Z is true,

else is Z false. So this is a logic statement.

It says if X is true and Y is true then Z is true, else Z is false. So I can process this with 0s and 1s, trues and falses. And I do this all the time so I have a succinct notation for this.

I express this as Z is X anded with Y.

X and Y is Z. So Z is true if X is true and Y is true. A shorthand notation for this is just a dot. And a circuit notation for this is called an "AND gate". That's a little circuit.

I haven't told you what's inside it.

It's an abstract little device called an AND gate which takes two inputs, produces one output Z where the output is related to the inputs in the following manner.

That's a little device called an AND gate.

I could also represent logic in truth tables.

And truth tables simply enumerate all the values and the corresponding outputs. Inputs can be 0-0-0-1-1-0 or 1-1. For an AND system output is 1, only if both are ones, it's a 0 otherwise.

So that's a truth table for AND gate.

So from 0s and 1s we deal with logic and we create devices like the AND gate to process digital signals.

And what we will do is look at a whole bunch of little symbols like this, like the AND gate to process our input signals.

And these devices might look like other functions like OR gates and so on. Let me show you a quick demo.

What I'm going to show you is a signal feeding an AND gate.

And one signal is going to look like this, and my signal Y is going to look like this. So you expect a processed output. So 1-0-1-0-1-0-1.

And the output is simply going to be -- This is my time axis going this way.

It is going to be an AND-ing of these two signal values like so.

What I'm also going to show you is I'm going to superimpose noise on this wire. I'm going to superimpose noise

on the wire, and what I want you to observe is the output of this digital gate. The output will stay exactly like this, even though I impose noise.

The ultimate test. So stay right there.

Let's do this demo. Give me a couple of seconds.

If you look at the signal up there, look at the middle wave form, and I'm imposing let's have a digital system in a noisy environment like a lumberyard, for example, or chopping a bunch of trees in my backyard and building digital systems on the side. And if I have my buddies revving up chainsaws superimposing noise on my second input, but look at the output. And just to show that I'm not bluffing here, what I'll do is I'll pass the noise through and make the noise larger.

And you'll notice that when the noise begins to surpass the noise margins the output begins to go berserk.

Watch. Can you increase it gradually?

Notice that as I put in a lot more noise then the output begins to go berserk, but as long as my input is within the noise margin my output stays perfectly stable.

So that's the "Intro to Digital Systems".

You'll see numbers in recitation.

And we'll see you at lecture on Tuesday.