
Name SID #

EEM16/CSM51A (Fall 2017)

Logic Design of Digital Systems

Prof. Ankur Mehta : `mehtank@ucla.edu`

Design lab 2 | **Assigned Wednesday Nov. 8, 2017**
due 4pm Wednesday Nov. 22, 2017

Instructions

This lab is to be done individually. You may consult with others to share thoughts and ideas, but all of your submitted work must be yours alone. Be sure to indicate with whom you've collaborated and in what manner.

You may use any tools or refer to published papers, books, or course notes. You're allowed to make use of online tools such as Logisim, WolframAlpha, etc., provided you properly cite them in the provided file.

Submission procedure

Upload your Verilog files, as named in the skeleton code provided, to the Gradescope autograder. Be sure to include the Verilog modules, testbenches, and coversheet comments.

0 Morse decoding using a neural network

0.1 Overview

Our design labs will revolve around creating a system to decode an input Morse code bitstream into an alphabetic character display. The key element in this system will be a feedforward neural network with 4 input nodes representing the Morse pulse widths, 4 hidden nodes, and 26 output nodes representing the likelihoods of each character. A schematic of the system is shown in Fig. 1.

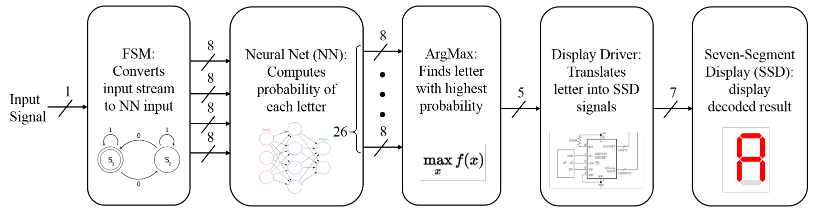
Input Signal
1
FSM: Converts input stream to NN input
8
8
8
8
Neural Net (NN): Computes probability of each letter
26
8
8
ArgMax: Finds letter with highest probability
$\max_x f(x)$
5
Display Driver: Translates letter into SSD signals
7
Seven-Segment Display (SSD): display decoded result

Figure 1: Block diagram of Morse decoder

0.2 Input

The input will be a 1-bit time series indicating the Morse code pulses as shown in Fig. 2. This time series will be parsed into a set of pulse widths (durations) using a finite state machine: each '1' pulse is separated by a short '0' gap, while sets of pulses defining a single character are separated by a long '0' gap.

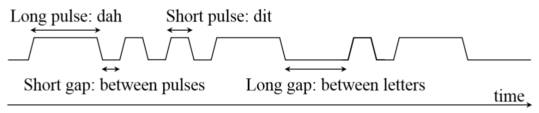

Figure 2: Input pulse timing diagram for the morse code sequence: −••− •− = PA

0.3 Neural net

A feed-forward neural net serves as a general purpose input-output function; it contains a number of weights and biases that determine the form of this function. Training these weights is out of the scope of this project, so the pre-trained weights will be supplied in a memory. The resulting network maps the (4) input pulse widths to the (26) likelihoods that those pulses code each possible letter of the alphabet.

The neural net is built of 3 layers of neurons, each fully connected to the layers before and after as shown in Fig. 3.

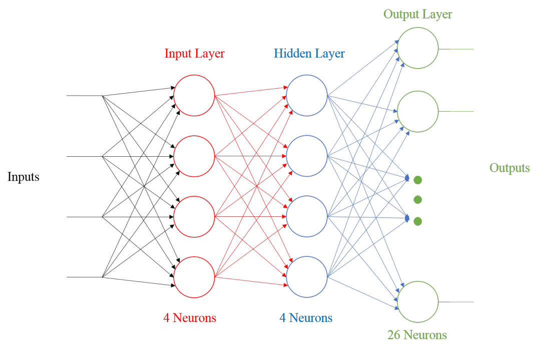

Figure 3: Connectivity diagram for the neural network

0.4 Neuron

Each neuron stores an 8-bit unsigned integer value. The input neurons will be assigned their values based on the measured pulse widths. The hidden layer and output layer neurons each compute their value as a function of the neuron values in the previous layer. This computation consists of three steps as shown in Fig. 4:

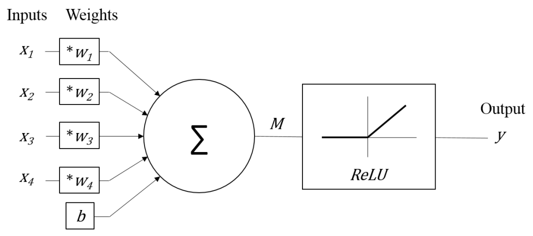

Figure 4: Mathematical operations within a neuron

- each incoming neuron value is multiplied by a pre-trained 7-bit signed integer weight to get a 16-bit signed integer weighted value,
- all weighted values are summed together with a pre-trained signed integer bias term, and
- the sum is rectified, thresholded, and scaled down to get the resulting 8-bit unsigned integer value.

0.5 Output

The values on the 26 output neurons are representative of the likelihood that the input Morse code sequence maps to each respective letter of the alphabet; the neuron with the highest value indicates the decoded letter. To get a single 5-bit value for the decoded letter (0=A, 1=B, . . . , 25=Z), the output values must be compared to determine the index of the maximum value.

0.6 Display

The decoded letter can be presented (with a few modifications) on a 7-segment LED display as shown in Fig. 5. Each segment of the display must then be driven appropriately as a function of the 5-bit letter index.

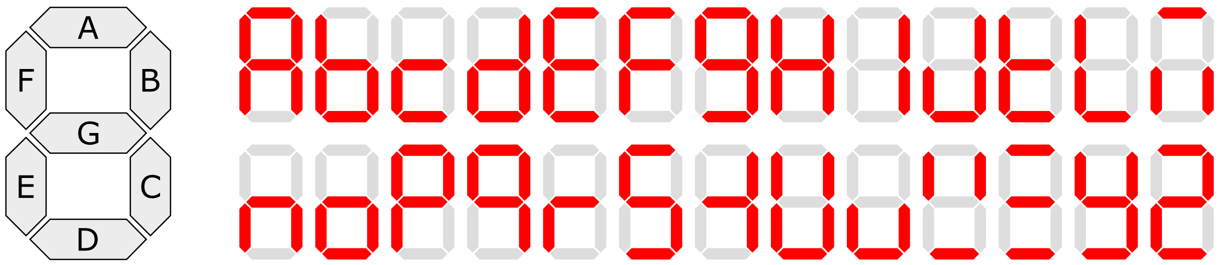

Figure 5: The alphabet as realized on a 7-segment LED display

2 Lab assignment 2

2.1 Multiplier

In this section you will implement the multiplier specified in pset 3 problem 2, except with 8-bit values instead of 4.

2.1(a). Create a declarative Verilog module to generate the lowest order partial product given the current (shifted) values of A and B as in pset 3 problem 2.0(a). Include a propagation delay of #1.

2.1(b). Create a declarative Verilog module to generate a one-bit output given an appropriate input as in pset 3 problem 2.0(c). Include a propagation delay of #4.

2.1(c). Using the above two modules, as well as the other modules provided, generate the structural Verilog implementing the multiplier as in pset 3 problem 2.0(e).

2.1(d). Set the clock period appropriately to ensure the dynamic discipline is met. Assume a setup time of #6 for the D-register. (You can assume the contamination delay meets the hold time requirements.)

2.2 Input de-serializer

In this section, you will create the module that takes in a 1 bit time series indicating the Morse code pulses as in Fig. 2, and generates a set of four 8-bit values measuring the pulse widths of the most recent Morse code letter.

2.2(a). Using any combination of structural and declarative statements, write a Verilog module that implements a counter to measure the width of an incoming pulse (1 value on the input bitstream) or gap (0 value on the input bitstream) in clock ticks as an 8-bit value. Output the following four values:

- the 8-bit width of the most recent completed pulse or gap,
- a one bit value indicating whether that width is short or long (the difference between short and long will be set as the Verilog macro `` `THRESHOLD ``),
- a one bit value indicating whether that width corresponds to a pulse (1) or gap (0), and
- a one bit flag that gets pulsed high for one clock cycle when a new width is ready.

2.2(b). Design a shift register module that stores and outputs four 8-bit values, with an 8-bit input value as well as one of the following input commands:

- CLEAR: Reset all stored values to 8'b0.
- LOAD: Shift the input value in to the lowest register, moving all stored values to the next higher registers.
- HOLD: Do nothing.

Implement this using any combination of structural and declarative Verilog.

2.2(c). Design an FSM based on the outputs of your width-measurement module that generates the commands for your shift register module, as well as a one bit flag that gets pulsed high for one clock cycle when the pulse widths stored in the shift register represent a complete Morse code letter. Recall that pulses within a letter are separated by short gaps, while letters are separated by long gaps. Implement this using any combination of structural and declarative Verilog.

2.2(d). Connect the FSM, shift register, and width-measurement modules using structural Verilog to create your input deserializer.

2.H EE89 extra assignment

2.H(a). Using a counter, build a clock divider module to generate a ~3Hz clock from the internal Basys3 100MHz clock. Display this clock on an LED.

2.H(b). Build a module to display the lowest 4 bits of each of four 8-bit values as a hex digit on the four 7-segment displays.

2.H(c). Synthesize the input de-serializer, using your 3Hz clock instead of the clock_gen module. Connect the 1-bit input stream to a push-button on the board. Display the results of the de-serializer on the 7-segment displays using the above display module.