Extra Credit Project

Forward Error Correction Utilizing Convolutional Codes and the Viterbi Algorithm

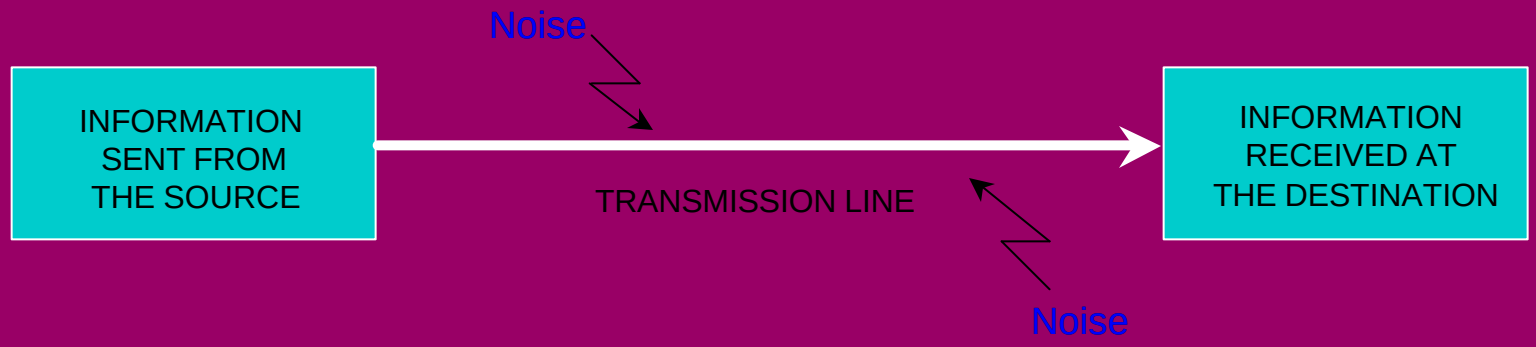
University of Southern California
EE552 Advanced Switching Theory and Logic Design
Instructor: Dr. James Ellison

By: Roger E. Doty

Presentation Contents.

- How Errors Get Into Transmitted Signals.
- Methods for Handling Errors in Transmitted Signals.
- A Typical Error Correcting Communication System.
- Convolutional Codes - Terminology.
- Forward Error Correction (FEC) Applications.
- Convolutional Encoding: Modulo-Two Addition.
- Convolutional Encoding: An Example.
- The Viterbi Decoder Algorithm.

How Errors Get Into Transmitted Signals.



Data sent from the source picks up noise as it passes along the transmission line or through the air.

The noise causes the data to become corrupted. The data is in the form of binary bits that may get “flipped” so that logical 0’s become 1’s and logical 1’s become 0’s !!!!

Some method must be used to ensure that the correct data is received at the other end of the transmission.

Methods for Handling Errors in Transmitted Signals.

There are two basic methods for handling noise induced errors in transmitted signals.

1.) Errors can be detected at the receiving end and the information re-transmitted.

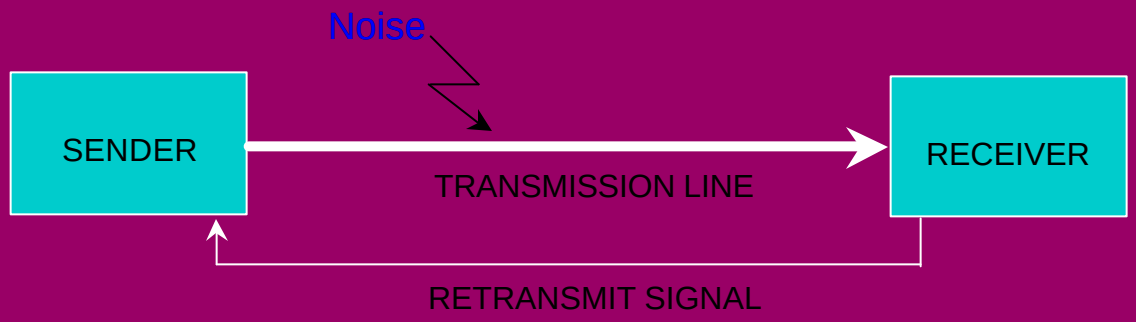
2.) Errors can be detected at the receiving end and the information is corrected using Forward Error Correction (FEC) techniques.

Question:
If an error is detected, why don't we just retransmit the data instead of adding error correction circuitry?

Methods for Handling Errors in Transmitted Signals.

Answer: Its a matter of throughput.

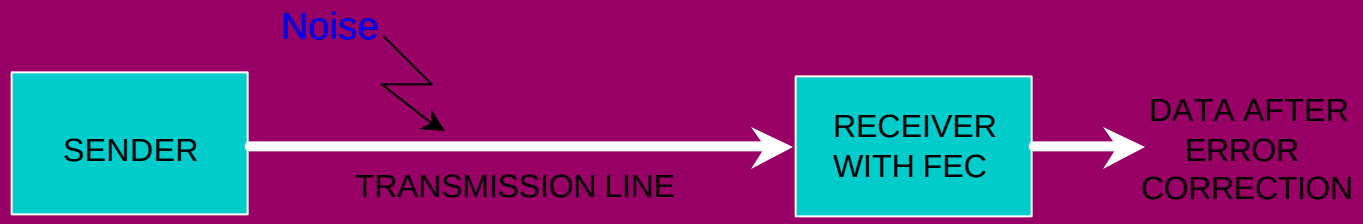
Retransmission techniques require a two-way link. The error is detected at the receiving end and a signal is sent to the transmitter to retransmit the data. This causes the same data to be transmitted twice. While this is happening no new data is being sent. This causes a decrease in data throughput.



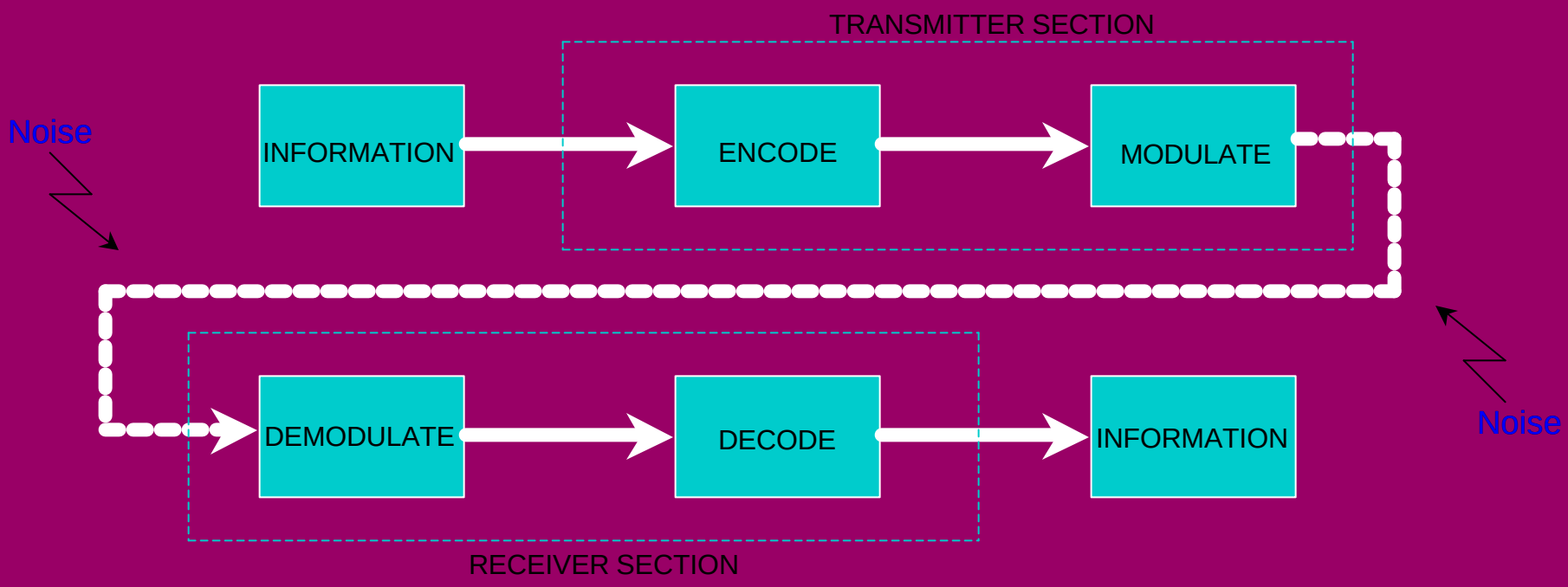
Methods for Handling Errors in Transmitted Signals.

Forward error correction requires only a one-way link, and its parity bits target both error detection and correction.

Data can continue to be transmitted and if an error is detected encoded information sent along with the data is used to correct for the error.



A Typical Error Correcting Communication System.



An FEC communication system adds information at the transmitter for error correction of corrupted data at the receiver.

Convolutional Codes - Terminology.

The following definitions are vital to understanding convolutional codes.

Constraint length: Denoted by K, is the number of *k*-bit shifts that a single bit can influence in the encoder output. Simply put, constraint length is the number of bits that the encoder uses to encode *n* bits.

Code rate: K bits enter the encoder, and *n* bits leave the encoder. So the code rate is K/*n*. Error-correcting capability (and complexity) increase as K/*n* decreases.

Example: In rate 1/2 code (1 bit in, 2 bits out), each bit entering the encoder leaves the encoder with 2 bits.

Typical values for code rate are: 1/2, 1/3, 2/3.

Forward Error Correction (FEC) Applications.

As previously discussed, forward error correction is used at the receiving end to detect errors and correct for them.

Some applications include CD and DVD players, HDTV, Data storage systems, Wireless communications, Satellite communications, and Modem technologies.

Designers have implemented FEC circuits in Application Specific Integrated Circuits (ASICs), Digital Signal Processors (DSPs), and Field Programmable Gate Arrays (FPGAs).

FEC techniques can be realized for both Block codes (Reed-Solomon codecs) and for Convolutional codes (Viterbi codecs).

Convolutional Encoding: Modulo-Two Addition.

Convolutional encoding of data is accomplished using shift registers and combinatorial logic that performs modulo-two addition.

The combinatorial logic is usually in the form of cascaded exclusive-OR gates.

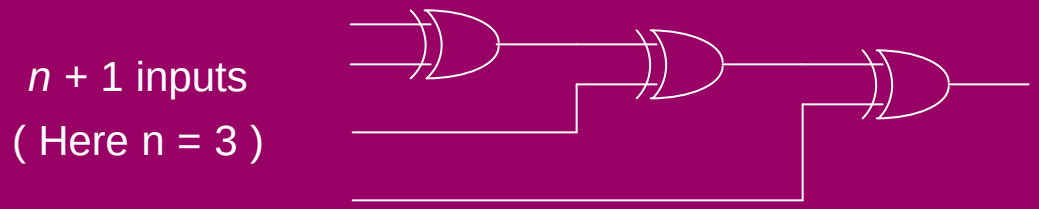
Recall the exclusive-OR function and it's truth table shown here ----->

Exclusive - OR Function		
Input A	Input B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

If we cascade a series of *n* two-input exclusive-OR gates,with the output of the first one feeding one of the inputs of the second one, the output of the second one feeding one of the inputs of the third one, etc., the output of the last one in the chain is the modulo-two sum of the *n* + 1 inputs.

Convolutional Encoding: Modulo-Two Addition.

Cascaded exclusive-OR gates.



This is a common symbol for a modulo-two adder ---> ⊕

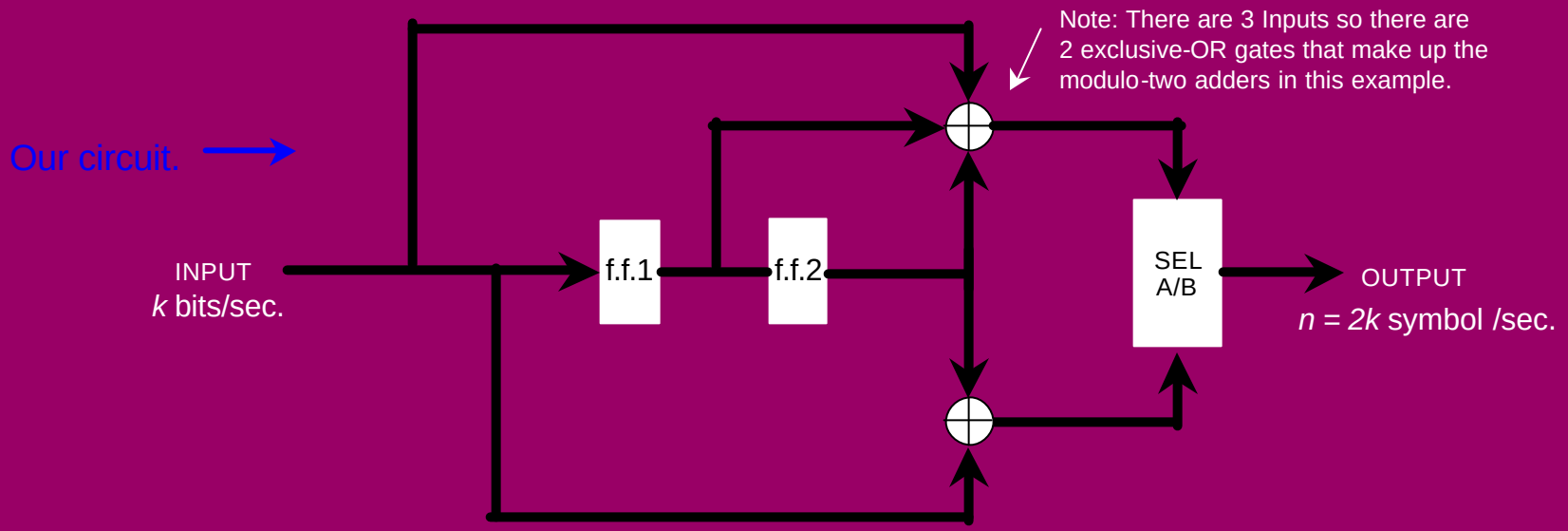
In addition to the modulo-two adder, Flip-flops are used to make the shift register.

We'll also need an output selector to toggle between the two modulo-two adders.

The next slide shows a convolutional encoder for rate 1/2, K = 3, m = 2 code. →

Convolutional Encoding: An Example.

Lets build an encoder!!



1.) Data bits are provided at a rate of *k* bits per second. The input bit is stable during the encoder cycle.

2.) When the input clock edge occurs, the output of flip-flop 1 is clocked into flip-flop 2, the previous input bit is clocked into flip-flop 1, and a new input bit becomes available.

3.) The outputs of the modulo-two adders become stable.

Example Continued →

Convolutional Encoding: Example.

1.) The output selector (SEL A/B) now cycles through two states.
First it selects the top modulo-two adder and sends its result to the output.
Second it selects the bottom modulo-two adder and sends its result to the output.

The shift register connections to the upper and lower modulo-two adders generate the code generator polynomials, 111 and 101. This code has been determined to be the best code for rate 1/2, K = 3.

Lets do an example of an input data stream and see what it's corresponding output data stream will be.

Example Continued →

Convolutional Encoding: Example.

Input/Output Example.

Let's put in an input sequence = 010111001010001.

Initially we clear the flip-flops.

At the first clock, the first bit 0 is available to the encoder. The flip-flop outputs are both 0, so the output of the encoder is 00 (remember we toggle the output select switch).

At the second clock, the second input bit is available to the encoder. The left-hand flip-flop clocks in the previous bit, a 0, and the right hand flip-flop clocks in the 0 output by the left-hand flip-flop.

Now the inputs to the top modulo-two adder are 100, So it outputs a 1. The inputs to the bottom modulo-two adder are 10, so the output is a 1.

The encoder now outputs 11 for the channel symbols.

Example Continued →

Convolutional Encoding: Example.

Input/Output Example.

Now let's look at a third clock. At the third clock, the next input bit, 0, is available to the encoder. The left-hand flip-flop clocks in the previous bit, a 1, and the right-hand flip-flop clocks in a 0 from two "bit-times' before.

Now the inputs to the top modulo-two adder are 010, So it outputs a 1. The inputs to the bottom modulo-two adder are 00, so the output is a 0.

The encoder now outputs 10 for the channel symbols.

And so on, and so on, until the entire input sequence is encoded.

If this is kind of confusing you could generate a timing diagram to illustrate the outputs of the various components although that could take a little bit of work !!

Example Continued →

Convolutional Encoding: Example.

After the entire input sequence has been presented to the encoder the output sequence produced would be:

00 11 10 00 01 10 01 11 11 10 00 10 11 00 11

- We can think of our encoder as a simple state machine.

Our example encoder has two bits of memory (remember the flip-flops?)

Therefore we have four possible states.

Example Continued →

Convolutional Encoding: Example.

Lets give the right-hand flip-flop a binary weight of 2^0 and the left-hand flip-flop a binary weight of 2^1.

The possible states are then given as 00, 01, 10, 11.

- Initially the encoder is in the all-zeros state.
- If the first input bit is a zero, the encoder stays in the all zeros state.
- If the first input bit is a one, the encoder goes to the '10' state. Then if the input bit is a '0', the encoder goes to the '01' state. Otherwise it transitions to the '11' state.

Example Continued →

Convolutional Encoding: Example.

This is more easily shown with a state transition table along with the encoder outputs for each transition.

Current State	Next State, IF	
	Input = 0	Input = 1
00	00,00	10,11
01	00,11	10,00
10	01,10	11,01
11	01,01	11,10

In the Next State column the left term is the next state given the current state and input, the right term is the output for the state transition.

With this table we can completely describe the behavior of our rate 1/2, K = 3 convolutional encoder.

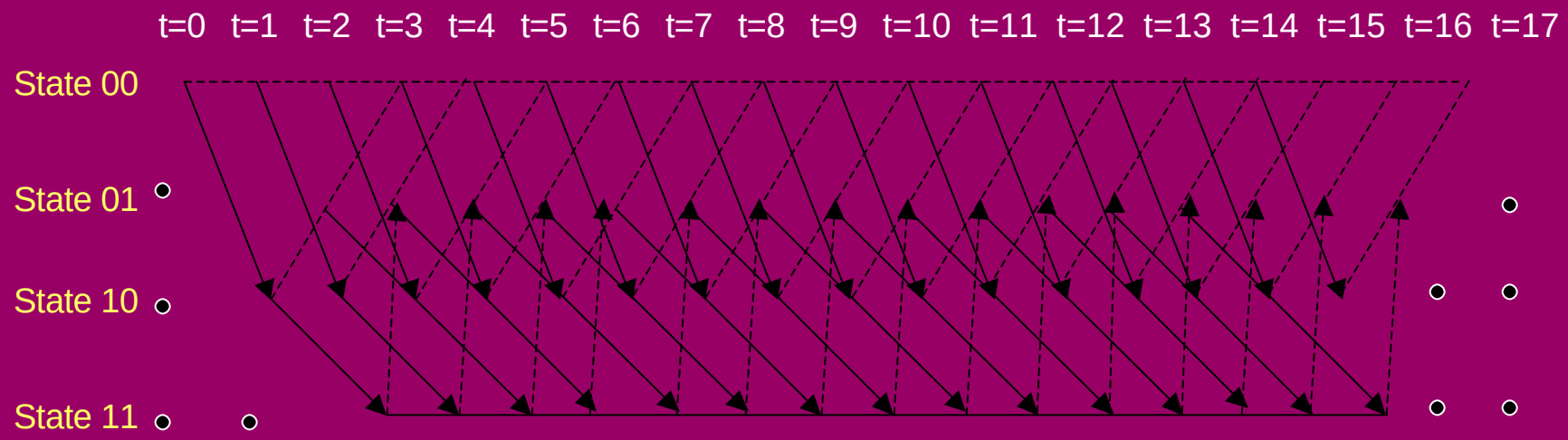
Example Finished

The Viterbi Decoding Algorithm.

Now that we have encoded our message we have to decode and recover it at the other end.

The single most important concept to understanding the Viterbi algorithm is the trellis diagram.

Shown below is the trellis diagram for our 1/2 K=3 encoder.



The Viterbi Decoding Algorithm.

The four possible states of our encoder are depicted as four rows of horizontal dots.

There is one column of four dots for the initial state and one for each time instant during the message.

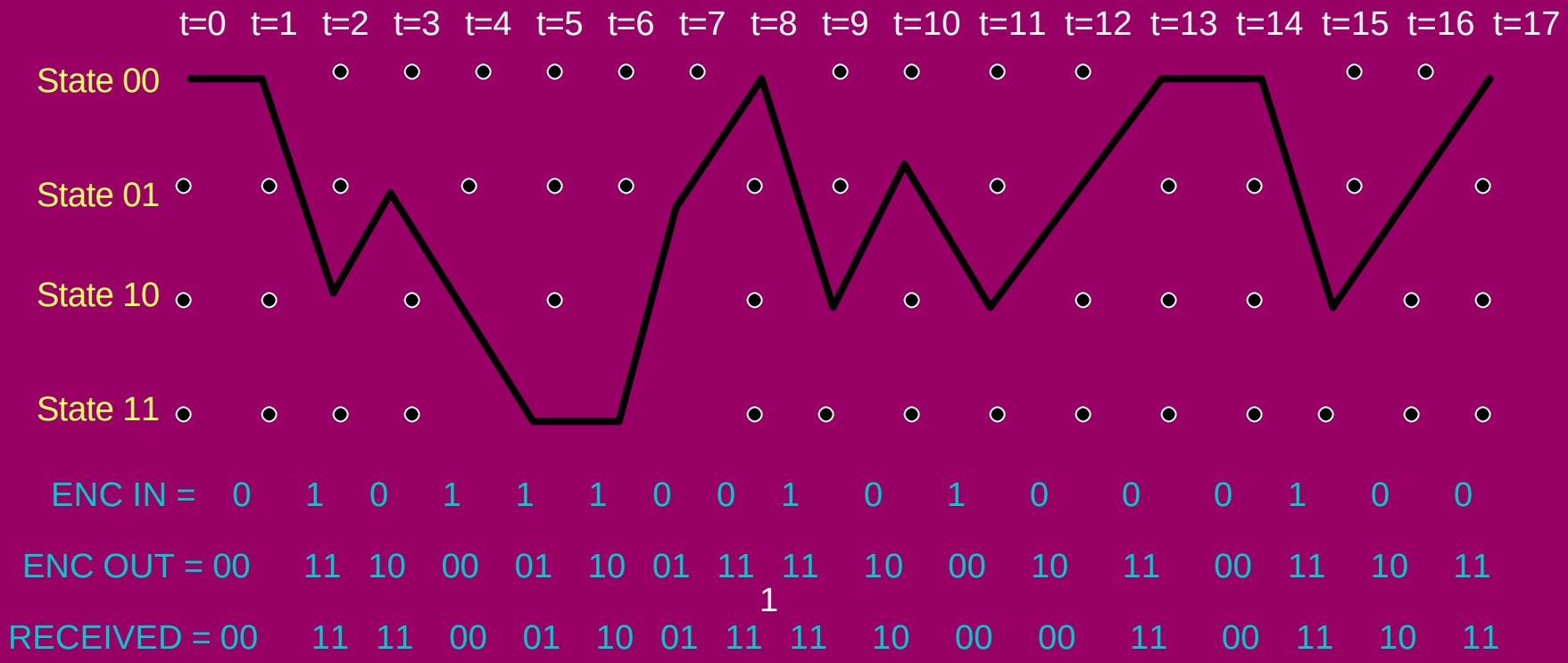
There are 17 time instants which is made up of our 15-bit message plus 2 memory flush bits.

- t=0 represents the initial condition of the encoder.
- The solid lines represent state transitions when the input bit is one.
- The dotted lines represent state transitions when the input bit is zero.

The Viterbi Decoding Algorithm.

- If you look closely you can see the correspondence between the arrows in the trellis diagram and the state transition table from the encoder development example.

Below is a diagram showing the actual states our trellis reaches during the encoding of our 15-bit message.



The Viterbi Decoding Algorithm.

Let’s take a look at how our Viterbi decoding algorithm actually works.

Suppose we received our encoded message with a couple of bit errors.

Each time we receive a pair of channel symbols, we’re going to compute a metric to measure the distance between what we received and all possible channel symbol pairs we could have received.

The Viterbi Decoding Algorithm.

Now lets move along our trellis example

Going from t=0 to t=1, there are only two possible channel symbol pairs we could have received: 00 and 11. That's because the convolutional encoder was initialized to the all-zeros state, and given one input bit = 1 or 0, there are two states we could transition to and two possible outputs from the encoder, 00 and 11.

Hamming distance is the metric were going to use between the received channel pair symbol and the possible channel pair symbols.

The Hamming distance is computed by counting the bits that are different between the received channel symbol pair and the possible channel symbol pairs.

The results can be either zero, one, or two.

Branch metrics are the Hamming distance values at each time instant for the paths between the states at the previous time instant and the states at the current time instant.

The Viterbi Decoding Algorithm.

At t=1, we received 00. We could only have received 00 and 11. The Hamming distance between 00 and 00 is zero and the Hamming distance between 00 and 11 is two.

The branch metric for the branch from state 00 to state 00 is zero.

The branch metric for the branch from state 00 to state 10 is two.

The accumulated error metric for the other two states is undefined.

At t=2, we received 11. The possible channel symbols we could have received going from t=1 to t=2 are 00 from state 00, 11 from state 00 to state 10, 10 from state 10 to state 01, and 01 from state 10 to state 11.

Hamming distance: 00 to 11 = 2
11 to 11 = 0
10 or 01 to 11 = 1

The Viterbi Decoding Algorithm.

We add these branch metric values to the previous accumulated error metric values associated with each state.

At t=3, now there are two different ways that we could get from each of the four states that were valid at t=2 to the four states that are valid at t=3. Here we must compare the accumulated error metrics associated with each branch. Then we discard the larger one of each pair of branches leading into a given state.

If the members of a pair of accumulated error metrics going into a particular state are equal, we just save that value.

As we move along, were keeping a sort of “history” of the predecessor-successor. The predecessor that survives is the one that has the Lower branch metric.

What do we do if the two accumulated error metrics are equal?
It really doesn’t matter which one you pick to survive. If you want to be consistent you could always pick the upper or lower branch.

The Viterbi Decoding Algorithm.

Add-Compare-Select Operation

This is the operation that we have been going through to process our trellis diagram. The steps are as follows:

1.) Add the previous accumulated error metrics to the new branch metrics.

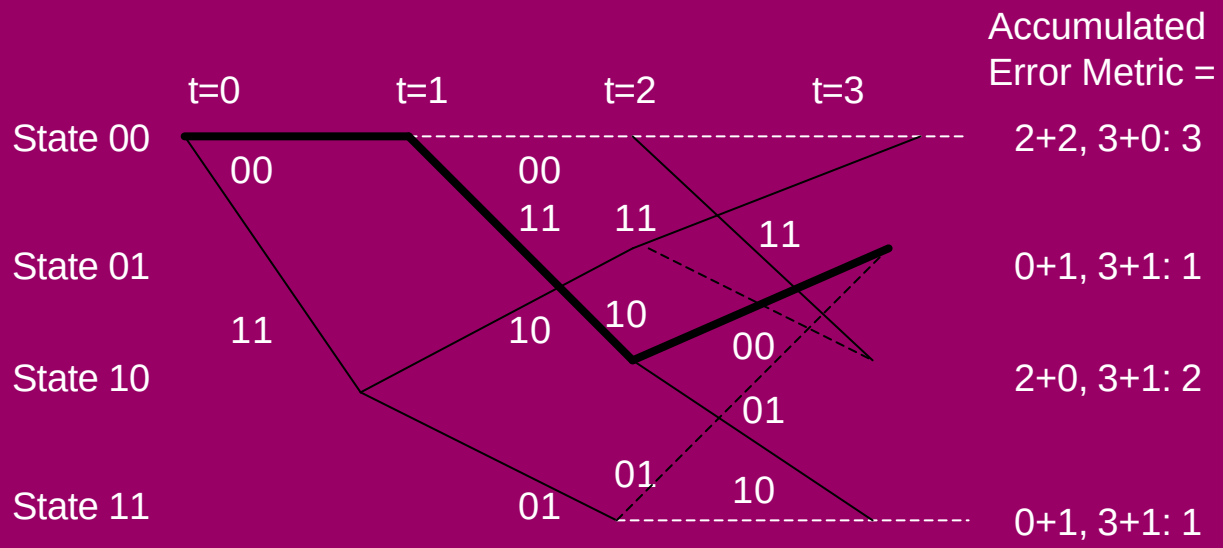
2.) Compare the results.

3.) Select the smallest accumulated error metric to be retained for the next time instant.

The next slide shows the results of our trellis diagram for t=0, t=1, t=2, and t=3.

The Viterbi Decoding Algorithm.

Trellis diagram through t=0, t=1, t=2, t=3.



ENC IN = 0	1	0
ENC OUT = 00	11	10
RECEIVED = 00	11	11

The Viterbi Decoding Algorithm.

From the previous diagram it should be noted that the third channel symbol pair we received had a one-symbol error.

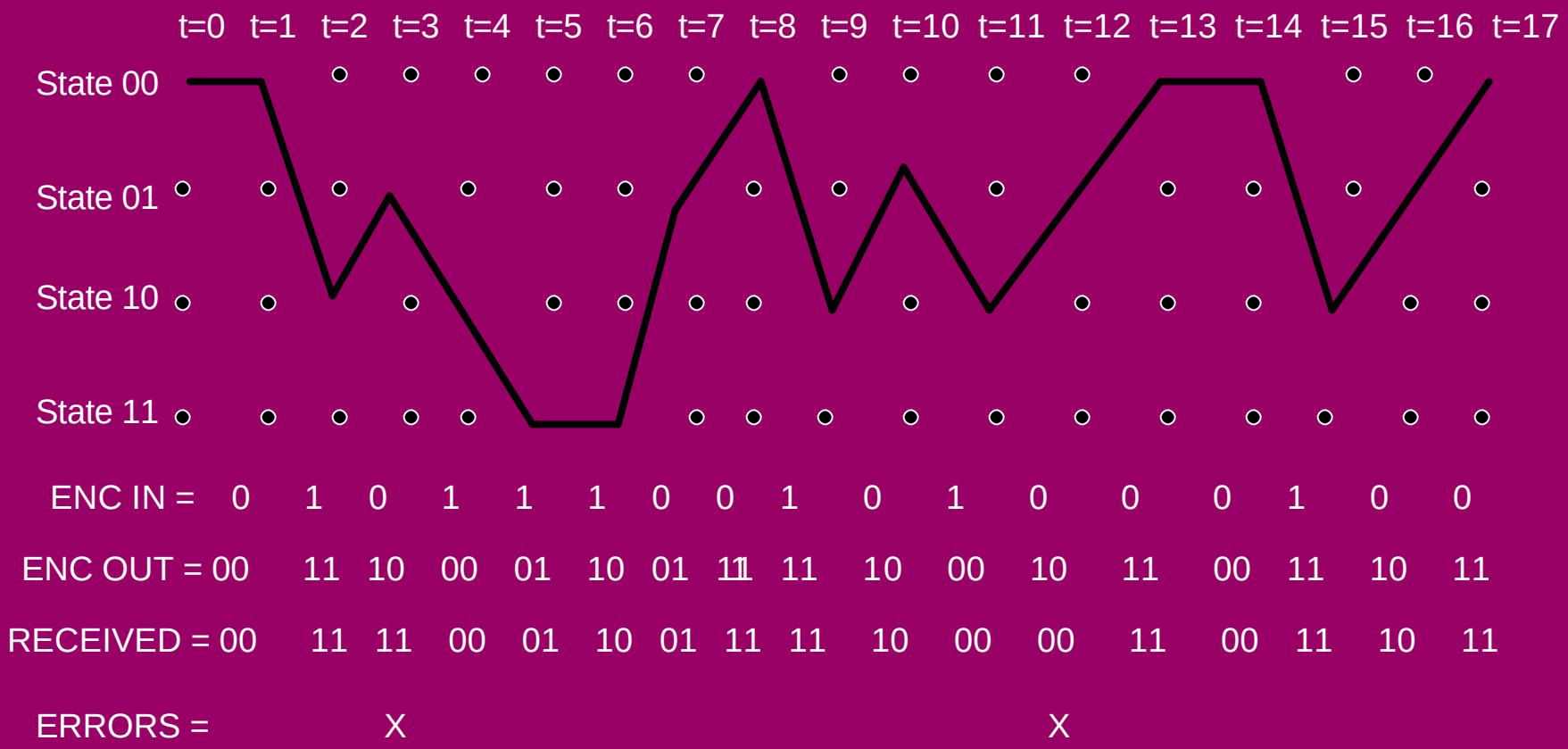
The smallest accumulated error metric is a one, and there are two of these.

You will notice that our trellis is beginning to take shape. The procedure for t=4, t=5, etc. is the same, each time carrying our accumulated error metric.

If we carry the process through to t=17, the trellis will look like the diagram previously shown showing the states through t=17.

That diagram is shown again on the next slide 

The Viterbi Decoding Algorithm.



The Viterbi Decoding Algorithm.

What we've done so far and what's left to do.

We began the decoding process by building the accumulated error metric for some number of received channel pairs. Now that this information is Built up, the Viterbi decoder is ready to recreate the sequence of bits that Were input to the convolutional encoder.

This is summarized in the following steps:

1.) Select the state having the smallest accumulated error metric and save the state number of that state.

2.) Trace back: Now working backwards through the state history table, for the selected state, select a new state which is listed in the state history table as being the predecessor to that state. Save the state number of each selected state.

Continued →

The Viterbi Decoding Algorithm.

3.) Now work forward through the list of selected states saved in the previous steps. Look up what input bit corresponds to a transition from each predecessor state to its successor state. That is the bit that must have been encoded by the convolutional encoder!!

- The following table shows the accumulated metric for the full 15-bit message.

t=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State 00		0	2	3	3	3	3	4	1	3	4	3	3	2	2	4	5	2
State 01			3	1	2	2	3	1	4	4	1	4	2	3	4	4	2	
State 10		2	0	2	1	3	3	4	3	1	4	1	4	3	3	2		
State 11			3	1	2	1	1	3	4	4	3	4	2	3	4	4		

Note that for our Viterbi decoder example, the smallest accumulated error metric in the final state Indicates how many channel symbol errors occurred.

The Viterbi Decoding Algorithm.

This next table for the state history shows the surviving predecessor States for each state at each time *t*.

t=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
State 00	0	0	0	1	0	1	1	0	1	0	0	1	0	1	0	0	0	1
State 01	0	0	2	2	3	3	2	3	3	2	2	3	2	3	2	2	2	0
State 10	0	0	0	0	1	1	1	0	1	0	0	1	1	0	1	0	0	0
State 11	0	0	2	2	3	2	3	2	3	2	2	3	2	3	2	2	0	0

The following table shows the states selected when tracing the path back Through the survivor state table above.

t=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0	2	1	0

The Viterbi Decoding Algorithm.

Using our table that maps the state transitions to the inputs that caused them, we can create the original message.

Current	Input was, given next state			
State	00=0	01=1	10=2	11=3
00=0	0	x	1	x
01=1	0	x	1	x
10=2	x	0	x	1
11=3	x	0	x	1

X denotes an impossible transition from one state to another.

Now we have everything we need to recreate the original message.

Using the above three tables and our final trellis diagram, we can tabulate the results below.

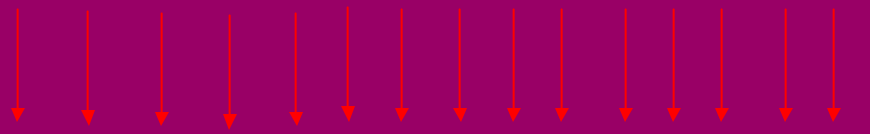
t=	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	0	1	1	1	0	0	1	0	1	0	0	0	1

The Viterbi Decoding Algorithm.

Lets compare our result to the original message.

Result →

t=	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	0	1	1	1	0	0	1	0	1	0	0	0	1



Original Message →

0 1 0 1 1 1 0 0 1 0 1 0 0 0 1

It Works !!!

The Viterbi Decoding Algorithm.

This presentation has shown how noise can be introduced Into a communications channel and the problems that it Can cause.

We have shown how to use convolutional encoding to encode a transmitted message and the Viterbi algorithm to recover that message.

An example was used to show how this is done and the Result was compared with the original message.

References

1.) “Tutorial on Convolutional Coding with the Viterbi Algorithm”.
By: Chip Fleming of Spectrum Applications.
Spectrum Applications Inc. Technical Article.

2.) “Self-correcting codes conquer noise. Part 1: Viterbi codecs”.
By: Syed Shahzad, Saqib Yaqub, and Faisal Suleman.
Electronic Design News Article. EDN Magazine. Feb. 15, 2001.

3.) “Convolutional Codes and the Viterbi Decoding Algorithm”.
By: Tsu-Chien Shen.
EE552 Extra Credit Project Report from Previous Semester.

4.) “Introduction to theTheory of Error-Correcting Codes”.
By: V. Pless.
3rd ed. New York: John wiley & Sons, 1998.

5.) “Digital Communications”.
By: J.G. Proakis.
3rd ed. Boston: WCB/McGraw-Hill, 1995.

Referenced Material

Slides 3 – 7
References 2,4,and 5

Slides 7, 8, and 9
References 2

Slides 19 – 35
References 1,3 and 4

The example developed in this presentation is based on reference 1.