1. The goal of this problem is to design a circuit that converts from a gray-code to a normal binary-code.

A gray-code is a binary encoding where each symbol (word) in a sequence differs from the previous symbol by exactly one bit position. For instance, a three-bit gray-code might have the sequence: 000, 001, 011, 010, 110, 111, 101, 100, 000, ..., whereas a normal 3-bit binary code has the sequence: 000, 001, 010, 011, 100, 101, 110, 111, 000, ...

One way to design this circuit is to wire the output of a gray-code decoder to a binary-encoder. For this exercise, we will use three bits encodings.
   a. Following this idea, show the design of the two blocks at the logic gate-level and the composite design as a block-diagram.
   b. Write the Verilog code for this design using structural verilog based on your answer above. Your design should have three module definitions.
   c. Write a alternative description for the encoder and decoder modules using continuous assignment statements.
   d. Write a behavioral-level Verilog description with a single module that performs both parts.

2. DDCA 4.2

3. DDCA 4.22

4. DDCA 4.40

5. One significant advantage of FPGAs is the ability to reconfigure the device quickly and reliably. Consider the task of reconfiguring an FPGA: millions of configuration bits are required but the programming interface is limited to a handful of ports. In this exercise, we will explore the use of a “scan chain” as a programming method for reconfigurable logic. A scan chain is essentially a shift register with three inputs:

   scan_in: Accepts the serial bit-stream that programs the device
   scan_clk: Clock signal for the shift register
   scan_out: Output of the last flip-flop in the chain
For example, this illustration shows a four-bit scan chain:

Based on this information:

a. List two reasons why it is important to have `scan_out`.

b. Implement the scan chain in Verilog. To copy the skeleton files, issue the command:

   ```
   curl http://inst.eecs.berkeley.edu/~cs150/fa11/agenda/hw/hw2.tar.gz | tar -xv
   ```

   For this exercise, complete `ScanChain.v`. In this file, you will write the Verilog code to implement a variable length scan chain. The length should be determined by the WIDTH parameter. Use your knowledge of parameters, generate statements and the FDRSE primitive to complete the design.

6. Now, use the scan chain you designed in the previous question in order to program a configurable logic block (CLB) that you will implement here. Your CLB should be able to implement any two functions of 3 variables or any one function of 4 variables (Hint: from where does this sound familiar?).

   a. Implement the configurable logic block in `CLB.v`. You may not change the module definition. Set the default WIDTH parameter to the number of configuration bits required by your design.

   b. Next, observe that the CLB is instantiated in `ScanChain.v`. The configuration input of the CLB is taken from the scan chain. Based on your design, set the proper default WIDTH parameter.

   c. Finally, use the scan chain to program your CLB as a full adder. A full adder takes three 1-bit inputs: A, B, and CarryIn. It then produces the Sum and CarryOut based on these inputs. Fill in the WIDTH and your adder configuration bits in `FPGA_TOP_ML505`.

To verify the functionality of your CLB, the skeleton files connect DIP switches 1-3 and LEDs 0-1 to the CLB. The DIP switches should be the inputs to the adder. LED0 should display the sum bit, and LED1 should display the carry bit. Examine the port connections to ensure that your configuration bits will give the proper output.

You will test your design by using a verilog simulator – instructions on how to do this are provided on the next page.
To simulate your design:

```
# export MGLS_LICENSE_FILE=1717@sunv40z-1.eecs.berkeley.edu:1717@sunv20z-1.eecs.berkeley.edu
# cd hw2/sim
# make
```

This will run HW2Testbench.v and exhaustively test your adder. The output of the make command will show the test results.

The simulations also generate waveforms of the signals in your design that you can view to aid in debugging. Issue the following commands:

```
# cd hw2/sim
# ./viewwave results/HW2Testbench.wlf &
```

This opens Modelsim, which shows waveforms for the signals in your device. On the left, you should see a tree of module names and signals. Right click on the one you would like to view, then select add > to wave > all items in region.

To submit this exercise, run "submit hw2" from your class account. (You may need to run register first.) Please submit only ScanChain.v, CLB.v, FPGA_TOP_ML505.v and sim/results/HW2Testbench.transcript.

Note that if you would like to see how your design operates in real life, we have provided further skeleton files and information below to you to do so. (Running/testing your design on the FPGA is by no means required however.)

To run your design on the FPGA:

```
# cd hw2/
# make
# make impact
```

After programming the FPGA, you should see that the proper sum and carry bits are shown on LEDs 0 and 1, respectively, when you toggle DIP switches 1-3.