1.0 Prologue:  
Convolutional codes, why should complicate our lives with them

People use to send voice waveforms in electrical form over a twisted pair of wires. These telephone voice signals had a bandwidth of 4KHz. If the channel polluted the signal with a bit of noise, the only thing that happened was that the conversation got a bit noisier. As technology developed, we digitized the voice signals at 8000 samples per second (twice the highest frequency to prevent aliasing) and transmitted the bits. If noise corrupted a few bits, the corresponding sample value(s) would be slightly wrong or very wrong depending on whether the bad bits were near the most-significant-bit or least-significant-bit. The conversation sounded noisier, but were still discernible. Someone saying “cat” will not be thought to have said “dog,” and probably would not even be thought to have said “caught.”

When people started to send data files rather than voice, corrupted bits became more important. Even one wrong bit could prevent a program from running properly. Say the noise in a channel was low enough for the probability of a bad bit to be $1 \times 10^{-3}$ i.e. the chances of a bit being correct is $0.999999999$ (nine 9’s). The chances of 1000 bits being all correct is $0.999999$ (six 9’s) and the chances of $10^6$ bits being all correct is $0.999$ (three 9’s). A 1 megabyte file ($8 \times 10^6$ bits) has almost a 1% chance of being corrupted. The reliability of the channel had to be improved.

The probability of error can be reduced by transmitting more bits than needed to represent the information being sent, and by convolving each bit with neighbouring bits so that if one transmitted bit got corrupted, enough information is carried by the neighbouring bits to estimate what the corrupted bit was. This approach of transforming a number of information bits into a larger number of transmitted bits is called channel coding, and the particular approach of convolving the bits to distribute the information is referred to as convolution coding. The ratio of information bits to transmitted bits is the code rate (less than 1) and the number of information bits over which the convolution takes place is the constraint length.

For example, suppose you channel encoded a message using a convolution code. Suppose you transmitted 2 bits for every information bit (code rate=0.5) and used a constraint length of 3. Then the coder would send out 16 bits for every 8 bits of input, and each output pair would depend on the present and the past 2 input bits (constraint length =3). The output would come out at twice the input speed.

Since information about each input bit is spread out over 6 transmitted bits, one can usually reconstruct the correct input even with several transmission errors.

The need for coding is very important in the use of cellular phones. In this case, the “channel” is the propagation of radio waves between your cell phone and the base station. Just by turning your head while talking on the phone, you could suddenly block out a large portion of the transmitted signal. If you tried to keep your head still, a passing bus could change the pattern of
bouncing radio waves arriving at your phone so that they add destructively, again giving a poor signal. In both cases, the SNR suddenly drops deeply and the bit error rate goes up dramatically. So the cellular environment is extremely unreliable. If you didn’t have lots of redundancy in the transmitted bits to boost reliability, chances are that digital cell phones would not be the success they are today. As an example, the first digital cell system, Digital Advance Mobile Phone Service (D-AMPS) used convolution coding of rate 1/2 (i.e. double the information bit rate) and constraint length of 6. Current CDMA-based cell phones use spread-spectrum to combat the unreliability of the air interface, but still use convolution coding of rate 1/2 in the downlink and 1/3 in the uplink (constraint length 9). What CDMA is, is not part of this lab. You can ask the TA if you are curious.

2.0 Example of Convolution Encoding

This is a convolution encoder of code rate 1/2. This means there are two output bits for each input bit. Here the output bits are transmitted one after another, two per clock cycle.

The output \( z_1 = x(n) \oplus x(n-1) \oplus x(n-2) \).

Here \( x(n) \) is the present input bit, \( x(n-1) \) was the previous (yesterdays) bit, etc.

The output \( z_2 = x(n) \oplus x(n-2) \).

The input connections to the XORs can be written as binary vectors \([1 \ 1 \ 1]\) and \([1 \ 0 \ 1]\) are known as the generating vectors or generating polynomials for the code.
2.1 The Encoder as a Finite-State Machine

The correlation encoder can be described as a Mealy machine. The state is the two bits in the shift register.

Let the first input bit to the shift register be $x(n) = 1$, and let the flip-flops be reset to zero so $x(n-1) = x(n-2) = 0$. Then:

State $= 00 = S_0 = [x(n-1), x(n-2)]$
Output $z = [z_1, z_2]$

\[ z_1 = x(n) \oplus x(n-1) \oplus x(n-2) = 1 \oplus 0 \oplus 0 = 1 \]
\[ z_2 = x(n) \oplus x(n-2). = 1 \oplus 0 = 1 \]
\[ z = [z_1, z_2] = 11 \]

After the clock, state bit $x(n-1) = 0$ will shift right into $x(n-2)$, the input $x(n) = 1$ will shift right into $x(n-1)$, and the next state will be $10 = S_1$.

2.2 The Trellis Encoding Diagram

To get the trellis diagram, squash the state diagram so $S_0, S_1, S_2, S_3$ are in a vertical line. This line represents the possible states at time $t=n$ (now). Make the time the horizontal axis. Put in another line of states to show the possible states at $t=n+1$.

Then add the transitions to the diagram. Make them all go from states $@ t=n$ to states $@ t=n+1$. Thus the self-loop at state $S_0$ in the state graph becomes the horizontal line from $S_0 @ t=n$ to $S_0 @ t=n+1$ in the trellis diagram.

The complete trellis diagram extends past $t=1$ to as many time steps as are needed.

Suppose the encoder is reset to state $S_0=00$, and the input is $1,0,1,1,0,0$. By following the trellis one sees that the output is $11 \ 10 \ 00 \ 01 \ 01 \ 11$. Also it passes through states $S_0, S_1, S_2, S_1, S_3, S_2$ ending in $S_0 @ t=6$. 
2.3 Lab and Problem Rules

The Convolution encoder/Viterbi decoder design problem will be done by a group of three persons.

The number of exercises is (usually) divisible by three so one person in each group can do every third problem and thus do one-third of the exercises. The three are to be submitted together with the name of the person doing each part attached to the part.

Five marks will be assigned for each person’s questions and will be given to the person in the group answering the question. Two person groups should take turns answering the odd questions which will be assigned three marks. This applies to temporary two person groups, groups where one member goes off. Another member can get three extra marks by doing his/her questions.

One common mark will be assigned to coordination within the group. Do they use common symbols? Do they hand the assignments in at the same time attached together? Do they refer to the other questions where appropriate? Violation of any one of the above may cost each group member his/her common mark.

All members of the group are responsible for knowing how to do each exercise.

Related problems will appear on examinations.

The problems and labs have subtle, and also not so subtle, changes from last year. One way to lose marks quickly is to submit answers taken from last year. The penalty will be zero for the question(s) involved and a 75% reduction in the mark of the whole group.

2.4 First Exercise: A Convolution Encoder.

1. Problem: Encoding a number
   • Take the last 4 digits in your student (the least significant digits).
   • Convert them to a hexadecimal number (Matlab has a function dec2hex).
   • Convert the hexadecimal number to binary (12 to 16 bits).
   • Use this as data for the encoder below. Feed in the least significant bit first. Also reset the shift register to 00 before you start.
   • Calculate the output bits and states when one encodes these bits using a code rate $1/2$, constraint length 3 encoder with generating vectors $[111]$ and $[101]$.
   Tabulate how the state and output values change with each clock cycle.

<table>
<thead>
<tr>
<th>Clock cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>input bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>shift reg (state)</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output $[z_1,z_2]$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2. Problem: Draw the circuit for an encoder which has:
   a code rate $1/2$, constraint length 4, generating vectors $[1101]$ and $[1111]$, where the “0” means no connection to $x(n-2)$.

3. Problem: Draw the state graph for the above constraint length 4 encoder. Draw the first 3 time steps of the trellis diagram for the above constraint length 4 encoder.
2.5 First Lab: Design a Convolution Encoder in Verilog

The constraint-length 4 encoder circuit is a finite-state machine. It is also a shift register. One can code it following the standard finite-state machine model with three-bit states, or as a shift register.

```verilog
reg [2:0] State; // (shift-reg length) = (constraint length - 1)
always @(posedge clk)
begin
    State <= State >> 1; // Right shift 1 position
end
```

The test bench is not part of the circuit. It supplies the input signals and may compare output signals with those that are expected. Test benches are easier to write and use if they are synchronous. This means they always send out signals slightly after the clock (or at least never at the same time as the clock). It also means a writing style with few #n delay.

```verilog
initial begin
    . . . #11 x=1;
    #10 x=0;
    #10 x=1;
    #20 x=0;
    #10 x=1;
    . . .
end
```

2.5.1 A Synchronous Test Bench

In a synchronous test bench, the test signals are timed by @(posedge clk) statements rather than each having its own timing. There are only two delays here, one to set the clock period, and the other to delay the input signal x so it’s changes are obviously past the clock edge.

Note there are things not included here, like reset.

```verilog
module SyncTestBench;
    reg [1:8] data; //Fill this with the data stream to be encoded.
    // Note the first bit is on the left
    reg x, clk;
    integer I; // Use integers for counter indexes

    initial begin
        I=1;
        data=8'b1010_1101; // Underscore has no meaning except
        // to visually space the bits.
        clk=0;
        forever #5 clk=~clk;
    end

    // send in a new value of x every clock cycle
    always @(posedge clk)
    begin
        if (I==9) $finish; // Stop the simulation when one runs out of data.
        // The #1 makes x change 1ns after the clock and never on the clock edge.
        // The nonblocking symbol "<=" on I ensures that any other clocked module using
        // I will grab the same I as this procedure, that is before I is updated to I+1.
        x<= #1 data[I];
        I<=I+1;
    end
```

FIGURE 6
end
endmodule

For the constraint length 3 system, you must have the test bench automatically compare your answer with the result you obtained from your student number.

1. Write a finite-state machine encoder for the constraint length 3 system.

   ```
   always @( state or x)
   nxtstate= {state[2:0],x};
   . . .
   always @(posedge clk or posedge reset)
   . . .
   state <=nxtstate
   ```

2. Write a shift-register based encoder for the constraint length 4 system (Sect 2.3 prob 2). Generating vectors [1101] and [1111].

   ```
   always @(posedge clk or posedge reset)
   . . .
   state <=state<< 1; // Left shift 1 position
   state[0] <= x;  // Overwrite the zero shifted in on the previous line.
   ```

3. Write a synchronous test bench so the other two modules can be simulated.

   a) Compare the result for the data 1 1 0 1 0 1 0 1 1 0 0 0 0 0

   Ans: Constraint length 3 encoded data-

   11,01,00,10,11,11,10,00,01,11,11,10,00,00,00,00.

   For the constraint length 3 system have the test bench automatically compare your answer with the desired result.

   Ans: Constraint length 4 encoded data-

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

3.0 Convolution Decoder

   The next part of the project will be to design a convolution decoder to retrieve the information bits from the transmitted bits. It should succeed even in the presence of some errors in the transmitted bits. The method we will use is called a Viterbi decoder.

3.1 Decoding Using the Trellis Diagram

   Consider a decoder that receives the transmitted signal 11 01 01 00 10 11 going from t=0 to t=6. Assume the trellis was reset to state S_0 (00) at the start. One goes through the trellis as before, only for decoding the numbers are output/input. So the first input, 11, gives a decoded output of 1 and takes the machine to state S_1.

   At t=1 in state S_1, the next input 01 causes a 1 output and a change to state S_3.
3.1.1 The Hamming Distance (Metric)

This distance is used to show how far apart two binary numbers are. Compare the bits in the same positions in the two numbers. The number of positions that are different is the Hamming distance (h).

Thus 11 and 01 are distance 1 apart (h=1), 1001001 and 1001010 are distance 2 apart (h=2).

Applying the Hamming Distance to Decoding

Suppose the first four received bits have an error so instead of 11 01, one receives 11 11. On the trellis in Fig. 8, there are two choices leaving state S₀, one for input 11/1 and the other for input 00/0. The number in the box is the Hamming distance between the received input and the bits for the transition. It is clear one should make the transition from S₀ ⇒ S₁.

The next input has an error. Note there are no 11 or 00 paths leaving state S₁. Both possible paths, 10 or 01, are at Hamming distance 1. At this time either transition looks equally likely, but wait!
At $t=2$, if one starts from $S_3$, then $h=0$ for the path to state $S_2$. However if one starts from $S_2$ one has $h=1$ for either the path to $S_0$ or to $S_1$. Thus at $t=1$ the proper path was not obvious, at $t=2$, the choice is clearer. We will chose a path through the trellis based on the \textit{path Hamming distance} or \textit{path metric}, which is the sum of the Hamming distances as one steps along a path through the trellis.

Figure 10 shows how the Hamming distances sum as one goes down various paths through the trellis diagram. At $t=5$, one path has a total distance of 1 from the input data. The others have a distance of 3 or 4. Thus the most likely path is $S_0 \rightarrow S_1 \rightarrow S_3 \rightarrow S_2 \rightarrow S_1 \rightarrow S_2$ with a \textit{path distance} of 1, and the corresponding output data is 11010 (Recall trellised edges represent a receiver output of 0, and \dots edges represent an output of 1).

### 3.1.2 Metrics

A metric is a measure of something. The more general name for what we called the \textit{Hamming distance} is \textit{branch metric}, and for the \textit{path Hamming distance} is \textit{path metric}. One does not have to use the the Hamming distance as a measure. In decoders where the input is an analog signal, the distance between the actual and expected voltage may be measured, and the sum of the squares of the errors might be used for the branch metric.
References:
Bernard Sklar, *Digital Communications Fundamentals and Applications*.
Prof. Ian Marsland, http://www.sce.carleton.ca/courses/94554/
Click on "Convolutional Codes." You will need Ghostview (gv) to read the Postscript file.

4.0 The Viterbi Decoder

The decoding example shown above has to follow four paths through the trellis, and remember them for future decisions. For larger decoders, such as the cell phone ones with constraint lengths (shift-register lengths + 1) of 6 and 9, the number of paths can get quite large (32 and 256).

Viterbi developed an algorithm in 1967, which allows the many paths to be discarded without tracking them to completion. He noticed that if two paths merge to one state, only the one with the smaller path Hamming metric, “H,” need be remembered. The other one can never be part of the most likely path.

This means that with the constraint length 3 (shift-register length 2) system in the previous examples only have to remember 4 paths. In general a constraint length K system will have to remember $2^{K-1}$ paths. In theory, the path length should go for the length of the message in order to get the true maximum likelihood path. However it turns out that path lengths of 4 to 5 times the constraint length can almost always give the best path.

The next few figures show how the decoder picks the best path, even when there are errors.

![Figure 11](image-url)
Going out to t=3
The paths temporarily double to 8
There are two paths to each node.
One has a larger path distance.
The larger “H” path can never be
the most likely path, hence we will
erase it.

FIGURE 12

Here just the path Hamming distances
are shown so it is easy to see which paths
should be erased.

Here the unneeded paths are eliminated.
This is all important information up to
t=3.

FIGURE 13

Entering t=4,
Eight paths are created temporarily
Again at each node, only the
lowest path Hamming-
distance path can be part of the
most likely path so four of the eight
paths will again be eliminated.
The second error has made
four paths, all with equal chances
of being the most-likely path.
(four paths with 2)
But see what happens next.

FIGURE 14
Convolution Codes

Eight new paths are created, keep the four lowest H ones entering each state.
Since two paths have the same H as they enter S1, we have no way of choosing a better path. Pick one randomly; Here we choose the one S0⇒S1.

There are two most likely paths (with H=2). Down from four at t=4

Entering t=5

Note how the correlation has continued to act. We are now down to one most likely path.

Entering t=6

Note how the correlation has continued to act. We are now down to one most likely path.

Entering t=7: The one best path is getting fairly clear.
The most likely path ends at S1 and a close second at S0. The paths to S2 and S3 have H=4.
As illustrated above, the Viterbi decoder can decode a convolution code in the presence of some errors.

If two branches entering a state have equal “H,” then the code is unable to tell if one path is more likely than another. Pick one path at random.

4.1 Exercise 2: Add-Compare-Select Design

The circuit to add H+h, compare H+h on the two paths, and select the smaller path metric, is called the add-compare-select circuit.

Problem Prolog: Using the algorithm

A typical step in the trellis decoder is shown.
The path Hamming metrics H at each trellis step are $H_0$, $H_1$, $H_2$, and $H_3$.
The Hamming metric for each edge are given subscripts matching the input which makes them 0. Thus the edge from $S_0$ to $S_0$, and $S_2$ to $S_1$ both use the symbol $h_{00}$. If the input is 00, $h_{00}$=0, If the input is 10 or 01, $h_{00}$=1, if the input is 11, $h_{00}$=2.

Pseudocode

This is Verilog in which the syntax is not critical. For example begin, end and semicolons may be omitted if the meaning is clear to the reader. In pseudocode the comments are often more important than the code.
1. **Problem**

a) Starting at \( t=8 \) with an input of 00, as in Figure 20, calculate and fill in the values of \( h_{ij} \) and hence the \( H_k \) for \( t=9 \).

b) Write pseudocode to calculate the branch Hamming metrics for each step. Let the two input bits \( \text{In}[1:0] \) be \( y, x \). Let the Hamming metrics associated with the eight trellis edges for this step be \( h_{00}, h_{01}, \) etc.

Calculate these metrics using a case statement:

```plaintext
CASE \{(y, x)\}
  2'b00 : begin \( h_{00}=0 \); ... \( h_{11}=2 \); end
  2'b01 : ...
```

2. **Problem:**

a) Starting at \( t=9 \), using the \( H_k \) from Prob 1, step a) and input 10, calculate and fill in Figure 21.

b) Use Boolean algebra to calculate them as 2-bit binary numbers. i.e \( 2=10, 1=01 \) and \( 0=00 \). Use `reg`/*wire*/[1:0] \( h_{00}, h_{01}, h_{10}, h_{11}; \) Example:

```plaintext
h00[1] = y&x ; h00[0] = y^x;
```

3. **Problem**

a) Start at \( t=10 \), with input 11 and use the \( H_k \) from Prob. 2, step a). Let the new \( H_k \) at \( t=11 \) be written with a prime i.e. \( H_0', H_1', H_2' \) and \( H_3' \). Fill in Figure 22 but put in an expression, as well as a number, for each \( H_k' \). This has already been done for \( H_0' \). Only the better path is written here.

b) Write pseudocode to update the \( H_k \) in going from step \( t=n \) to step \( t=n+1 \). Use `if` statements to calculate the \( H_k' \) to be associated with the four states at \( t=n+1 \). Use \( H_{0nxt} \) instead of \( H_0' \) since Verilog cannot handle primes.

```plaintext
if (\( H_0+h_{00} < H_2+h_{11} \)) begin \( H_{0nxt}= H_0+h_{00}; \) end else ...
```

c) The flip-flop procedure.

Write a procedure to clock the flip-flops and replace the old \( Hs \) with the new ones. Combinational logic in parts 2 and 3 calculated the D inputs for the flip-flops. For example:-

```plaintext
always @\( \text{posedge clk} \)
  \( H_0 <= H_{0nxt}; \) ...
```
Don’t put combinational logic in a flip-flop procedure, and don’t forget a reset.

4. **Problem:** When is the output correct?
   Experience has shown that all backward paths converge into one if one traces them back 4 or 5 times the *constraint length*. Using the paths in Figure 18, you will find that if one traces back far enough it does not matter which path one follows.

   a) Take a copy of Figure 18. Start at t=8; start at each state in turn and colour backwards until you reach t=0 or until you hit previous colouring. At what time (t=?) do the paths all converge?

5. **Problem:** When is the output correct?
   Look at Figure 14 in which the ending time is t=4.

   a) Using data available at t=4 could you say, with confidence, what the original data bit was between t=0 and t=1? Why not?

   b) Take a copy of Figure 17. Start at t=7; start at each state in turn and colour backwards until you reach t=0 or until you hit previous colouring. At what time (t=?) do the paths all converge?

   c) Follow the trellis backwards, and from the information in the trellis find the most probable original data. Write out the message in the correct order with the earliest (t=0) bit on the left.

6. **Problem:** Finding the original data from the state.
   a) The states can be placed in two sets, even and odd.
   If one is in an even state (S₀ or S₂), what was the original data in the previous step?
   If one is in an even state (S₁ or S₃), what was the original data in the previous step?
   Write pseudocode to send out the proper output bit based on the state during the trace back.

   ```
   reg [1:0] state
   if (state is odd) output= ...  // Make this more exact.
   ```

7. **Problem:** Use Figure 23 only.
8. **Problem:** How to backtrack.

Figure 24 is the same as Figure 18 except the numbers are all removed. It still contains enough information to trace back from t=8.

Figure 23 shows a trellis decoder only. It gives no information about the data. Figure 24 shows paths, but when you trace back to the area between t=2 and t=3 you cannot tell from the figure what the data was. However the those who did questions Prob: 6 or 7 can tell you.

Figure 25 is the same as Figure 23 except some little parallelograms have been drawn associated with each state in each time step.

Fill a minimum of information in each parallelogram. This information would allow your lab partner two to backtrace knowing that H2 had the minimum path Hamming metric at t=8. Thus by looking only at Figure 25 and starting at state S2 at time t=8, one should be able to tell what the original bit was between t=2 and t=3. You may establish some conventions like a 1 in the state S2 box means.... However they must be independent of the data.

---

a) If the decoder was in state S2 at t=3, what was the original data (before encoding) between t=2 and t=3? (The obvious answer is right.)

If it was in state S0 at t=3, what was the original data between t=2 and t=3?

If it was in state S1 at t=3, what was the original data between t=2 and t=3?

If it was in state S3 at t=3, what was the original data between t=2 and t=3?
a) Using Figure 25, fill in the boxes at \( t=3 \) if you have not done so already, so that one can determine the original data bits between \( t=2 \) and \( t=3 \), and also between \( t=1 \) and \( t=2 \). You should hand in the filled in Figure 25 and your list of conventions.

b) How many bits per step must be stored to allow for backtracking and extraction of the original data?

9. **Problem:** In communications *latency* is the term for the time difference between the time the input signal was received and the output signal is sent out. *Throughput* is the number of input signals that can be processed per second. The point of this problem is to show it does not matter how long it takes to decode the data as long as you can keep up with the input.

c) If a decoder had to wait until all paths converged before it had confidence it could send out a correct output, what would the latency be in clock cycles?

There are two answers for c):

(i) What latency was needed for the data stream as shown in Figure 18 and 17?

(ii) What was the latency, mentioned earlier in these notes, that experience has shown gives the most likely bit for almost all cases?

d) If the decoder delays the signal by 12 to 15 clock cycles, *latency*, would anyone care assuming:

• The signal was a digitized phone conversation?

• The signal was a www page?

• The signal was a digital TV signal?

e) If the decoder could not take in the next input until it had spent 12 or 15 clock cycles processing the previous data, *throughput*, would this matter?

### 5.0 Extracting The Original Data.

Consider Figure 26. The path with the lowest path Hamming metric \( \bar{H} \), starts at state \( S_2 \) at \( t=8 \) with \( H=2 \). Backing up would take the path to \( S_1 \) at \( t=7 \). The edge is a solid line which seems to say the original data was 0. Unfortunately we can’t be sure of this. Because of the convolution code, this path’s \( H \) of 2 could increase in the next few cycles and another path might get the lowest \( H \).
However if one goes back to $t=2$ and travels ahead in time, only paths that start at $S_3$ or $S_2$ make it all the way to $t=8$. The others die out. Only the path from $S_3$ has $H=2$ at $t=8$, thus we are fairly sure the edge from $S_3$ at $t=2$ to $S_2$ at $t=3$ is on the most likely path and the original data between these two clock edges was 0 (a solid line is 0, a broken line is 1).

This illustrates why we waited six cycles here before sending out the output. At time $t=8$, we can be somewhat confident that the “0” data at $t=2$ is the most likely. In general one would wait twice that time to be very sure.

### 5.1 Trace Back In More Detail

Tracing back is a long process if the full trace is done every data cycle. The back trace can be done only if the clock runs several times faster than the data rate. To trace back 15 cycles to find each output bit, means that the input data rate must be no more than clock/16. One input cycle, followed by fifteen trace-back cycles. It turns out one can increase the data rate up as high as clk/2, but that will come later.

Figure 27 shows the trellis after decoding a 11,10,00,01,01,00,00 input stream.

Figure 28 shows how storing one bit, which shows whether to take the upper or the lower path during backtrace, will allow one to reconstruct the trellis.

Figure 29 shows all the surviving paths. If one traces any path back from $t=7$, one will reach $S_1$ at $t=2$. Since all the back traces converge, one has confidence that the value of the data originally generated between $t_1$ and $t_2$, was one (dashed lines represent a one). This example converged quickly, other examples may take longer.

Also note from Figure 28, or from Problem: 6 or Problem: 7 that odd states have only dashed lines (ones) entering them, and even states have only solid lines (zeros) entering them. This means that at the end of the traceback, the data was “0” if one is in an even state and “1” if one is in an odd state.
Convolution Codes

FIGURE 27

Trellis path for a pre-encoding data stream of 1011011... The surviving paths are shown with heavy lines. The paths that die out have light lines. The results for t=7 were left as an exercise.

FIGURE 28

A data bit is stored at each trellis state to show which path to take during a backwards trace.

FIGURE 29

The survivor paths with the excess straw removed.
5.1.1 The Trellis Butterfly.

For all rate 1/2 trellises, one can find a small picture which describes the trellis completely. The picture looks something like a butterfly.

To make the numbers work, one must call the end of the shift register where the data enters, the least significant bit. Since one is used to having the least significant bit on the right, we will flip the shift register of Figure 2 around without changing the circuit.

\[
\begin{array}{c}
\text{FIGURE 30} \quad \text{Trellis made of butterflies} \\
0 & 0 \\
1 & 1 \\
2 & 2 \\
3 & 3 \\
\end{array}
\]

Then going from J to 2J represents a left shift, and shifting in an x of 0. Going J to 2J+1 represents a left shift including shifting x=1 into the flipflops.

\[
\begin{array}{c}
\text{FIGURE 31} \\
J=0 & J=1 & J=2 & J=3 \\
J+2^{k-2} & J+2^{k-2} & J+2^{k-2} & J+2^{k-2} \\
t=4 & t=5 & t=6 & t=7 \\
\end{array}
\]

For a constraint length k=3, 2^{k-2}=2.

Figure 31 shows how to travel backwards through the trellis using the bits stored during each time step to determine whether to take the upward or downward path. Here we start at state J=0. Knowing that one is in state J allows the two paths to be calculated on the fly.

5.1.2 Timing for the simple decoder

The simplest decoder will have the data input at 1/17th of the clock rate. It will do an add-compare-select on that data and store the “came from” bits in memory 1 out of 16 cycles. The rest of the time it is doing the backtrace. That is it will write 1 bit, and then back trace 15 bits to be sure it has found the correct path. Then it will backtrace 1 more bit which it will use as output, before it processes the next data input.

One will need control signals as shown.

<table>
<thead>
<tr>
<th>Input Symbol</th>
<th>Traceback Data</th>
<th>Clk</th>
<th>WriteMem</th>
<th>ReadMem</th>
<th>OutputData</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This will be slow because the throughput will be 1/17 of the input symbol rate (a symbol here is two-bits).
5.2 Summary of the Design

Figure 32 shows one way to do the Verilog design. The top modules only collect signals and pass them on to the lower level modules. When an ASIC is built the arguments for the top module are the pins of the ASIC. If this circuitry was to all be built on one chip, one would put on a top-top or wrapper module where the shaded box is to define the pins.

The error generator is used for testing. To test the circuit, connect the encoder to the decoder through the error generator, and see if the dataOut equals the dataIn. Under normal (not test) operation, the encoder would need an output lead to the outside world and the decoder would need an input lead from outside.

To keep the data rate high, the complete two-bit symbol comes in serially in one clock cycle. This is immediately converted to two parallel bits, both lasting one clock cycle.

5.3 Second Lab: Start of Convolution Decoder in Verilog

Now we will consider the overall project. You should be able to design each block in the block diagram except the Survivor Memory block which will not be done until Exercise/Lab 3. For the initial circuit, you do not need to do a trace back. Just send out as correct, the data from the state with the lowest H. Of course this will not have any error correction.

You should consider these concepts:

a) Normally the encoder and decoder are widely separated so they cannot run from a common clock. The decoder will have a clock recovery module. This is beyond 97.478 and we will use a common clock for both.

b) Your design will be a rate=1/2, constraint length=3; [111],[101] decoder.

c) To save work you will want to parameterize your design. One can do this automatically in Verilog for some parameters. For others it too much trouble.
• First, design a distinctive comment style like
  /*Para*/ comment on parameters /*Para*/
  which indicate code where there are parameters nearby which will need attention. In your
  comments cross-reference all the modules that are effected by the parameters.
• Second, there are two ways to pass parameters in Verilog. Which one works for synthesis?
d) Considerable emphasis will be given to testing. One simple test is a loopback test where the
  output is sent back to the input and the two compared. Other thing you can do to your circuit
to aid testing will be considered later.
e) The Error Generator block is necessary if you want to simulate to the error correction prop-
teities. First it will be done as a test bench so it is only useful during simulation. Later you
might consider making it part of the loopback test so it can be used for testing in the field.

What to do for the lab
1. Draw a block diagram somewhat like that of Figure 32. However make it bigger and show
  the arguments passed to all modules. If a module will be over a page of code, try to divide it.
2. Write Verilog for the Decoder Top Module and Encoder Top Module (already done?)
3. Design a serial to parallel module. The serial_in signal changes at twice the clock rate. Let
  the serial_in bits be labelled  s, e, r, i, a, l ...
  Decide how these bits will come out of a transparent-high latch. See the latchSig below.
You will probably use a transparent latch and two flip-flops in the module.

Write the Verilog code for the serial to parallel module.
How do you write a latch in Verilog? Hints, see Figure 37.
4. Write the Add-Compare-Select module. (See Exercise 2.)
5. Modify the Test Bench to handle the decoder and the encoder. This should include a loop-
back test which compares the dataIn (x on the right) with dataOut.
You will have a delay (latency) between dataIn and dataOut. At the start dataOut will be the bit
  corresponding to the present lowest H (path Hamming metric) so the latency will be only
  that of the serial-to-parallel converter. Later you will want to add a latency of 4 to 5 times the
  constraint length.

1. We like originality in block diagrams as long as you can give a reason for changes.
6. Add to Add_Compare_Select module to generate a four 1-bit signal called came_from. These signals indicate whether the trellis lines leading back from the next states to the present-time-step states, came from a higher state or a lower state. Thus it would show whether the next state S0 came from the present S0 (up) or S2 (down). Figure 35 shows these bits and their meaning. These four came_from signals will be sent to the survivor-path memory-module which will be written later.

7. Consider the Error Generator. For this question treat it like a test bench so you can use nonsynthesizable constructs like $random. This gives a new random integer every time it is called. Randy will be this random integer truncated to 5 bits. Try to make it so there is an error every 16 time steps (every 32 serial_in_bits) on average. A completely random 5-bit number will, on average, be 01110 (or any single value) one time out of 32. However once in a while it might be 01110 twice in a row.

8. Write a pseudorandom generator as was used in 97.350 to replace $random in question 7. There is a lot about pseudorandom generators in the notes. If you use such a generator, you must make its period much longer than 31 or your errors will be periodic. Supposedly random errors that come every 31 bits are not a good test. There was a question about this circuit on the Winter 2001 final which is available on the web.

9. Take any module except the test bench and run it through the Synopsys Design Analyser. Try to get a copy of the synthesized circuit for your report. In 2003 we will do this later.

5.4 Some Hints

Latches have something happen on both edges. Do not use posedge clock. Transparent, latches must follow the data. You need more than @(clock).

Put the reset for the latch in the procedure for the latch, not with some other latch or flip-flop. Unfortunately Synopsys will not synthesize a proper reset on a latch without inserting a

```verilog
//Synopsys directive comment in the latch code.
//Synopsys async_set_reset “rst”
//ambit synthesis set-reset asynchronous signal= “rst”
always @( clk or rst or x)
  ...
```

Latches cannot be put in the same procedure as flip-flops. Do not put logic in the same procedure as the latch.