Term Project Assignment

Part 1 Objective: Complete the decoder Verilog starter files to make it run successfully on the test bench provided on Canvas (viterti_tx_rx_tb.sv). For part 1, disable error injection in the channel (in viterbi_tx_rx1.sv) which connects the encoder output to the decoder input. Complete decoder.sv and its submodules.

Part 1 deliverables:
1) Your working SV code, the Questa/ModelSim transcript, and proof of synthesis (either Quartus RTL viewer diagram or Mentor Precision netlist text, plus utilization report). Transcript will show your score / 256 trials.
2) Explain how this encoder works, since it differs somewhat from the Homework 7 unit, although similar in principle. Use this encoder with the decoder – your Homework 7 encoder will provide decoder input vectors that won't work with this design.
3) Explain how the decoder works and show that the minimum branch metric = 0 when no errors are present. How does the decoder determine what sequence of bits was sent to the encoder?

For item 1), use the comments in the partial code and the notes below. I shall cover this in class and discussion during Week 9, so be sure to attend or at least watch the recordings.

Add-Compare-Select

path cost(i) = path metric(i) + branch metric(i)

valid_o	selection	path cost
0	0	0
0	1	0
1	0	path_cost_0
1	1	path_cost_1

valid_o = 0 if neither path valid

path_o_valid	path_1_valid	selection
0	0	0
0	1	1
1	0	0
1	1	*

*: 1 if path_cost_0 larger than path_cost_1; else 0

Branch Metric Computation Blocks

There 8 modules, identical except as noted here:

tmp00 = rx_pair[0]; tmp01 = rx_pair[1]
  exception: for bmc 1,2,5, and 6, invert rx_pair[1] in the above expression

tmp10 = !tmp00   tmp 11 = !tmp01

path_0_bmc[1] = tmp00 & tmp01
      path_0_bmc[0] = tmp00 ^ tmp01
  same for path_1_bmc with tmp10 and tmp11

Encoder

Note: differs a bit from Homework 7

cstate	d_in = 0		d_in = 1	
	nstate	d_out_reg	nstate	d_out_reg
0	0	00	4	11
1	4	00	0	11
2	5	10	1	01
3	1	10	5	01
4	2	10	6	01
5	6	10	2	01
6	7	00	3	11
7	3	00	7	11

Trace Back Unit

wr_en_reg = selection
d_o_reg = if (selection) d_in_1[pstate] else d_o_reg = 0

State Transition Table: nstate[i] (Next State) = f(pstate[i], selection, d_in_0[i], d_in_1[i]

pstate	selection=0	selection=0	selection=1	selection=1
	d_in_0[pstate]=0	d_in_0[pstate]=1	d_in_1[pstate]=0	d_in_1[pstate]=1
0	0	1	0	1
1	3	2	3	2
2	4	5	4	5
3	7	6	7	6
4	1	0	1	0
5	2	3	2	3
6	5	4	5	4
7	6	7	6	7

Part 2 Objective: Now that you built it, try to break it. Inject bit inversions (errors) between the encoder and the decoder in viterbi_tx_rx*.sv. Make a new version for each test.

Test the robustness of your design. **Create a table of number of errors injected, patterns of errors injected, and number of errors uncorrected.**

2.a) Try various channel bit error patterns of rate of 1/16:

2.a.1) Invert bit[0] of the channel once every 8 samples.

2.a.2) Repeat for bit[1].

2.a.3) Repeat with bit 0 and 1 both inverted once every 16 samples.

2.a.4) Invert bit[0] twice in a row out of every 16 samples: 14 good, 2 bad, 14 good, 2 bad, … .

2.a.5) Repeat for bit[1]

2.a.6) Invert bit[0] four times in a row out of every 32 samples.

2.a.7) Repeat for bit[1]

2.a.8) Invert bits 1 and 0 twice in a row out of every 32 samples.

2.b) Using the **$random** function, devise randomized versions of the above tests.

2.c) How long a string of consecutive bad bit[0]s is needed to produce output errors?

2.d) Repeat for bad bit[1].

2.e) Repeat for bit [1] and [0] both.