JPEG Huffman Decoder

Introduction

The main objective is to develop a Huffman decoder which decodes the given uncompressed data stream in a file.

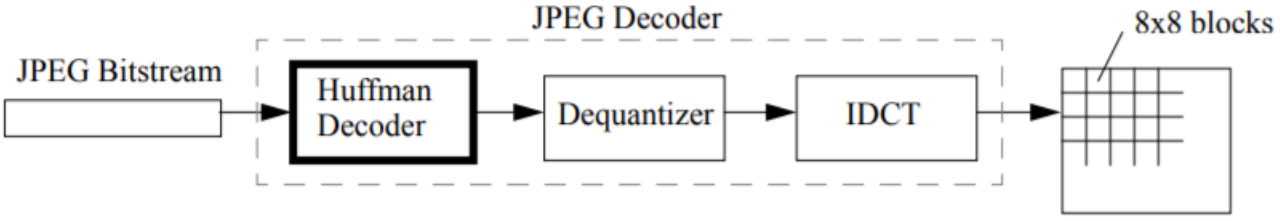


FIGURE 2. JPEG Decoder Flowchart

We focus on Huffman decoder for the project. JPEG-lite simplifies JPEG. The input has been modified to 4x4 blocks as opposed to the 8x8 blocks used in the JPEG standard to reduce the layout effort of hardware elements. (we sacrifice our compression ratio by a factor of 2). The JPEG standard is used in a variety of applications such as downloading graphics from the internet, digital cameras, medical imaging tools. In this project, we are focusing on a small portion of the algorithm known as the Huffman decoder which is used for data compression. Images generally contain two types of coefficients. The high frequency coefficients are located at the edges of objects while the low frequency ones are present almost everywhere else. Since the high-frequency components are usually small, they can be approximated by zero with little error. This is the basis for this project.

The Huffman decoder's top-level verilog, top.v, contains 4 sub-units: stimulus.v, datapath.v, control.v, and writeoutput.v. We need to implement the Huffman Code Identification Algorithm in datapath.v and the finite-state machine in control.v. The interactions between these units are shown in Figure 7.

Stimulus.v provides the input signals to the Huffman decoder. It receives its inputs from three files: image.dat, table1.dat and table2.dat. Table1.dat and table2.dat contain the necessary information to initialize the lookup tables used in the datapath. After initialization, stimulus.v sends bits from image.dat to datapath.v as the JPEG bitstream.

Datapath.v contains the lookup tables, the shift registers and other hardware elements needed to implement the Huffman Code Identification Algorithm. The primary inputs are the JPEG bitstream from stimulus.v, and control signals from control.v that reset and initialize the shift register and code_length counter. The primary outputs of datapath.v are match_s1, run_length_s2, coeff_size_s2, and coefficient_s2. Match_s1=1 indicates that a complete Huffman code has been shifted in from the JPEG bitstream. The results of the table2 lookup: run_length_s2 and coeff_size_s2, provide information to the control about the run_length and size of the coefficient that follow the Huffman code in the JPEG bitstream. Coefficient_s2 contains the bits that are currently in the datapath shift register.

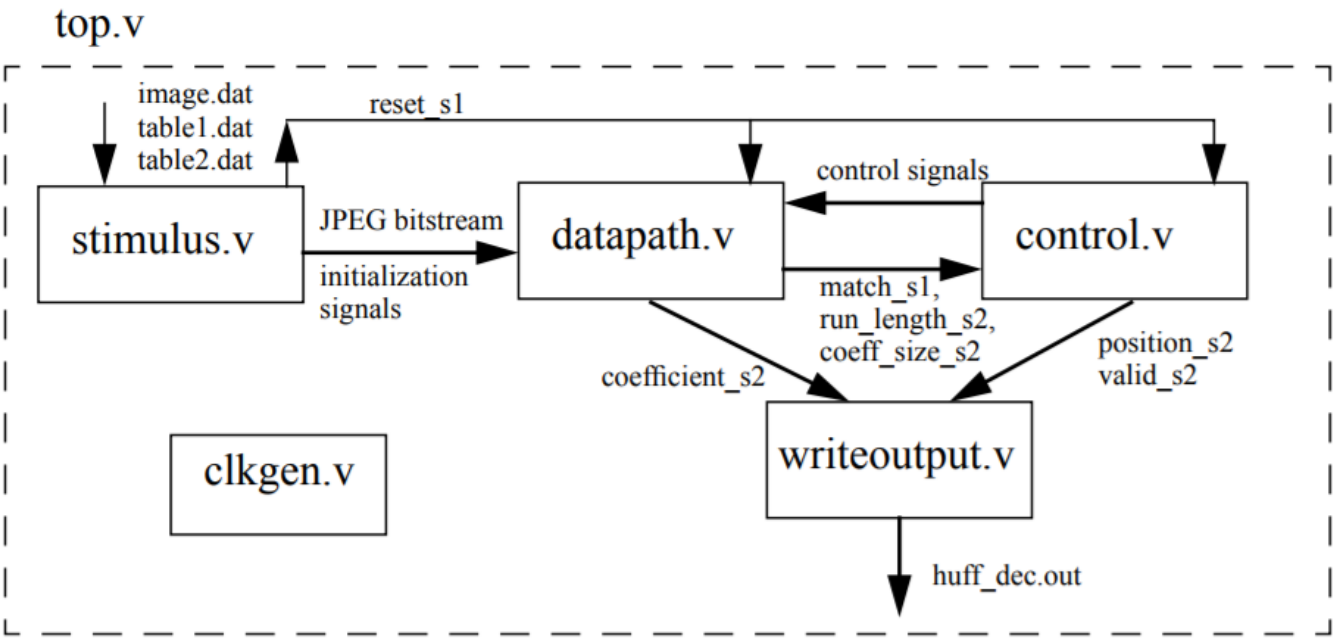


FIGURE 7. Verilog Hierarchy

Control.v contains a finite state machine, logic that counts how many coefficient bits have been shifted into the datapath shift register, and logic that calculates position_s2. The primary inputs are match_s1, run_length_s2, and coeff_size_s2. The control block's primary outputs are position_s2, valid_s2 and control signals needed by the datapath. As described above, position_s2 indicates the location within the 4x4 block and valid_s2 is asserted when the entire coefficient has been shifted in from the bitstream.

Procedure

The design was implemented by following below steps:

Milestones:

1st Milestone: The Finite State Machine provided in the reference pdf was implemented in the control block (control.v) for the project, the control signals that were generated were created. Using a switch case condition, we moved to the next state if certain conditions were met otherwise we remained in the current state itself. The Control block receives the following signals from datapath:

1. match_s1, sets high if a huffman code is properly detected.
2. coeff_size_s2, indicates the size of the coefficient.
3. run_length_s2, indicates number of zeroes preceding a valid value.

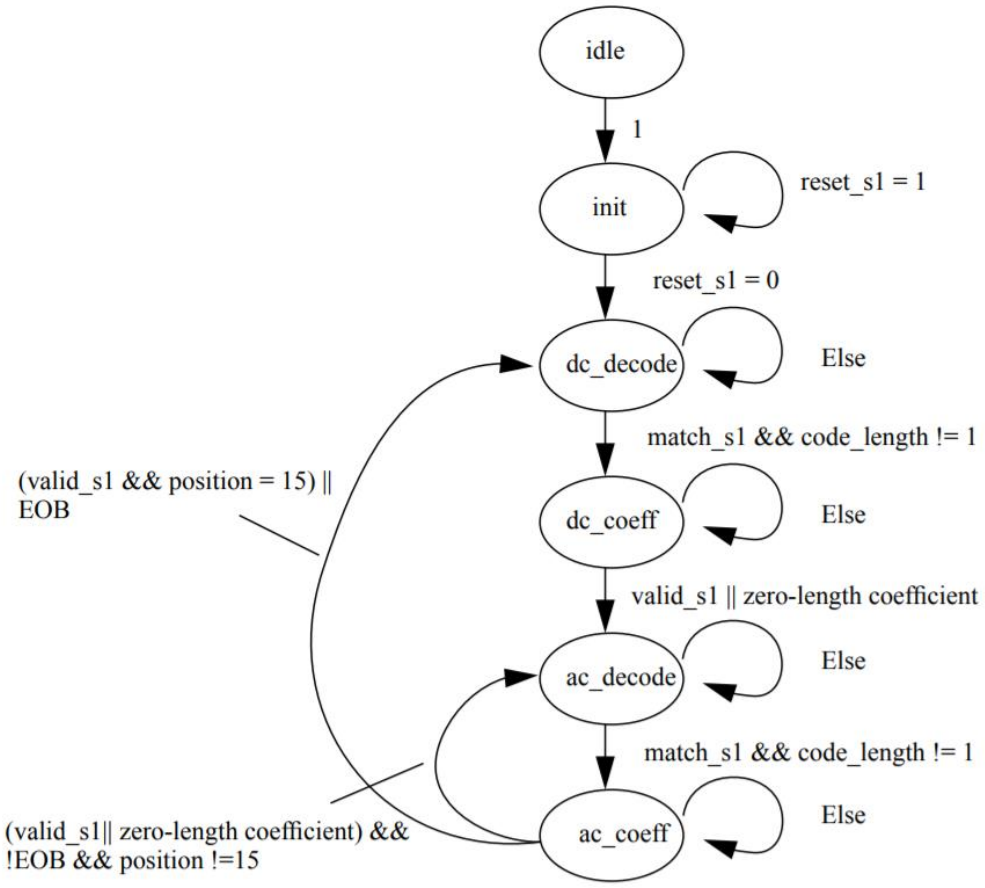


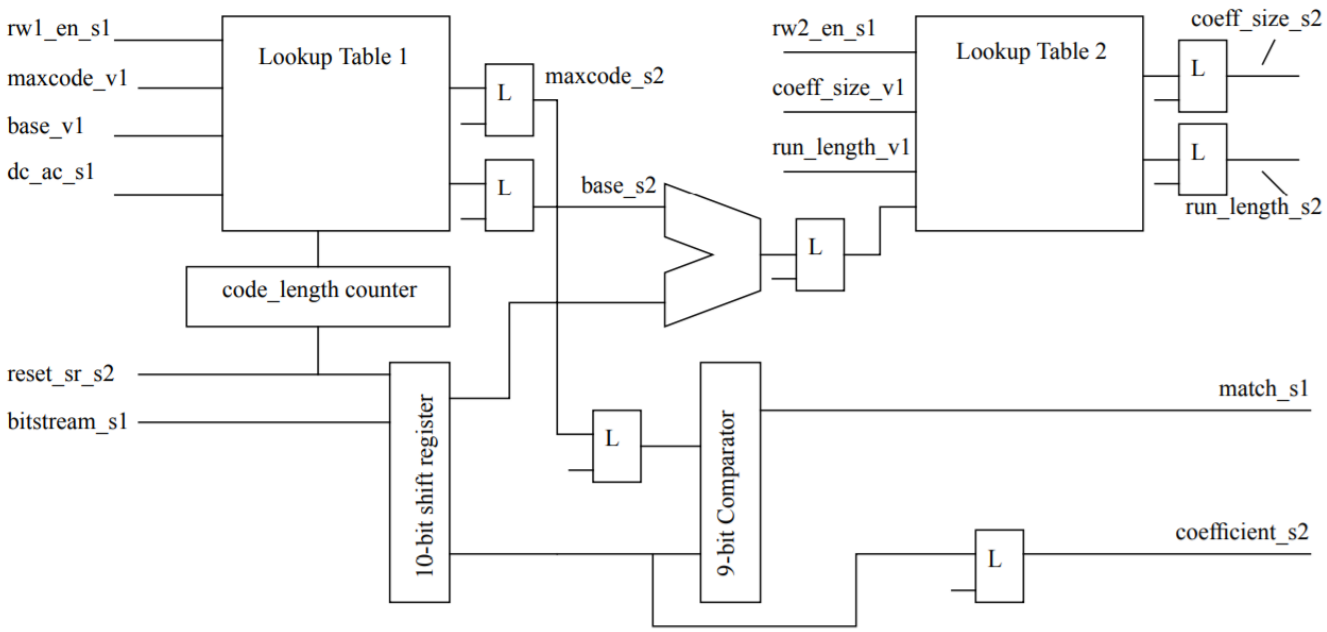
FIGURE 8. Finite State Machine

2nd Milestone: The combinational logic in the datapath.v was implemented using the algorithm provided in the reference pdf.

Latches for Table 1 outputs: Table 1 output is sent to Table 2 and a comparator. So, latches are required to hold the data until the units that are receiving these signals completely loop in the data. It is achieved by assigning base_s1 to base_s2 and maxcode_v1 to maxcode_v2 whenever base_v and maxcode_v1 are modified respectively.

Table 2 address calculation: The base value generated by Table 1 and the number of bits detected from the input bit stream for calculating a huffman code are required to calculate address to read from Table 2. It is achieved by just adding base_s2 with bits_s2.

9-bit Comparator: If bits value is less than or equal to the maxcode then match_s1 should be set to 1 and the Huffman code length must be determined. The match_s1 is sampled by control block but it is only sampled when code length is greater than 1. This is achieved by assigning match_s1 to 0 when bits_s1 is greater than maxcode_s1. If not match_s1 is set to 1.



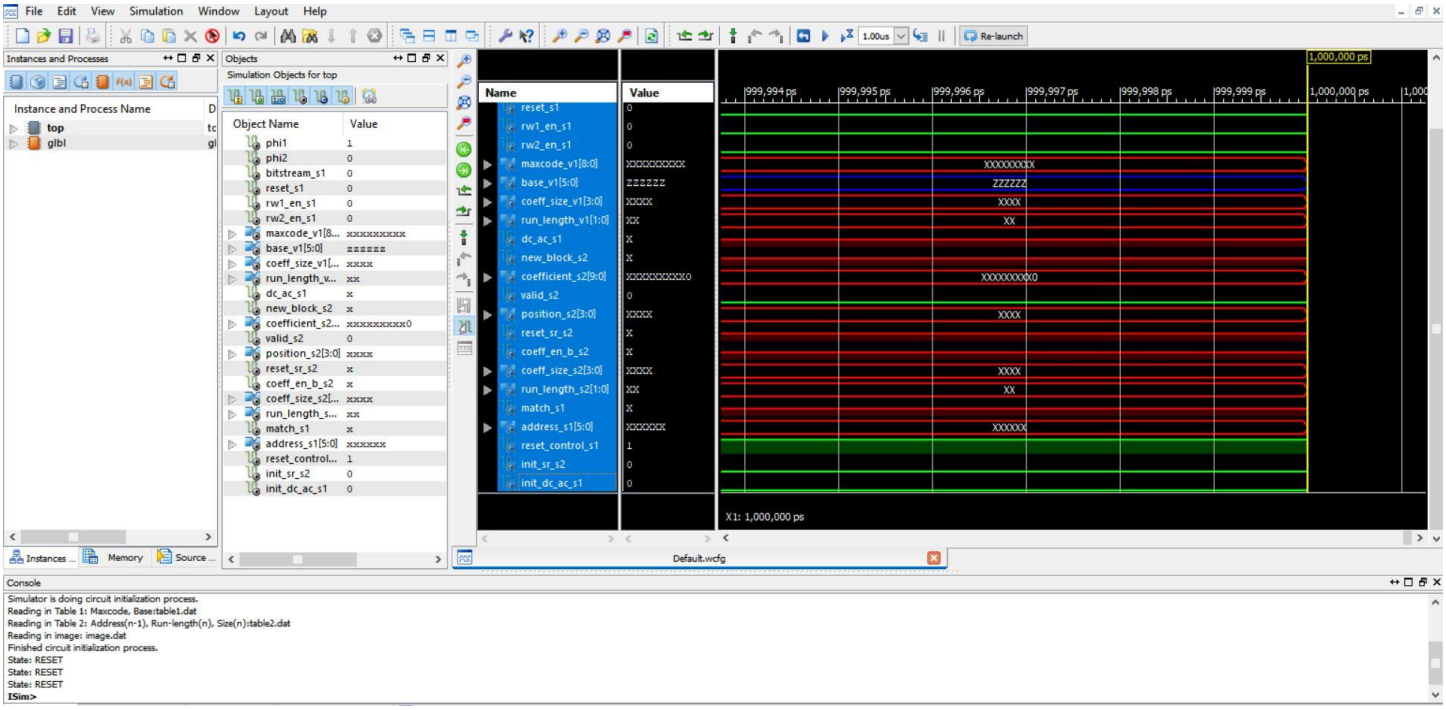
3rd Milestone: Reg.v file is used to calculate the address of Table 1 from which maxcode and base values are retrieved. This is basically a code length counter block. This is achieved using an 8-bit code_length shift register. This functional block generates the word line selects for lookup table 1 and only one-word line will be high at one time.

The most difficult part was to implement the shift register in reg.v file as intermediate states had to be maintained to properly develop the shift register and the ac_coefficient part in control.v file as I had to check the co-efficient size and run_length valid for only one cycle.

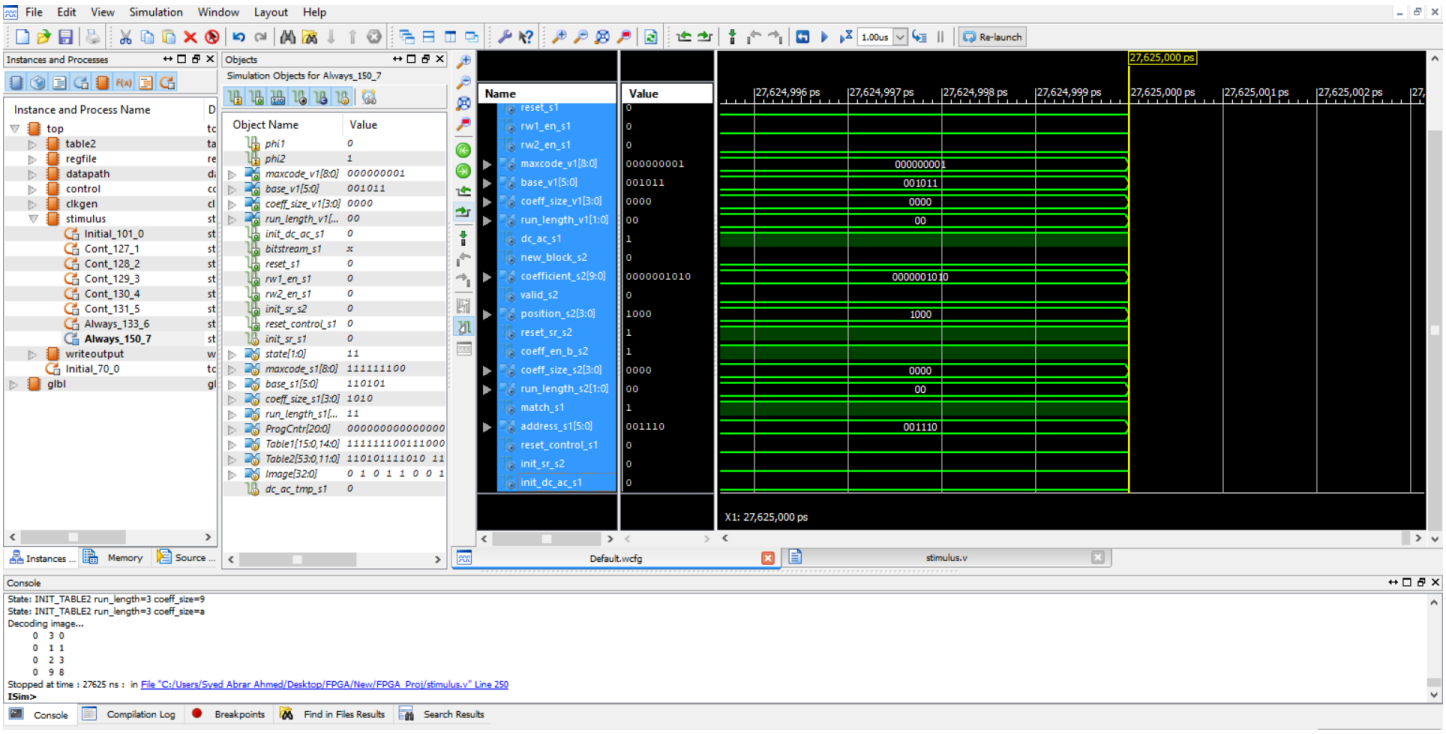
Conclusions and Results

I tested the implementation by verifying the huff_dec.out file. After simulating the code in ISE, I renamed test1.dat and test2.dat to image.dat one at a time as inputs. The decoded output is obtained in huff_dec.out. I compared the test1.out and test2.out with that of huff_dec.out file and found it matching. Below are the waveform outputs for test1.dat and test2.dat input data.

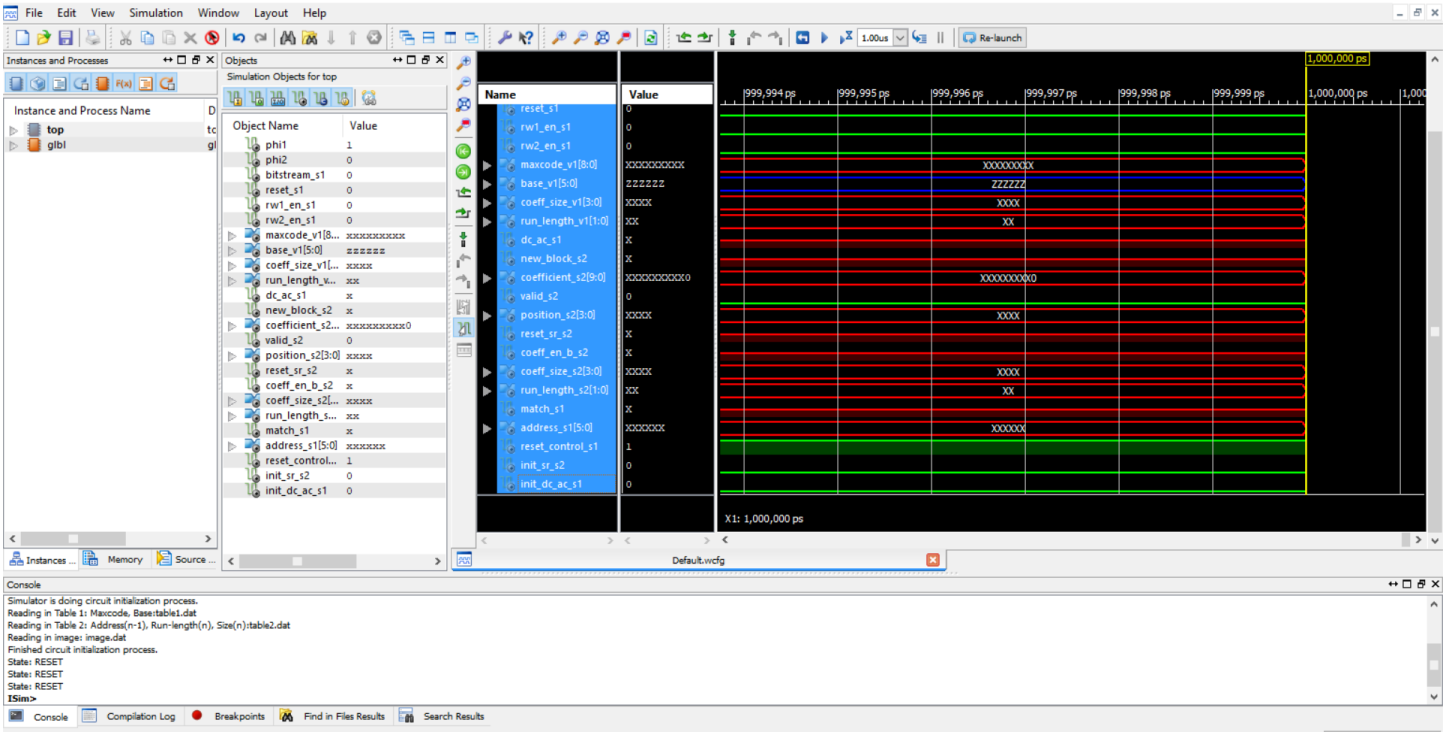
For test1.dat before execution:



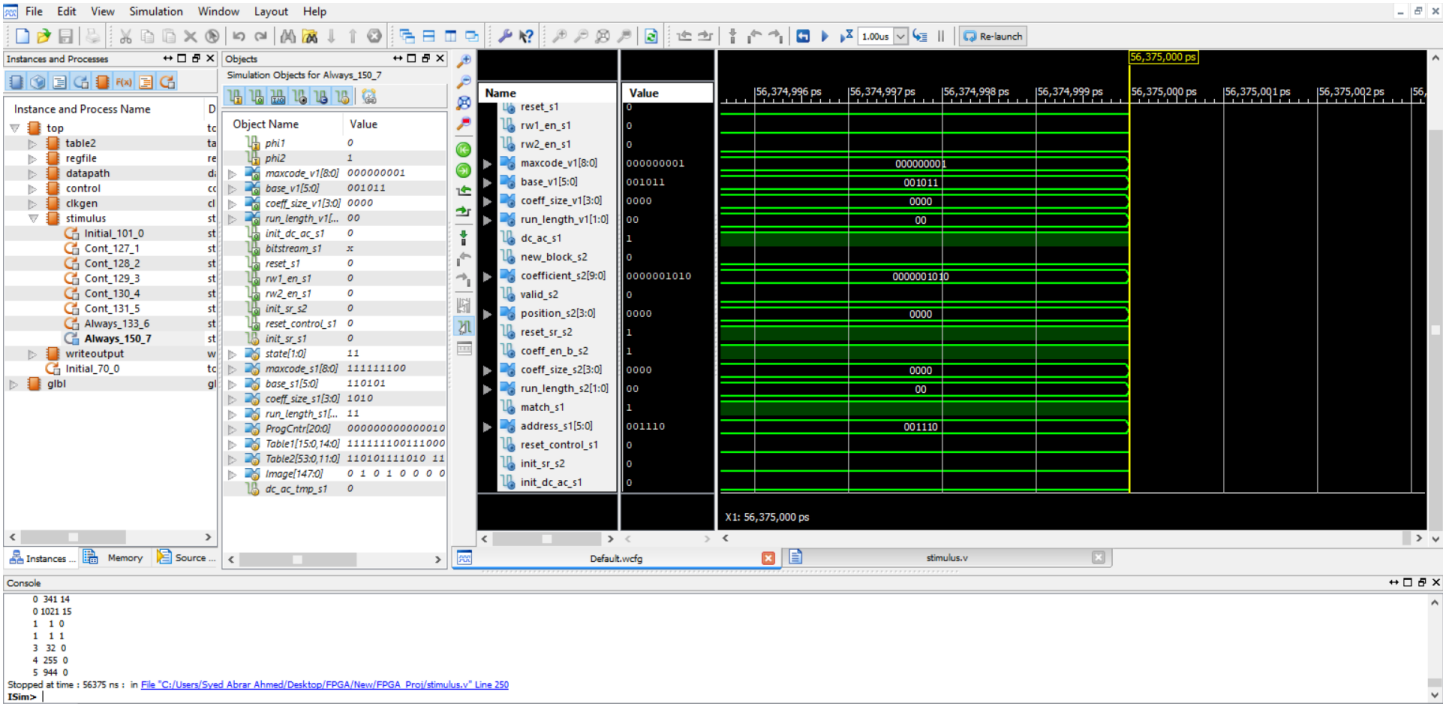
For test1.dat after execution:



For test2.dat before execution:



For test2.dat after execution:



References

- PDF document provided by Professor **Mingjie Lin**.
- http://www.eecs.ucf.edu/~mingjie/EEL5722/index.html
- www.verilog.com
- www.xilinx.com