Encoder Module

Description:

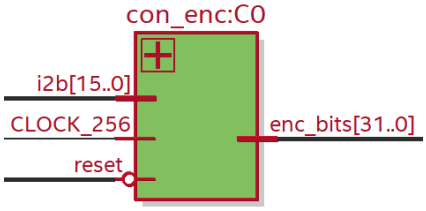

Figure 1: Convolutional Encoder

The convolutional encoder block is configured with a constraint length of 7 and standard polynomials [171, 133] in octal notation. It takes a 16-bit input from the ADC subsystem and processes each bit and outputs 2 bits. Therefore, it operates on a clock that is 16 times faster than that the clock that the down-sampler operates on because when new input will arrive from i2b then it would have processed each bit for 16 times and send 32-bit encoded output in serial. A more detailed RTL block diagram is shown below.

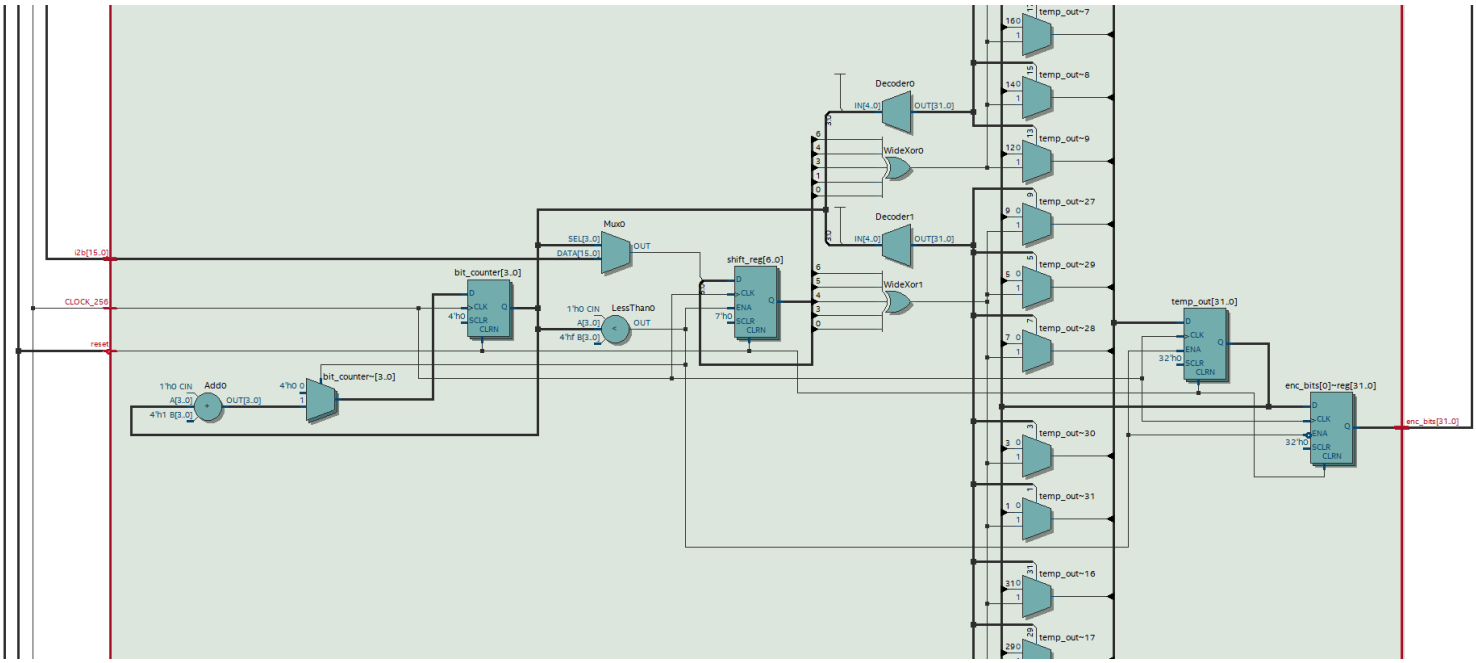

Figure 2: RTL diagram of the convolutional encoder

The encoding involves using a shift register to hold the current input bit. The output bits 1 and 2 are calculated by doing bit wise AND with the generator polynomials and then XOR reduction to 1 bit. The output bits are stored a temp_out register and a counter is used to keep adding 2 bits to the temp_out register every clock cycle. After 16 clock cycles, the 32 bit enc_bits is given the value of temp_out and is the output from the module.

Testing strategy:

The strategy was to calculate the expected value assuming a 16 bit input for a few tests and then use $random function in the testbench to test lots of different 16 bit inputs. Below is an example of the method used to calculate the expected output value.

Expected value calculation:

If i2b (input to our module) = 16'b0000111100001111

Parameters: K = 7, G1 = 7'b1011011 (171 in Octal), G2 = 7'b1111001 (133 in Octal)

Initial state:

- shift_reg = 7'b0000000
- temp_out = 32'b00000000000000000000000000000000

Iteration 1:

- shift_reg = {6'b000000, 1'b1} = 7'b0000001
- Output bit 1 (temp_out[0]) = ^(shift_reg & G1) = ^(7'b0000001 & 7'b1011011) = ^(7'b0000001) = 1
- Output bit 2 (temp_out[1]) = ^(shift_reg & G2) = ^(7'b0000001 & 7'b1111001) = ^(7'b0000001) = 1

Iteration 2:

- shift_reg = {5'b00000, 1'b1, 1'b1} = 7'b0000011
- Output bit 1 (temp_out[2]) = ^(shift_reg & G1) = ^(7'b0000011 & 7'b1011011) = ^(7'b0000011) = 0
- Output bit 2 (temp_out[3]) = ^(shift_reg & G2) = ^(7'b0000011 & 7'b1111001) = ^(7'b0000011) = 0

Summary of output bits for each iteration:

Iterations	Temp_out	value
1	temp_out[1:0]	2'b11
2	temp_out[3:2]	2'b00
3	temp_out[5:4]	2'b11
4	temp_out[7:6]	2'b10
5	temp_out[9:8]	2'b00
6	temp_out[11:10]	2'b00
7	temp_out[13:12]	2'b00
8	temp_out[15:14]	2'b00
9	temp_out[17:16]	2'b11
10	temp_out[19:18]	2'b00
11	temp_out[21:20]	2'b11
12	temp_out[23:22]	2'b10
13	temp_out[25:24]	2'b00
14	temp_out[27:26]	2'b00
15	temp_out[29:28]	2'b00
16	temp_out[31:30]	2'b00

Final output enc_bits = 32'b10101100101011001010110010101100

The encoded bits are sent as output as serial at the end of 16 clock cycles.

Waveform:

The simulation shows the encoding process. We can see the counter incrementing and updating temp_out when reset was turned low. After 16 cycles enc_bit is updated and we can see correct output value what we expected for the given input.

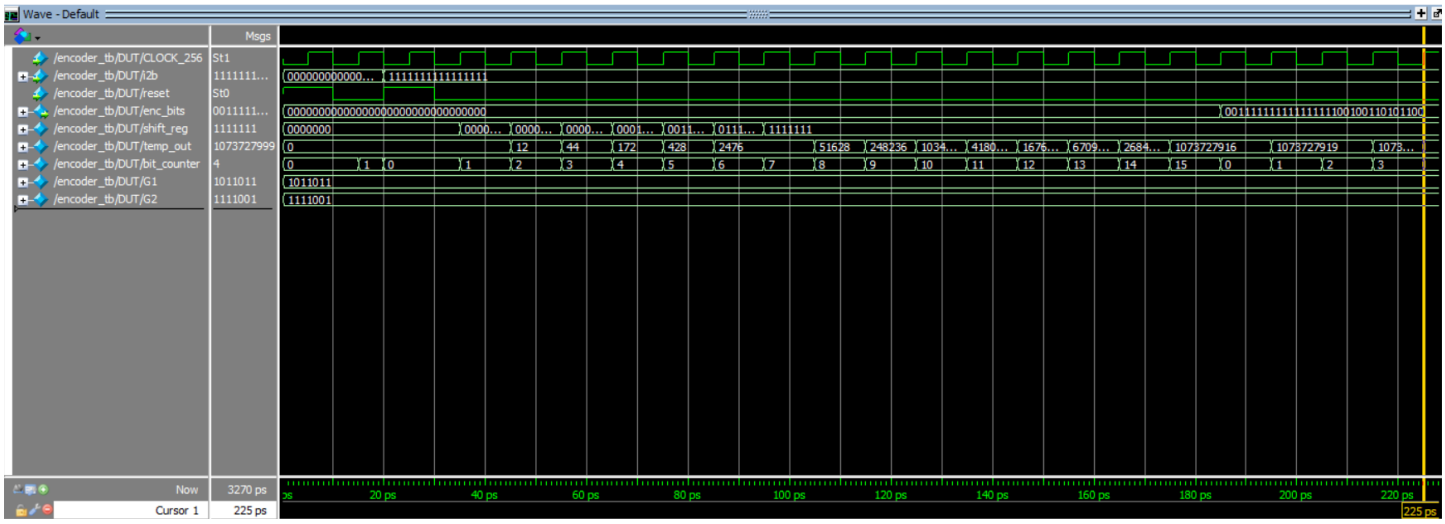

Figure 3: Modelsim waveform for convolutional encoder