
Name **SID #**

EEM16/CSM51A (Fall 2017)

Logic Design of Digital Systems

Prof. Ankur Mehta : mehtank@ucla.edu

Design lab 1 (draft) | **Assigned Monday Oct. 16, 2017 due 4pm Monday Oct. 30, 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 space below.

Submission procedure

(coming soon)

Collaborators

Identify with whom you've collaborated and in what manner, if any.

Online resources

Identify which online tools you've used, if any.

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.

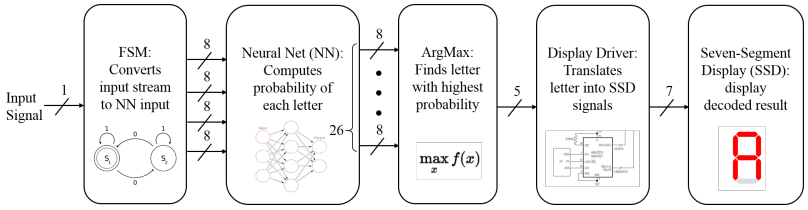

Figure 1: Block diagram of Morse decoder

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.

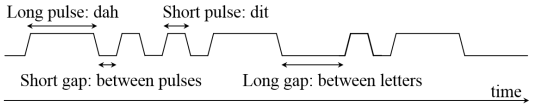

Figure 2: Input pulse timing diagram for the morse code sequence: — • • — • — = PA

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.

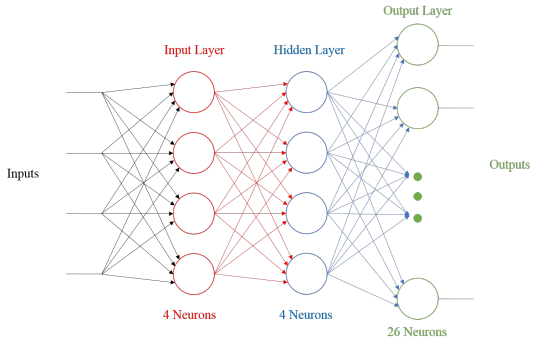

Figure 3: Connectivity diagram for the neural network

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 lengths. 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:

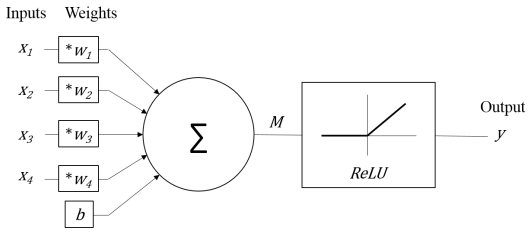

Figure 4: Mathematical operations within a neuron

- 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.

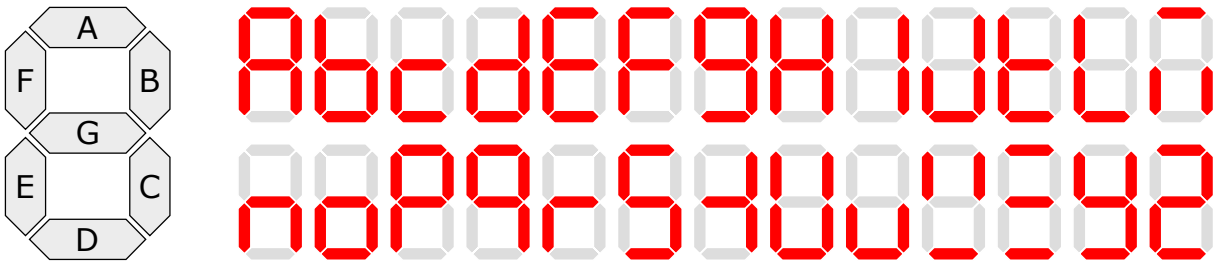

Figure 5: The alphabet as realized on a 7-segment LED display

1 Lab assignment 1

In this assignment you will implement the results from problem set 2 in Verilog.

1.1 Five input adder

1.1(a). Using the FA module provided, generate the structural Verilog implementing the FA5 module as in pset 2 problem 1.1(a).

1.1(b). Using your FA5 module, generate the structural Verilog implementing the 4 bit ripple-carry adder as in pset 2 problem 1.2(c).

1.2 Maximum index

1.2(a). Using Verilog operators, generate the declarative Verilog implementing the MAM module as defined in pset 2 section 2.3.

1.2(b). Using your MAM module, generate the structural Verilog implementing the maximum index function as in pset 2 problem 2.3(c).

1.3 Display

1.3(a). Using `case` statments, generate the procedural Verilog implementing the full ROM-based 7-segment display driver as in pset 2 section 3.1(b).

1.3(b). Using indexed assignments, generate the structural Verilog implementing the full mux-tree based 7-segment display driver as in pset 2 section 3.2(c)-(d).

1.H EE89 extra assignment

1.H(a). Synthesize the above modules using Vivado (Xilinx FPGA design environment).

1.H(b). Connect the 5-bit input of the display module to 5 of the switches on the board, and the output to a 7-segment display.

1.H(c). Compile and upload the designs to a Basys3 board and demonstrate their logic functionality to a TA.