ECE 111 Term Project - Viterbi Decoder

Contents of Zip File:

- **Folder Part 1: Contains all files needed for top module synthesis of FIRST part only. File viterbi_tx_rx_part1.sv is the top module, with no error injections. Encoder1.sv actually has the code of encoder2.sv, just named it differently.**

- **Folder Part 2: Contains separate files for each bit inversion test. For example, if the file is called viterbi_tx_rx_2b_2.sv, it corresponds to test number 2b- 2.**

- **ECE 111 Term Project PDF: (You are here) Simulation Transcript, RTL Viewer, Resource Utilization and Usage reports, encoder/decoder writeup, bit inversion test table, test observations/comments.**

Part 1:

(**NOTE:** Synthesis was done with top module (encoder and decoder), so all resource summaries/RTL Viewer contain information about entire module. For each section, I will mention what is specific to the decoder for this part as well)

Transcript (NO ERRORS):

```
# error_counter,err_inj = 00000000000000000000000011111100 00          0         252
# error_counter,err_inj = 00000000000000000000000011111101 00          0         253
# error_counter,err_inj = 00000000000000000000000011111110 00          0         254
# error_counter,err_inj = 00000000000000000000000011111111 00          0         255
```

```
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
```

```
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
```

```
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
```

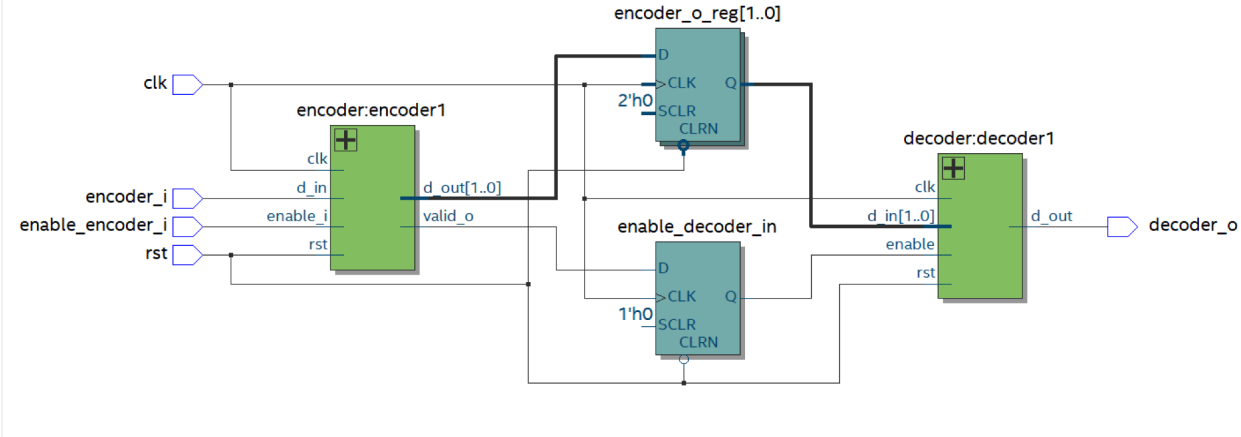
```
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 0, out = 0
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
```

```
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
```

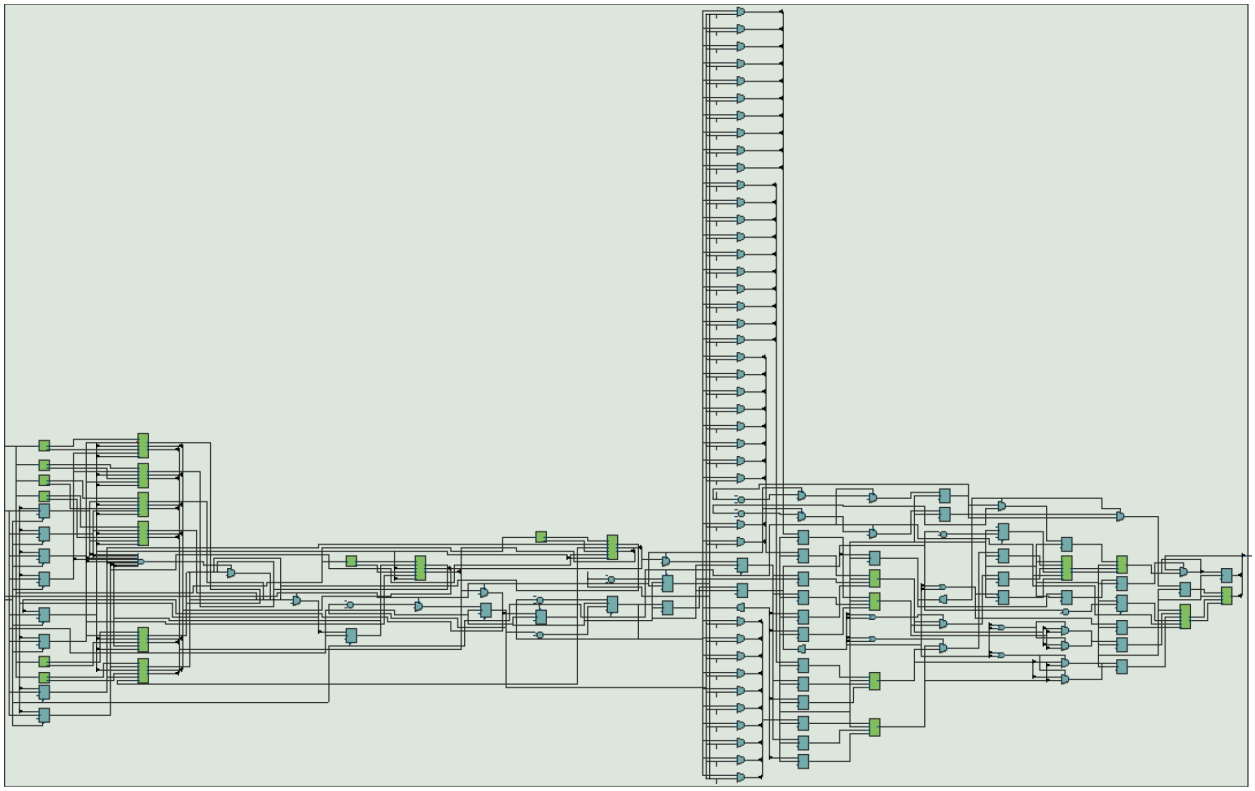
```
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# yaa! in = 1, out = 1
# good =         256, bad =           0
# ** Note: $stop    : L:/ECE111/ECE111_Term_Project/viterbi_tx_rx_tb.sv(371)
#    Time: 1381700 ps  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at L:/ECE111/ECE111_Term_Project/viterbi_tx_rx_tb.sv line 371

VSIM 123>
```

RTL Viewer(s): Each of the modules contains within the top module will be expanded below

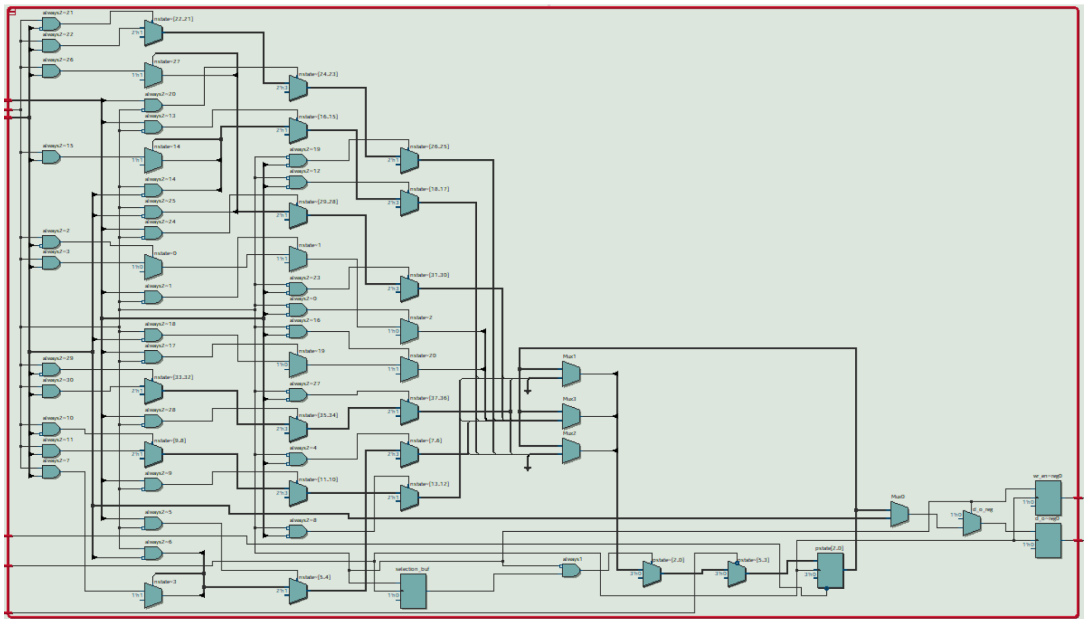
1. **Top Module**



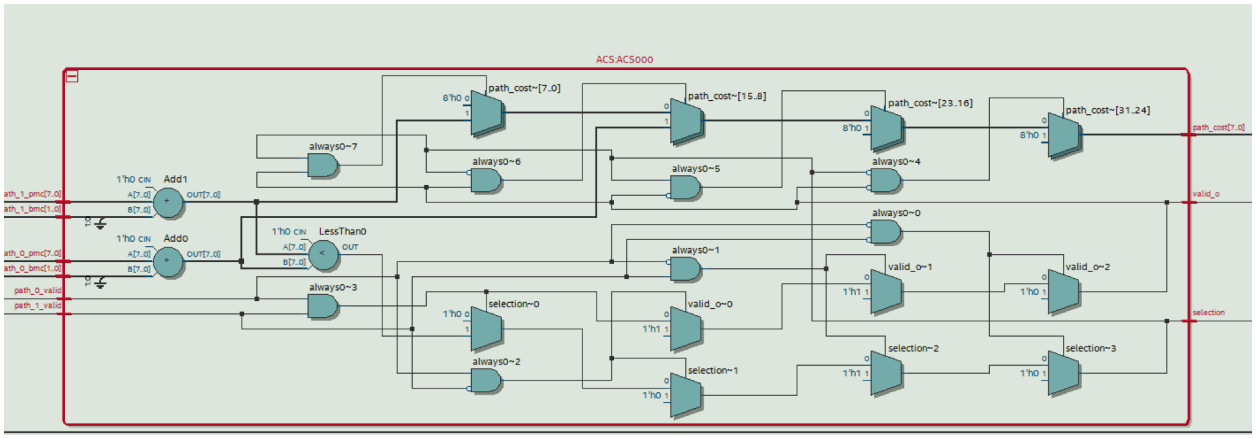
2. Decoder



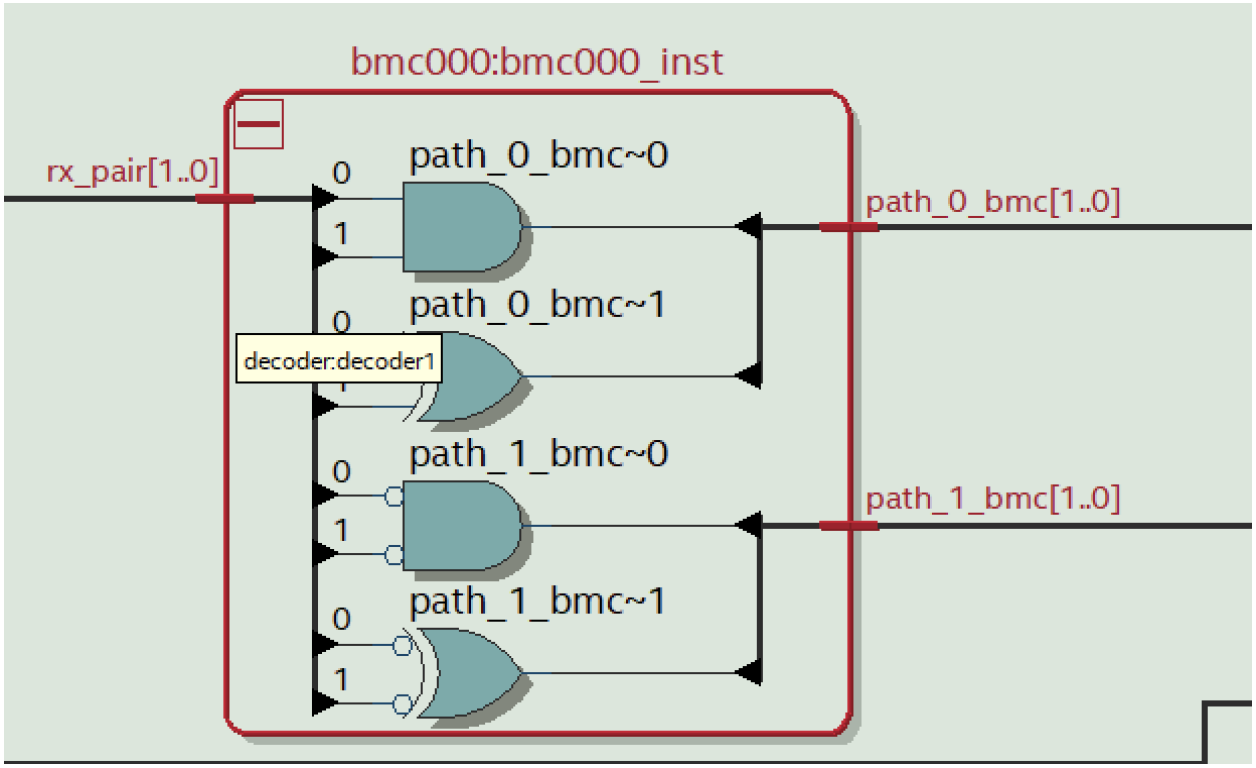
2a. TBU1



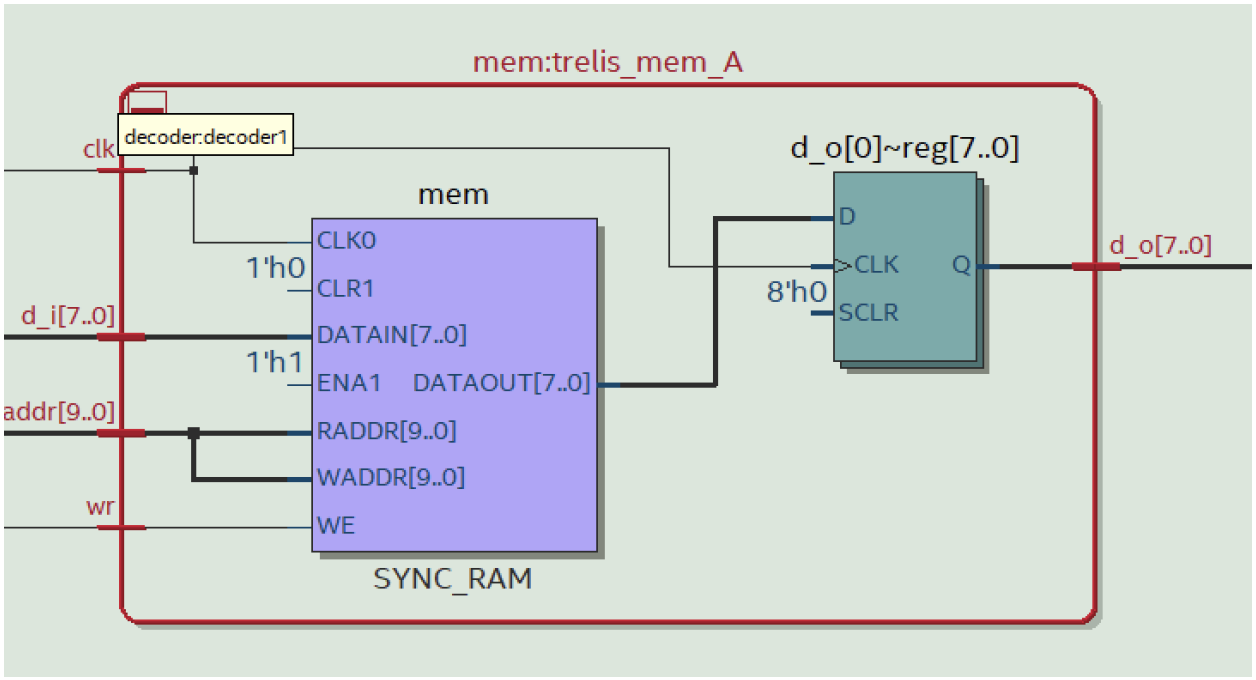
2b. ACS000



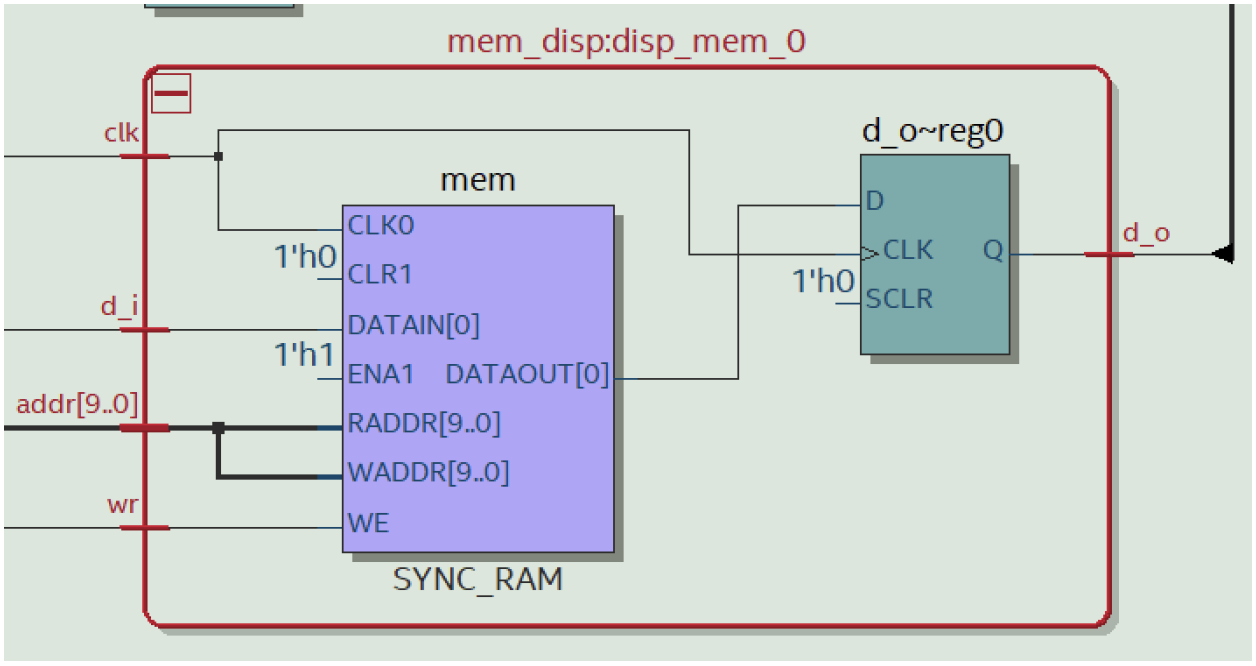
2c. BMC000



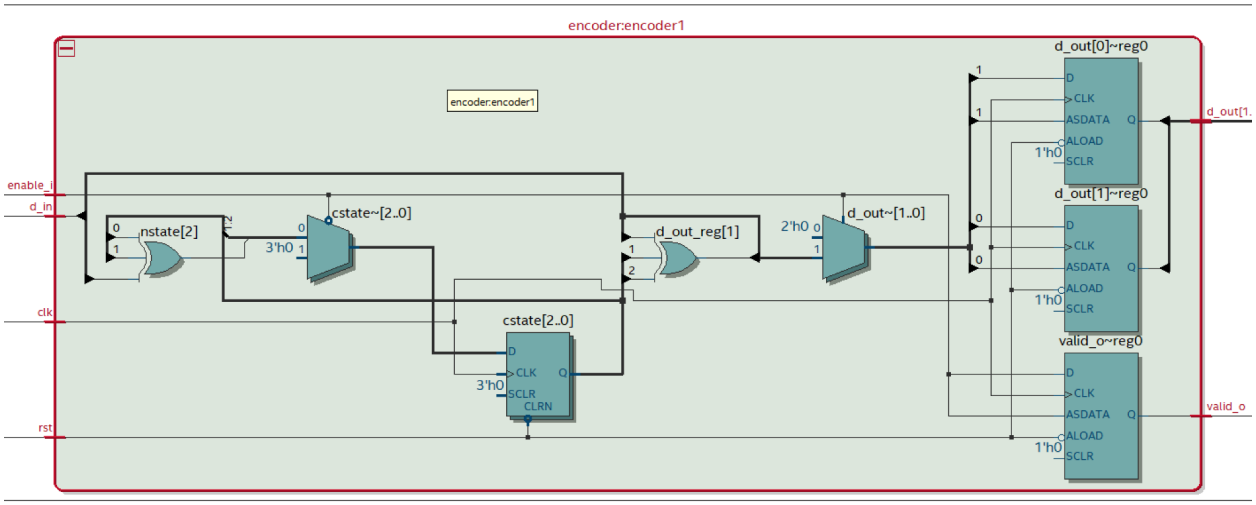
2d. Trellis Mem A



2e. Mem Disp 0



3. Encoder



Resource Utilization Report

Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Elements	DSP 9x9	DSP 12x12	DSP 18x18	DSP 36x36	Pins
viterbi_tx_rx	375 (0)	248 (3)	34816	0	0	0	0	0	5
|decoder:decoder1|	361 (130)	239 (231)	34816	0	0	0	0	0	0
|ACS:ACS000|	24 (24)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS001|	26 (26)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS010|	26 (26)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS011|	25 (25)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS100|	25 (25)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS101|	25 (25)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS110|	25 (25)	0 (0)	0	0	0	0	0	0	0
|ACS:ACS111|	25 (25)	0 (0)	0	0	0	0	0	0	0
|mem:trelis_mem_A|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram_epe1:auto_generated|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|mem:trelis_mem_B|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram_epe1:auto_generated|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|mem:trelis_mem_C|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram_epe1:auto_generated|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|mem:trelis_mem_D|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|altsyncram_epe1:auto_generated|	0 (0)	0 (0)	8192	0	0	0	0	0	0
|mem_disp:disp_mem_0|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|altsyncram_7pe1:auto_generated|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|mem_disp:disp_mem_1|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|altsyncram:mem_rtl_0|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|altsyncram_7pe1:auto_generated|	0 (0)	0 (0)	1024	0	0	0	0	0	0
|tbu:tbu_0|	15 (15)	4 (4)	0	0	0	0	0	0	0
|tbu:tbu_1|	15 (15)	4 (4)	0	0	0	0	0	0	0
|encoder:encoder1|	14 (14)	6 (6)	0	0	0	0	0	0	0

*Decoder: The decoder itself uses 361 ALUTs and 239 dedicated logic registers. The remaining ALUTs and logic registers are for the encoder as well as the registers that connect the enable pin and data between encoder and decoder.

Resource Usage (For top module)

	Resource	Usage
1	▼ Estimated ALUTs Used	375
1	-- Combinational ALUTs	375
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	248
3		
4	▼ Estimated ALUTs Unavailable	60
1	-- Due to unpartnered combinational logic	60
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	375
7	▼ Combinational ALUT usage by number of inputs	
1	-- 7 input functions	8
2	-- 6 input functions	62
3	-- 5 input functions	14
4	-- 4 input functions	21
5	-- <=3 input functions	270
8		
9	▼ Combinational ALUTs by mode	
1	-- normal mode	199
2	-- extended LUT mode	8
3	-- arithmetic mode	168
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	509
12		
13	▼ Total registers	248
1	-- Dedicated logic registers	248
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	5
17	Total MLAB memory bits	0
18	Total block memory bits	34816
19		
20	DSP block 18-bit elements	0
21		
22	Maximum fan-out node	clk~input
23	Maximum fan-out	282
24	Total fan-out	2638
25	Average fan-out	3.96

Write Up

2. The encoder that we use in this project is systematic, as for every bit we put in the input, we get two bits out, one of which is the original data, and the other is the parity bit, which is recursively generated. This is useful because if the original data was corrupted for any reason, since you have an organized output with both a data and a parity bit, it makes decoding the data simpler. The encoder that we use in this project is different from the one we used in lab 7 in that it is a recursive, systematic encoder, as compared to a non recursive, non systematic encoder from lab 7. In lab 7, our encoder is non-systematic because our original incoming data stream does not appear in the output. The output is actually a combination of 2 non-recursive data streams, which only feed the data forward. Unlike the project encoder, this output contains neither pure parity nor data bits.

3. Our decoder has three main units of hardware to make it work: the branch metric generators (BMCs), the Add-Compare-Select unit (ACSs), and the trace back unit (TBU). The goal is to find the minimum weight path or the shortest path through the trellis. We would do this by calculating the hamming separation between the ideal output and the actual received signal. Whichever path has the lowest hamming separation, that path is the one that is selected. The decoding process begins at the ACS module (which also uses the BMCs to actually calculate the 2 branch costs for a given state). The ACS module first adds the previous state metric to the current branch metric, for both possible paths. Then it compares the two paths, and chooses the path with lower cost. Finally it will prune the path with the higher cost, and selecting the lower cost path. This process occurs 8 times in parallel per cycle, due to our 8 states. Once we reach the end, we'll have 8 of the shortest paths for each state. From this point, we traceback, comparing the total mismatches, and for each "column" (cycle), we pick the least accumulated error metric. When tracing back, this would also explain that when there are no differences between the ideal output and the received output, the minimum branch metric is 0, because the hamming separation is also 0. In this way, we are able to recognize which state we need to traceback to for every cycle, and can decode the most likely input sequence to the encoder.

Part 2: TESTING

Test #	Test Name	Injected Errors	Error Pattern	Errors Uncorrected
2a_i	1/16 rate	15	~encoder[1] - Invert MSB	0
2a_ii	1/16 rate	15	~encoder[0] - Invert LSB	5
2a - 1	⅛ rate, bit[0] inversion	31	~encoder[0] - Invert LSB	3
2a - 2	⅛ rate, bit[1] inversion	31	~encoder[1] - Invert MSB	0
2a - 3	1/16 rate, both bit inversion	30	{~encoder[1], ~encoder[0]}	7
2a - 4	1/16 rate, bit[0] inversion 2x row	30	~encoder[0] - Invert LSB, 2x row	9
2a - 5	1/16 rate, bit[1] inversion 2x row	30	~encoder[1] - Invert MSB, 2x row	4
2a - 6	1/32 rate, bit[0] inversion 4x row	31	~encoder[0] - Invert LSB, 4x row	38
2a - 7	1/32 rate, bit[1] inversion 4x row	31	~encoder[1] - Invert MSB, 4x row	16
2a - 8	1/32 rate, both bit inversion, 2x row	30	{~encoder[1], ~encoder[0]}, 2x row	23
2b_i	Random, 1/16 prob,	15	~encoder[1] - Invert MSB	0
2b_ii	Random, 1/16 prob,	15	~encoder[0] - Invert LSB	0
2b - 1	Random, ⅛ prob, bit[0] inversion	26	~encoder[0] - Invert LSB	5
2b - 2	Random, ⅛ prob, bit[1] inversion	26	~encoder[1] - Invert MSB	3
2b- 3	Random, 1/16 prob, both bit inversion	30	{~encoder[1], ~encoder[0]}	16

2b - 4	**Random, 1/16 prob, bit[0] inversion 2x row**	**30**	**~encoder[0] - Invert LSB, 2x row**	**22**
2b - 5	**Random, 1/16 prob, bit[1] inversion 2x row**	**30**	**~encoder[1] - Invert MSB, 2x row**	**15**
2b - 6	**Random 1/32 prob, bit[0] inversion 4x row**	**32**	**~encoder[0] - Invert LSB, 4x row**	**32**
2b - 7	**Random 1/32 prob, bit[1] inversion 4x row**	**32**	**~encoder[1] - Invert MSB, 4x row**	**11**
2b- 8	**Random, 1/32 prob, both bit inversion, 2x row**	**32**	**{~encoder[1], ~encoder[0]}, 2x row**	**19**
2c- 1	**Longest bit[0] errors to cause output fault in EARLIER trials**	**3**	**Bit[0] errors injected at trials 166 to 168**	**0**
2c- 2	**Longest bit[0] errors to cause output fault (longest possible)**	**10**	**Bit[0] errors injected at trials 245 to 255**	**0**
2d- 1	**Longest bit[1] errors to cause output fault in earlier trials**	**4**	**Bit[1] errors injected at trials 151 to 154**	**0**
2d- 2	**Longest bit[1] errors to cause output fault (longest possible)**	**8**	**Bit[1] errors injected at trials 247 to 255**	**0**
2e- 1	**Longest bit[0] and bit[1] errors to cause output fault in earlier trials**	**1**	**Bit[0] and Bit[1] errors injected at trial 155**	**0**
2e- 2	**Longest bit[0]**	**8**	**Bit[0] and Bit[1] errors injected**	**0**

	and bit[1] errors to cause output fault (longest possible)		**at trials 247 to 255**	

Part 2a & 2b Observations:
Although both parts contained the same trials, the difference was that in 2a, the errors were placed at a specific BER, while 2b contained the same rate of error as a probability, making the errors come in randomly. The commonality between the two was that as the burst of errors increased (from 1 to eventually 4 in a row, of various bit combinations), the errors also increased. However, the difference was in the amount of errors that were uncorrected. The specific BER tests corrected more errors per individual test than the random tests in most cases, except one. When 4 error bits in a row were injected at 1/32 rate, the random test actually did better at correcting the errors than the specific BER test.

Part C-E: If errors are even 1 greater than the injected errors recorded above, output errors occur.
Observations: These tests posed an interesting question. For consecutive bit errors, how does the placement of the consecutive errors in the 256 trials affect the final output? To answer this question, I first tested multiple different combinations of consecutive errors across the 256 trials. Turns out, if we place numerous consecutive errors near the end of the trial sequence (from trial 255 downwards) we can put many more consecutive errors than on average across the earlier trials in the sequence. Therefore, I decided to split the test for each into 2 separate parts, one for consecutive errors in the earlier trials, and one for the longest possible sequence (at the end), giving a more holistic picture of what happens in the presence of consecutive errors injected across the whole sequence of trials.