The starter code from the course web page includes test benches for each problem below. For each of the following, please submit the source code for your Verilog module along with a printout or screen capture of waveform plots indicating a successful run. Please make sure to set the display radix (base) on your waveform viewer to “binary”.

1. Design a combinational Verilog module

   ```verilog
   module majority(A, Y);
   ```

   that takes a 3-bit input \( A \) and outputs a single bit \( Y \) which is equal to 1 if and only the majority of the input bits are 1’s.

2. Design a combinational Verilog module

   ```verilog
   module sort_bits(X, Y)
   ```

   that takes a 3-bit input \( X \) and computes a 3-bit output \( Y \) which reorders the bits of \( X \) such that all of the zeros come first, then all of the ones. For example, if \( X = 100 \), we expect \( Y = 001 \).

3. Design a combinational Verilog module

   ```verilog
   module binary_to_gray(B, G)
   ```

   that takes an unsigned 4-bit binary number \( B \) as input, and computes the corresponding 4-bit gray code \( G \) as output. One way to convert from binary to gray code is to observe the following rules:
   - The MSB of the output is equal to the MSB of the input.
   - Bit \( i \) of the output is obtained by XOR-ing together bits \( i \) and \( i + 1 \) of the input (i.e. that bit, and the one to its left).

4. Design a combinational Verilog module

   ```verilog
   module mux_4_to_1(I, S, Y)
   ```

   which takes a 4-bit input \( I \), a 2-bit address \( S \), and outputs a single bit \( Y \). The \( Y \) output should always be equal to the bit of \( I \) whose index is given by \( S \). To implement this 4-to-1 MUX, your module must use the `decoder_2_to_4` module provided in the starter code, similar to the example on the board from class.