PROFESSOR: The CDMA and the cellular standard, the IS95, which is a kind of an old standard by now, but it's still widely used. There's still three major standards that are used for cellular communication, cellular voice communication. This is one of them. This is the one which is primarily being used as a jumping off point for all of the new standards that people are trying to think of. I want to talk about it for that reason but also because it really is a nice way of pulling together almost everything we talked about in the course. All of it comes into this one particular way of sending cellular voice.

Let's go through some of the details of it. Some things here might sound more detailed than what you're interested in but I think it's good to have the numbers on it because it gives you some idea of which things, which factors are really relevant and which ones aren't so relevant. So it uses a frequency ban from 800 to 900 megahertz. Most of the new standards that people are thinking up are up in the gigahertz range around two gigahertz or five or six gigahertz. I don't know the exact bandwidths. All of these bandwidths depend critically on what the FCC wants to allocate and also depend critically one which particular bands are good for electromagnetic propagation and which ones tend to absorb all of the radiation.

Anyway, you use the band 800 to 850 for the reverse channel for the cell phone to the base and you use the band 850 to 900 for the forward channel. Why do you want to separate the reverse channels and the forward channels by as much as you can? If you think for just a minute about the power levels at your sending antenna, either at the base station or at the cell phone itself, and then you think about how much power you're receiving, he see there's an enormous difference between these two. Because things get attenuated enormously after they got transmitted. So what you're trying to do is to filter out what you're sending, which is an enormously high

power and you have to filter it out by enough so that it doesn't clobber what you're trying to receive. This means you need very good filters.

How do you build very good filters? It's a lot easier to build good filters if you're not trying to make them cut off immediately. If you have 50 megahertz, and you actually have a lot more than that, you have exactly 50 megahertz because this is all separated into a bunch of different channels. People in wireless, most practical communicators, when they talk about a channel, what they're talking about is a particular bandwidth. All along what we've been talking about is a particular medium going from transmitter to receiver. That's what they're talking about is these channels, these channels or sub-bands are 1.25 megahertz wide each. We'll see what the significance of that is in a while. So you have a transmit and receive pair which are both 1.25 megahertz wide and which are separated by 50 megahertz. So you have one channel then, at the bottom of the range, another channel a little bit higher, another channel, a little bit higher, and so forth all the way up. So you have lots of different channels and you put lots of different cell phones in each channel for each base station. So that's roughly the overall architecture of it.

It's the same kind of system that we've been talking about all along. You start out with voice wave forms, you have voice compression. You then have channel coding, you then have modulation and then it goes out on the channel. Guess what happens when it comes back? You have demodulation, channel decoding, and then you turn it back into voice. Those are the three major steps which we viewed before primarily as two steps. All of the channel stuff is much harder than all of the source stuff. Let's talk about the source stuff first. The input voice gets segmented into these 20 millisecond segments. Why 20 milliseconds? Well it sort of seems like it's probably a number that somebody picked out of a hat. It's kind of striking that all of the standards use the same interval. If you think about it awhile -- I'll talk about this more on the next slide -- the reason for this interval is that in voice communication, delay is enormously important. You don't want to have much delay and since you don't want to have much delay, you have to do encoding over very short segments.

Well anyway, what this does is it takes each 20 millisecond segment of voice and it

maps it into 172 bits. What that means is if you multiply the 172 by the 50 segments that you have per second, it comes out to be 8.6 kilobits per second. That might not sound like much of an achievement. Think about the old fashioned way that people used to encode voice back in a time when they were first doing digital communication. What you would was you would think of the voice as having a bandwidth that went from about 400 hertz up to about 3,200 hertz. You would then view it as a base band wave form zero to 4,000 hertz. If you were going to try to sample that wave form you'd have to sample it 8,000 times per second. If you want good quality, you then have to use a large number bits. You need a large number of bits for each sample. The standard thing was to use eight bits per sample. That gave you 64,000 bits per second in the old fashioned way of doing it. People have worked for many years. Many engineers at the old Bell labs spent their whole career trying to find out how to compress voice into smaller and smaller numbers of bits.

You know, 8.6 kilobits per second is still sort of a good rate. It's still a good rate if you insist that you have to do it in 20 millisecond segments. If you can take much longer segments to encode voice, you can do much better. Voice really has a lot of constraints over relatively long periods of time. Well actually, you know this because if you take a person's voice and you map it into text and then you also add a few things about pitch and things like that, you can really come by with a very small number of bits. Here you're constraining yourself to something that sounds like the actual voice and also has to be done in these 20 millisecond segments because of delay. So it still is a reasonably good achievement. All of the standards have figures about like this, in the same order of magnitude. I expect that all of the new standards will do similar things. They will probably have slightly smaller numbers of bits. They will probably do much more computation. They will achieve a little bit more because this was already -- I think -- relatively close to how far you can go.

The next thing they do is they add 12 parity checks per segment. They use that for error detection. I don't want to talk about that a lot because it probably isn't necessary. All of the standards do something or other like this. The reason is, if you take this 20 millisecond segment and you go through all of this enormous

processing that we're going to go through: namely turning it into an encoded wave form, transmitting it, receiving it, detecting it, doing all the stuff we're going to do to it. If you get the 20 millisecond segment confused, and usually if you make errors you make a lot of errors. If at the receiver you wind up with the wrong thing, it is really going to sound awful. They found that it's much better if you're not quite sure of what that segment is, to just send silence for 20 milliseconds. The silence is hardly detectable at all; it just sounds like good voice. I mean for most of us it really sound wonderful to have a little more silence than we usually do. Well enough of that.

Then there's another thing. This is eight zeros per segments that are added to this 172 bits as something which is called a terminator for the convolutional code. I might say a little bit about that but not much. So what we've done is to add 12 bits and eight bits to this 172 bits. That brings us up to 192 bits, which brings us up to 9,600 bits per second which is what we're now going to try to transmit. OK so that's the voice compression part of the thing and a little bits of overhead done for various special functions. The main reason I want to talk about these overhead factors is that every communication system I know starts out with very nice principles studying very carefully what are the essential things. By the time you got done with it, there are all sorts of little overhead items that got thrown in to make the thing work and it's frustrating to everybody. It's usually 10% or 20% of the capacity of the thing, but it's always there. That's why I bring up these two terms because that's exactly the kind of things they're doing. They're just patching to make things work. So you have many details that lose all of the efficiency that you would like to have. Those are what makes it hard to compare different systems. Each system, architecturally nice though it might be to start with, always winds up with these little things because of these silly little constraints that say, since you have to have small delay, you have to segment voice into segments which are not much more than 20 milliseconds long. Those things cause you to do these other things. That's where all of this comes in.

OK, all of the timing in this IS95 cell phone system, everything is keyed to this 9,600 bits per second and it's all keyed through this 20 millisecond interval. Everything is done in terms of 20 millisecond intervals. It's all a completely block system. The

source generates 20 milliseconds worth of data. You encode that into a 192 bits. That gets mapped into some code word, gets turned into some high frequency wave form. At the output you look at 20 milliseconds and that gets turned back into your best hope of the 172 bits that you started with and from there to some wave form. Every 20 milliseconds is completely independent of every other 20 milliseconds. You don't even save any knowledge about what the channel is between those periods of time. Everything is independent from one to the next.

The point is here, we talked a great deal about all of digital communication sort of being generated from people's realization that they ought to separate voice coding from channel coding. In other words, there's just binary interface between all sources and all channels. This is a violation of that; all cell phones have this violation. They don't have this pure separation because all of them face the fact that to do this small delay -- anytime you want to get small delay -- you have to have some mixing of what you do with the source and what you do with the channel. The delay has to be some combination of what happens in both places. So, we do have that violation and it's because of interactive voice. You probably never thought about why it is that you need small delay on voice. If you try to talk to somebody with a 50 millisecond delay, in between what you're saying and what the other person is saying, you will find it turns you into a stutter in about five minutes, everyone. The problem is we all have these social conventions that we use to tell us when we can start to talk when we hear silence from the other person. If you have a break in that routine, even at 50 milliseconds, it totally screws it up. Both people start to talk at the same time and it's very hard to have a conversation. Back in the early days of telephony, they found that about as much delay as they could tolerate was about 20 milliseconds. When it got much longer than that, it gets very objectionable. If you talk to somebody on the phone who is in Japan, you have a little more delay than that. You find it's very hard to talk to them. I mean, you have to practice a little bit, you have to think about it. If it's somebody for whom English is a second language, or you have Japanese as a second language, it's very easy. Then you both talk very slowly so you catch up on that delay. but. If you're both talking English very rapidly, or both Japanese very rapidly, it becomes hell on

wheels. As far as the channel is concerned, we could also do very well by using longer code words on the channel. Again, we're stuck by this 20 milliseconds. Everything has to be blocked into 20 millisecond periods.

OK, so the first thing as far as coding is concerned, is what's called a convolutional encoder. We unfortunately, have not talked about coding at all in this course; we're not going to talk about it. I just want to enough about convolutional codes so you get some idea of what they are. Here's the simplest example of a convolutional code that I think you can think of. You have a stream of input bits which are coming in one per second. Each time an input bit comes in, that input bit sits at this dock here. I guess most people would prefer to put an extra memory element in but it just complicates the diagram. You have this input bit, you have the previous input bit, and you have the input bit before that. What comes out, a time J is a linear combination of this bit, that previous bit, and the bit one bit before that. You call this a convolutional code with a constraint length of two because it has memory of these two bits. If you think about it a little bit, this is a device that has four states. OK, because at time J, it needs to remember whether this bit was a one or a zero and whether this bit was a one or a zero. So it's just a linear modulo two device that happens to have four states. It produces two bits in each interval of time which are functions of this bit and of the current state.

In this convolutional encoder which is used in IS95, the constraint length is eight instead of two. In other words, you have 8 of these memory devices here and the rate is one-third instead of one-half Here you have two bits coming out for each bit going in. There you have three bits coming out for each bit going in. It's the same principle. If you think about it a little bit, as you increase the constraint length, the number of possible states that you have is going up very rapidly. So, if you have a constraint length of eight, which means eight binary digits stored there, you have two to the eighth different states, which is 256 states. If you added one more state to this thing, you would double the complexity. Well, it turns out that because of the way the decoder works, if you added one more bit to this block length, it would multiply the complexity of the decoder by a factor of two. So, when you try to decide how long these constraint lengths should be, you have to take into account that

every time you make it one bit longer, yes, the thing is going to work better. Also, you make it twice as complex. Therefore, you have to wait another six months before you produce it at the same cost.

OK, we're going to use a Viterbi algorithm for decoding. If you don't know what the Viterbi algorithm is fine. If you think in terms of finite state devices, think of it this way; you have a device here which has four states. Somehow the decoder has to realize at each instant of time, if it's gonna decode, it could either decode the bits here. A nicer way to think of it is, it decodes the state, at each instant of time. If it decodes the state at each instant of time, it doesn't really have to have any memory. As soon as it knows what the state is at one unit of time, then it has a very simple decision to make to figure out what the incoming bit is and what the state is at the next instant of time. And what a Viterbi decoder is, is something that does maximum likelihood decisions on the state. There's a nice trellis diagram to talk about how that's done, in the notes. You can read about it, it's interesting. Many of you who have probably seen about it, it's sort of the easiest thing to explain, as far as coding theory is concerned. It's the easiest thing to explain until you put all the notation in. Then whoever does it, as soon as they put all of the notation in, it becomes a bloody mess. Anyway, the idea is simple.

So at that point, we can sort of summarize what we've done as far as waste compression and channel coding. So far, the IS95 standard is pretty much the same as any other cellular standard. All of them, it turns out, use convolutional codes. Whether that's a good idea or not is another question, but they all do. It certainly is a very sensible thing to do. All of them use the same segmentation. All of them use similar techniques for doing voice compression. All of them have all these little overhead terms that come in. If you look at the numbers of what's going on, we started out with a 172 bits per second, which was what comes out of the voice compressor. You then have these two overhead items. Incidentally this eight bit convolutional terminator, we can now get some idea of what that's for. You put in your 172 bits here. You're thinking of this thing in terms of the state of each unit of time. If you're going to figure out what's going on at the end, after you put in your

172 bits, you have to put in a couple of extra zeros to let this thing settle down. You have to let this bit go all the way through this device so you got enough data out about it that you can decode it. So that's what that terminator is for, for putting zeros in at the end of the sequence so you can view it all as one block code.

Again, this is overhead, which is caused by the need for small delay. If you're going to need small delay, that overhead term would be much, much smaller. So at that point, we're up to 9.6 kilobits per second and we're up 192 bits per segment. Then we go into this convolutional encoder and the convolutional encoder multiplies things by a factor of three. In fact, the rate one-third is a little bit different for most of the other standards because one of the things that we're going to be doing here, since we're mapping this input into one and a quarter mega bits, that's a considerably broader band than these other standards used. The other standards are quite narrow band and therefore they don't want to have a large number of bits per second. Here we're going to wind up talking about 28.8 kilobits per second at this point which is 576 bits per second, which is three times what we started out with. We've expanded the number of bits by a factor of three. If you question whether this makes sense to get a slightly smaller error probability through coding but increasing your rate by a factor of three, that's a perfectly sensible question to ask. It just turns out that you do get a saving from that and that's what all of coding theory is about, how much savings you actually get. In fact, the savings are pretty considerable. So, at this point, we're up here at 28.8 kilobits per second with 578, 576 bits per segment. We then go through an interleaver. Don't ask yet what the interleaver is for. All the interleaver does is takes these 576 bits and it's scrambles them all around. The receiver knows how they've been scramble so the receiver can unscramble them again. Why you'd need that is something we will talk about later, a good deal later.

The next thing we're going to do, you thought you were through with coding but no. In this system, they're actually two stages of coding that you go through. You can view one of them as modulation instead of coding. Therefore, you can claim you're only doing one stage of coding and one stage of fairly complex modulation. You remember when we talked about orthogonal codewords. We said we could view

orthogonal codewords almost as easily, either as a coding technique or as a modulation technique. Mainly, you'd gather together lots of bits and then you'd form codewords, which we're signals, which had a large number of dimensions in them. We're going to do the same thing here. We are going to take this interleaver output; we're going to segment it into six, six bit blocks. Then each six bit block is going to choose one of a set of 64 orthogonal codewords. I mean, with six bits, you have two to the sixth, which is 64 different combinations. So, if you're going to use orthogonal codewords, you need 64 of them. When we were talking about orthogonal codewords before, we didn't quite know how to generate them.

We then said last time, that may be using PN sequences is a nice way of doing it. That's not quite exact. Here using Hadamard matrices is a nice exact way of doing it. That happens to be the way it's done and it's simple. I mean, you have a Hadamard matrix for each integer b. If you start out with b equals one, which is one bit that you want to encode. You have a matrix which consists of 2 orthogonal codewords one of which is 0, 0 and the other is 0, 1. There aren't too many choices for that and they're all equivalent. Then when you want to go to b equals two, the thing you'd do is make a matrix and the rows of the matrix are going to be the codewords. The columns correspond to the bits in the codeword. The first codeword here is 0, 0 second codeword is 0, 1. The first codeword here is 0, 0, 0, 0 the next one is 0, 1, 0, 1. For orthogonal codes, the block lengths of the code and the number of codewords is going to be the same. So the thing you'd do is you take this matrix, you put it there, you put it there, and you put it there. Then you compliment the matrix and you put it there. OK why does that work? Well if we just look at this part of it, this was an orthogonal code, this was an orthogonal code. In other words, each of them differ from each of the others in half the positions, which is why you think of it as being orthogonal. This is the confusion that runs through all of communication theory. We might as well get rid of that confusion right now. When you talk about binary sequences and you say they're orthogonal, what you mean is, if you turn that sequence into a 2PAM signal, those corresponding signals are orthogonal. In other words, an orthogonal binary sequence, an orthogonal set of binary sequences is a set of binary sequences where each of the sequences differ

from each of the other sequences in half of the positions.

Here with these Hadamard matrices, the way that's done is that one of the sequences is always all zero and the other sequences are all half ones and half zeros. It turns out that when you add any two sequences, add this sequence to this sequence for example, you got something else, which is half ones . In fact, something else which is in there, is what's called a group code. When you add any two codewords you got another code. Anyway, when you look at this, which is mapped into here and here -- I think the argument is difficult here because it's too simple minded, so you don't see it in it's generality -- each time you go from one b to the next b, the thing you do is the same thing. You start out with this matrix, put it there, you put it there, you put it there, and you put the complement of it there. What that does is when you look at the first half of the matrix, all of these differ in half the positions because they differ in half the positions here, and they differ in half the positions here. So, they differ in half the positions over all. When you look at everything down here, they differ in half the positions for the same reason because complementing this doesn't change any of those distance properties. When you look at the difference between these and these, it's the same kind of thing if you think about it for a couple of minutes.

So, this is an orthogonal code. Why do we want to use this orthogonal code rather than just an orthogonal code which uses the first degree of freedom and sends an enormous amount of energy? Or the second degree of freedom sends an enormous amount of energy and so forth? That's the same thing we were talking about last time with regard to channel measurement. You don't want to send enormous amounts of energy in any one degree of freedom. You want to spread it out over all the degrees of freedom. That's what all these codes are doing; all these codes are using 2PAM on each degree of freedom to generate a codeword. So the codewords all have sort of uniform power in them.

OK, so at this point, we have gone into this orthogonal code generator with 28.8 kilobits per second. We have come out of it with 28.8 kilobits per second times 64 over six. Namely for each six bit segment in, we've gotten 64 bits out. So the total

number of bits that we have has gone up by a factor of 64 divided by six. So we're suddenly up to 307.2 kilobits per second. What we're trying to do is get ourselves -- by hook or by crook -- get ourselves up from these very small bit rates that we need to represent voice, up to these large bit rates, 1.25 megabits per second, which we're going to spread this wave form to. All the reasons we want to spread the wave form, I'll talk about them later. For now let's just think about it as saying, somehow or other we want to spread the wave form out and turn a small number of bits into a large number of bits. So we've done a pretty significant part of that right here.

If we think of turning these binary rows of the Hadamard matrix and modulate them by 2PAM we wind up with wave forms. The wave forms are orthogonal. If you wanted to think of taking these wave forms and then transmitting them up to passband, we would wind up with the bandwidth of 307,000 hertz. That's not enough yet, we want to go up to 1.25 megahertz. Anyway, these wave forms are called Walsh functions. So the wave forms that you got from Hadamard matrices are called Walsh functions. You have to find something to honor everybody that's done a significant amount in this field and Walsh was one of these people. Walsh functions are quite famous and all they are are just these orthogonal wave forms. So, we now have an orthogonal code of length 64. The receiver, I'm going to talk about the receiver more later. In essence, what it's going to do is take the 64 orthogonal wave forms, only one of them will get transmitted in each of these small units of time. When the period of time corresponding to six bits into the encoder, we have to decode which of these 64 wave forms was sent. We're going to do that by using a rake receiver. That's what's used in IS95.

This rake receiver is different from the one we talked about. The one we talked about, we were making a decision between two hypotheses. Here we're making a decision between 64 possible hypotheses. If you can remember the structure of that rake receiver, it sort of split into two separate sections. One section was used for the decision on one of the wave forms, the other one, on the other wave form. When we do this with 64 wave forms, it spreads into 64 different sections, which is why it's called the rake receiver. If you draw a picture of it looks like a rake. You start out here with the handle and then you have it spread out into 64 different tines

and all of these tines have all of this signal processing going on. At the output you come back and you change what your model of the channel is depending on the actual decision that you've made.

So what we're using is a rake receiver but a rake receiver which has 64 decision devices instead of two. Suppose we go into an orthogonal code using seven bits instead of six bits. Now, at this point, you're talking about real money. OK, because at this point, you're talking about taking this rake receiver and turning it from 64 arms into 128 arms. Now, you can do that because these things are essentially free now and computation is essentially free. At the time when this was designed, there was a very hard constraint in terms of complexity about how many bits they can encode together into orthogonal wave forms. I'll show you why in a little bit. It wouldn't have done them much better to have a larger set of orthogonal wave forms anyway.

It also uses incoherent soft decisions as a way of doing its detection. We talked about incoherent detection. This rake receiver uses that instead of the coherent detection that we talked about in class. If you just put together in your mind, the lecture we spent talking about rake receivers and the one we spent talking about incoherent detection and you put them together, you have what this receiver is doing. I'll talk about just a little more as we go. OK, if we now imagine these are orthogonal codewords and suppose that each orthogonal codeword has an energy e sub s. Suppose for the time being that the channel is represented perfectly by a single-tap model and we know what the energy in that single-tap model is. We don't know what the phase so we're using incoherent detection. What's the error probability? Well the error probability for an incoherent receiver for making a decision between two possibilities was one-half times e to the minus energy divided by 2 N 0. Here we have 64 different possibilities. So we're going to transmit one of these wave forms and we now use the union band to say there's this probability of error to each of the other 63 possible wave forms. we could make an error to any one of them. That's equally likely to make an error to any one of them because they're each orthogonal to the one that we're dealing with. So this is a reasonably

good bound on the error probability.

OK, now you go back from this to look at the reason why, one of the main reasons why you want to do this. E sub s is the amount of energy you use to transmit six bits instead of one bit. Therefore, when you look at the energy per bit that you're using, it's e sub s divided by six. So, your probability of error, -- if in fact you were making hard decisions at the output of this incoherent detector -- would be in the order of 63 over 2 times e to the minus 3eb over N0. This is a considerably larger exponent and therefore, a considerably lower error probability than we were getting when we were talking about transmitting single bits. In other words, the same thing is going on here as was going on when we were talking about getting up the channel capacity using a large number of orthogonal wave forms. It's the same idea. When you encode lots of bits together, you make a joint decision on all of them, you make a gain. That's exactly what's happening here. You made a gain because you've taken the amount of energy that you have available for each one of your bits and you've combined it all so that you get this bigger exponent in here. If you're lucky, the exponent is big enough that it overwhelms this term and your decision comes out with very small error probability. Well, as it turns out, we're not interested in making these hard decisions. What we're really interested in doing is just sending a likelihood ratio back to this Viterbi decoder which it can use in making final decisions. To see how well that's going to work, this is exactly the thing you want to look at. This is what's telling you, if E b is large enough your decisions, if you had to make them, would be very good. Therefore, if instead of making decisions, you pass likelihood ratios on, those likelihood ratios are going to have a lot of information in them. Essentially, those likelihood ratios tell you what those codewords are.

OK, now, the effect of fading is to change this E sub b because when we were talking about this incoherent detection, we were looking at the energy per bit after going through the channel. So, this is the error probability for a particular received energy per bit. Sometimes when the channel is badly faded, this number is very small. This bound is a lot bigger than one but you don't care about it because the bound is no good then. When the channel is good, this number is large and this

probability there is very small and your likelihood ratios give you a lot of information. If you had multitap diversity instead of the single-tap here, if you have a lot of taps. If you have a lot of taps you can visualize this as getting independent readings on the channel. Therefore, you can view it as getting independent information about the bits that were sent and this is what diversity is. I mean, the trouble with these Rayleigh fading channels is when they're faded there's nothing you can do. If you have multiple taps instead of one tap, then in fact, you have diversity between all of them. If any of them are good, then you decode. Yes?

AUDIENCE: You were saying earlier that PAM is not ideal [INAUDIBLE]. [INAUDIBLE] Why is the standard using 2PAM instead of pulse position modulation?

PROFESSOR: Well because it's using these long codewords. Because back when we were analyzing it, we were analyzing it using two codewords. Those two codewords were orthogonal to each other. We could have made them plus one minus one and plus one minus one for the other one if we had in fact known what the channel was ahead of time, OK. That brings up a good point because you have to get to started somehow. I mean you have to know somehow, what's going on at the channel at the point where you start. I'm not going to talk about that.

If you have this multitap diversity than E sub b is going to be better than it would be. Otherwise, this says, if you can make the bandwidth bigger, as you make the bandwidth bigger, the bits are coming in faster. As the bits are coming in faster, this bit time is the separation between the taps and this digital model. So that as you make the bandwidth bigger, you automatically get more diversity. You get more taps in this discrete model. Looking at it another way, you have a certain frequency coherence in the channel. If you're sending a broad bandwidth, you get lots of different frequencies. If you get lots of different frequencies, there's a good chance that one of them is good. OK, so we are going to get some gain in diversity there. Typically we're going to get a relatively large amount of energy if the thing works.

Well, now we come to a question that we've been carefully avoiding all term. It's sort of the question of, if you make codewords orthogonal or if you make codewords

have nice structure at the input to a channel and the channel has some strange behavior, what's going to be the output of the channel? What we're doing here is making our codewords orthogonal at the input of the channel, we can't guarantee that they're still orthogonal at the output. What we do know is that PN sequences are almost orthogonal anyway. If you put PN sequence through anything, they still look like PN sequences. In other words, if you think of a PN sequence as an IID binary sequence, and you have a very long IID binary sequence, and you corrupt it in any way you want to, it still looks like an IID binary sequence. So, the point is, if you can make these orthogonal codewords look more like PN sequences, you can make them behave better at the output of the channel as well as the input to the channel. So you'll get something that looks orthogonal whether the channel essentially looks like a one tap channel or a multiple tap channel.

OK, we also have bandwidth to burn because we're only up to 307 kilohertz. What they decided to do was to take a PN sequence and run it at four times the speed of this bit sequence. So we have bits coming in at 307,000 bits per second. We're going to have a PN sequence which is changing four times for each bit time. So suddenly we're increasing the speed of this whole system by a factor of four. For every one bit, we multiply it by a sequence of four bits. So instead of getting a change every one over 307,000 seconds, you now get a change every one over 1.2288 millionths of a second. So, suddenly you're up to a sequence which is right at the bandwidth you want. I mean we want it to get up to 1.25 megahertz. Suddenly, we're up so close to that it's going to be a real challenge to build a filter, which will really make the output look like it's limited to this 1.25 megahertz. So we need very tight filters here. If you multiply this PN sequence , you're using the same PN sequence on all of the orthogonal codewords. If you think just a little bit about it, if these orthogonal codewords -- I'm I mean think of them still as binary sequences -- if each of them differ from each other when in half the places, when you multiply both of them by this PN sequence, they're still gonna differ in half the places. You still got the same affect out. They're still perfectly orthogonal before going through a channel but now you have a good chance that they're pretty close to orthogonal after going through the channel.

So, suddenly, we've got it up to the bandwidth we want to be at. There's another reason for the PN sequence. I mean, so far we've been thinking in terms of, how do you build a cell phone and a base station so they will work together? Actually, you want to build a base station and thousands of cell phones so they'll work together. So you want some way of making different cell phones look different from each other. What they use this PN sequence for is to make different cell phones look different. Each cell phone is going to start out on this PN sequence at a different point. The PN sequence is going to be generated by a 42 bit feedback shift register. If you don't know what a feedback shift register is, don't worry about it. Most of you probably do. A 42 bit feedback shift register, with the right kind of settings in it, is going to generate a periodic sequence with a period of two to the forty second minus one, which is about four times 10 to the twelfth. It doesn't repeat very often. I mean, this really looks like an IID sequence and binary digits. It really goes through every non zero 42 bit combination in this period here. It has very nice properties to it. But the property we want for this system is that in fact, when you give a cell phone a particular setting in that shift register, it starts it off generating a PN sequence which looks totally different from that generated by any other cell phone. That's the only thing that makes these different cell phones different. It's enough to make each one of them look orthogonal to each of the others.

OK, their basements sequences and thus passband wave forms are now going to be essentially orthogonal to those of other cell phones. In other words, at this point, we're sort of going into fantasy world. I mean, we started out generating 64 wave forms which were strictly orthogonal to each other. Then we said we want these wave forms to still be orthogonal after we go through this unknown channel. We said, well we can do this by making them look like PN sequences. But the other thing that PN sequences do, is it all of the possible wave forms for one cellphone will look orthogonal to the wave forms for all of the other cell phones. So that in fact, what's happening is, if you look at a base station, and the base station is trying to receive information from each cell phone, what it receives from one cell phone is going to be essentially independent of what it receives from the next cell phone.

Now if we think of the base station as trying to receive what's coming from one cell phone, what's the interference from the other cell phone going to look like? We're going to take thousands and thousands of sequences, which are now turned into wave forms, which look like IID wave forms, IID binary wave forms, and now we're adding together a large number of them. They fill up the whole spectrum uniformly. So they have a spectral density -- I mean it's a random process -- and it has a spectral density which is flat over this bandwidth of 1.25 megahertz. And you've added a lot of them together, so it looks like white noise. So the effect of these other cell phones is just like a small addition of white noise as far as trying to detect what's going on at one cell phone. Now, in fact you could be very clever here because, as soon as a base station decodes one cell phone, it can say, I know what that wave form was, subtracts it off from the received wave form, and uses that to help in decoding other wave forms. These standards do not do anything as sophisticated as that. Lots of papers talk about doing it. Information theorists love this idea. I think it's one of these ideas that sooner or later, ah going to become practical. As soon as it becomes practical, everyone will use it. What does it take to become practical? For someone to use it, because we all know it'll work. It's just a matter of actually doing it. But anyway, it's not done now. All of these interferes are just going to look like white noise as far as this one cellphone detection is concerned.

Now, if we compare this IS95 system with all of the narrow band systems that people use, at this point you can find the main difference between these different systems. When you use a narrow band system, things are a ranged so that only one cell phone is talking to one base station on one of these narrow bands of frequency. Namely the channels are very narrow but because they're narrow if you have two cell phones which are using that same bandwidth, they're just going to interfere with each other totally. You won't get through. So each cell phone which is using one base station has to be using a different narrow bandwidth. Cell phones using different adjoining base stations are also going to have to use different bands. In fact, that's a terribly complicated problem just because these base stations are put in sort of, random locations and trying to program things in such a way that you don't have two cell phones which are using adjacent base stations using the same

frequency. It's a real nightmare. So, you're going to have cell phones using the same base station, which are a little bit separated, but still because of all sorts of channeling effects and strange electromagnetic effects, base stations still receive data from these far away cell phones. They're also trying to receive data from the cell phone that's using that frequency. So you got a lot of interference in these narrow band systems; you got a lot of interference from cell phones using different base stations. In the IS95 system, you get more interference from cell phones that are using this particular base station because they're all using the same bandwidth. They all just look like white noise to each other. So they do interfere with each other, they raise the noise level. When you look at the interference with cell phones using other base stations, it's not catastrophic like it is with these narrow band systems. So what you effectively wind up with is that in the IS95 system, you have more interference within a base station but less interference between base stations. In the narrow band systems you have no interference within a base station but considerable interference between base stations. Which is better? Nobody really knows. Well except for the fact that almost all new systems are planning to use CDMA. It seems that most people have sort of agreed that you get less noise, overall, if you're using if you're using a CDMA system.

So that let's us sort of summarize where we are with all of this. We started out with 28.8 kilobits per second. We then going into this Walsh function, orthogonal wave form and coder, which comes out with 307.2 kilobits per second. We then multiplied by this long PN sequence, 42 bits long with four bits coming in here for each bit coming in here. Now we're implementing this and anybody with any sense who was trying to implement PN sequences with multipliers would just say, I don't want to turn things into to PAM sequences yet. I want to keep them binary and add them mod 2 because that's easier. So all of this is still mod 2 binary digits. This is mod 2 binary digits. This is mod 2 binary digits. The only thing we haven't talked about yet is now they throw in two other PN sequences. One on the real part of what's going on here and one on the quadrature part. The thing they're doing is taking this one sequence, splitting it into two parts; adding a PN sequence here, adding a PN sequence here, then doing the 2PAM, then filtering and sending this stuff out on the

cosign channel and the stuff out on the sign channel.

I've never had a very good sense of why you need these two PN sequences here. I can like sort of explain it to you in the following way. If you didn't have PN sequences here, what you'd be doing here is simply sending this same sequence twice. So you be sending it on the cosign channel, you'd be sending it on the sign channel, which would sort of be equivalent to sending it at 45 degrees. Once you realize that, you say well, I might as well leave this whole thing out and just use the cosign channel and nothing else. If I'm just using the cosign channel and nothing else it won't make much difference because the cosign just refers to the phase at this transmitter. I'm talking about many cell phones whose transmitters are all at different places relative to a base station. Therefore, they're all going to be using different phases so they're all going to fill up the signal space. The thing that, that doesn't take into account is the fact that if we do that we're not using all the degrees of freedom available to us. We're still just using the number of degrees of freedom that we got here, which are real degrees of freedom. When we do this and this, we got twice as many degrees of freedom. The PN sequences that we're looking at, which are the things that are orthogonal to each other. We have twice as many bits in a give interval of time when we have these two PN sequences as we do when we only use this one PN sequence. So in fact, if we view PN as IID binary digits, you get twice as many binary digits this way as you got before. Therefore, everything works better. You come much closer to being orthogonal with higher probabilities. If you didn't understand that's fine. It's not explained in the notes either. It's not explained in the IS95 standard. I think the explanation is right but it needs some fleshing out. OK, here's another interesting feature about this rake receiver. It doesn't use this digital model of the channel that we've talked about. It in fact, uses the analogue base band model. In other words, the base band model of the channel is the channel is represented by an analog filter. The analogue filter has impulses or almost impulses at different delays. Now what you'd like to do in this rake receiver here is instead of having taps at these sample times for the given bandwidth, they're going to adjust the taps so the taps are right on top of these things that look like impulses in all of these path returns. So the question is, what's the difference

between these two strategies?

I mean, we sort of show that the digital model was completely equivalent to the analog model. Then I cheated you; I didn't realize I cheated you until about nine o'clock this morning. I cheated you in the following way: I told you that if you model the channel in terms of the discrete set of taps according to the sampling therom, this was completely equivalent to trying to model it -- as far as a band limited input was concerned -- in terms of an analog filter. That's absolutely right if you use an infinite number of taps for the channel. If you now say that the impulse response of the channel is about one tap long and then you say, how many tabs do I need to represent it? It's an interesting question because if the impulse response is one tap long and you filter to look like a sign x over x, the impulse response can look like this. So you have one tap here, one tap here, which is zero, one tap here, one tap here, which is zero. If you've gotten your spacing off by half a cycle, what do you have then? You have a tap here. You have a tap here. You have a tap here. You have these taps which are going down as one over time. So in fact, modeling this kind of thing with one tap would be terrible. Now when you use the analog filter and you actually put these taps of the rake receiver at the places where you have actual physical responses you do much better. You particularly do better if the impulse response duration of the channel is the same order of magnitude as 1 over the bandwidth of the source that you're using. At that point this becomes an important issue. That's exactly the situation we have here because of the impulse response of these channels. The sample time on this discrete model it's about eight-tenths of a microsecond. If you think of how far light can travel in eight-tenths of a microsecond, it's about the same as the multipath spreads you would find outdoors when you're in a relatively rural area. If you're in a city area, multipath spreads are going to be even smaller. So if you want to a rake receiver to work well you should really have a rake receiver which is trying to zero in on the actual physical response times. This is what this system does. They did it right and I did it wrong.

OK, final thing. I told you I wanted to talk about why we did scrambling. This rake decoder, in other words, this decision device, which every six bits tries to make a decision on these 64 different orthogonal wave forms. You can just make hard

decisions which gives you six bits out each time. You can produce six bits out also with some likelihood about how likely they are to be right. So what you'd like is some information which is essentially the logarithm of the probability of being correct over the probability of error. You can estimate that in your rake receiver. You can estimate it by knowing whether you're making a close call or whether there's one orthogonal wave form which is far more likely than all of the others. So if you're making a close call, then you produce an output that says, I think it's this output but I'm not very sure. This is what goes into this viterbi decoder. The viterbi decoder, since it's operating on these states -- in other words, it's operating on what it visualizes is being the state of the viterbi decoder -- it can actually deal with these log likelihood ratios as well as anything else. That's one of the reasons you want to use a viterbi decoder because most coding devices that people have invented don't work very well with analog signals. This does work well with analog signals.

The problem is, anytime you make a mistake on the 64 orthogonal wave forms, you have a six bit batch of digits, a six bit block, and you're going to be wrong, typically in three of them. well, you're going to be wrong in a random number of them. Half the time you're going to be wrong. What this means is if you put this data into a decoder, what you're going to find is that every time there's an error in the in the rake receiver, you're going to have a burst of errors, some random bunch of errors over six bits. This is typically going to be in the order of three bits an error all at once. If you look at the structure of that decoder, anytime you got a large number of errors in all at once, it's going to screw the thing up terribly. I'll give you one example of that.

You can take a much simpler coding example. Suppose you transmit either all zeros, seven zeros or seven ones. At the receiver there's a certain probability that you make an error on each one of these bits. You first make a decision on each bit and then after that you make a decision on the whole code word. If the bit errors are independent of each other, then you make an error in the overall detection of these code words only when you make four errors out of seven. Or five errors out of seven which is negligible. So you calculate, what's the probability of making four

errors out of seven here? Well it's the number of combinations of seven things taken three at a time which is 35 times p fourth, p to the fourth power which is the probability of making four errors. If those bits are perfectly correlated mainly, if whenever you make one bit error you make all seven bit errors, then the probability of error is p If p is relatively small, p is always a lot bigger than 35 times p to the fourth. In other words, the lesson here is that highly dependent errors cause trouble for any kind of decoder. Now you see why you had the scrambler in here, why you have the interleaver. You Have the interleaver to take these highly correlated errors which are coming out of the rake receiver where you make blocks of six errors all tend to appear together. You're scrambling them out so they're widely separated so that the viterbi decoder is able to deal with them. That's the last piece of the whole thing. I could talk a little bit about how the viterbi decoder works but I think you're probably exhausted by now. I think I will stop at this point. Than you all for your attention all term and good luck on the final.