6.111 Proposal - Voice Controlled Game Sprite

Aakanksha Sarda and William Huffman

November 5, 2013

Project Overview

The primary goal of this project is to implement a system which takes as input a user's voice saying one of a few commands and produces as output a visual representation of a game sprite following those commands on a computer monitor. This specific implementation seeks to use four spoken directional commands ("up", "down", "left", and "right") as well as one additional movement-halting command ("stop") to drive a pixel representation of a car on a computer screen.

In addition to requiring the use of the FPGA to implement the logic component of this project, a mic is required to record user input and a computer monitor is required to display the output of the FPGA. Both the mic and the computer monitor are part of the standard 6.111 lab kit, and have been used successfully in previous labs. The mic will be incorporated through the FPGA's audio microphone input in a way similar to that used in lab 5. The computer monitor will receive data via the VGA output on the FPGA in a way similar to that used in lab 3.

The bulk of the project is broken into two major components: a speech recognition component and a visualization component (See Figure 1 for the block diagram). The speech recognition component takes as input the audio in from the mic, as well as a listening signal which indicates that the user is ready to input a command (this will be done via holding down the enter key, as in lab 5). From this, the speech recognition module will interpret the mic input into one of the five legal commands and output that command into a 3-bit bus connecting to the visualization module. The speech recognition module will also output a 1-bit signal that is high for one clock cycle when a new legal command has been received - this serves to tell the visualization module that its current command information is out of date and should be refreshed.

The visualization component takes as input the 3-bit command bus from the speech recognition module and the 1-bit new command signal, and produces as output XVGA data for a 1024x768 pixel display. Upon a new command from the speech recognition module, the visualization module waits until the current frame has been drawn and then updates its internal state to begin executing the new command. In addition to having the primary goal of drawing a moving car to the screen, the visualization module has the additional goal of incorporating a map with walls and a finish line into the display. These additional goals require the visualization module to incorporate basic collision detection to determine when the car has hit a wall (at which point it will stop) and when the car has crossed the finish line (at which point the visualization will briefly display a message and then reset).

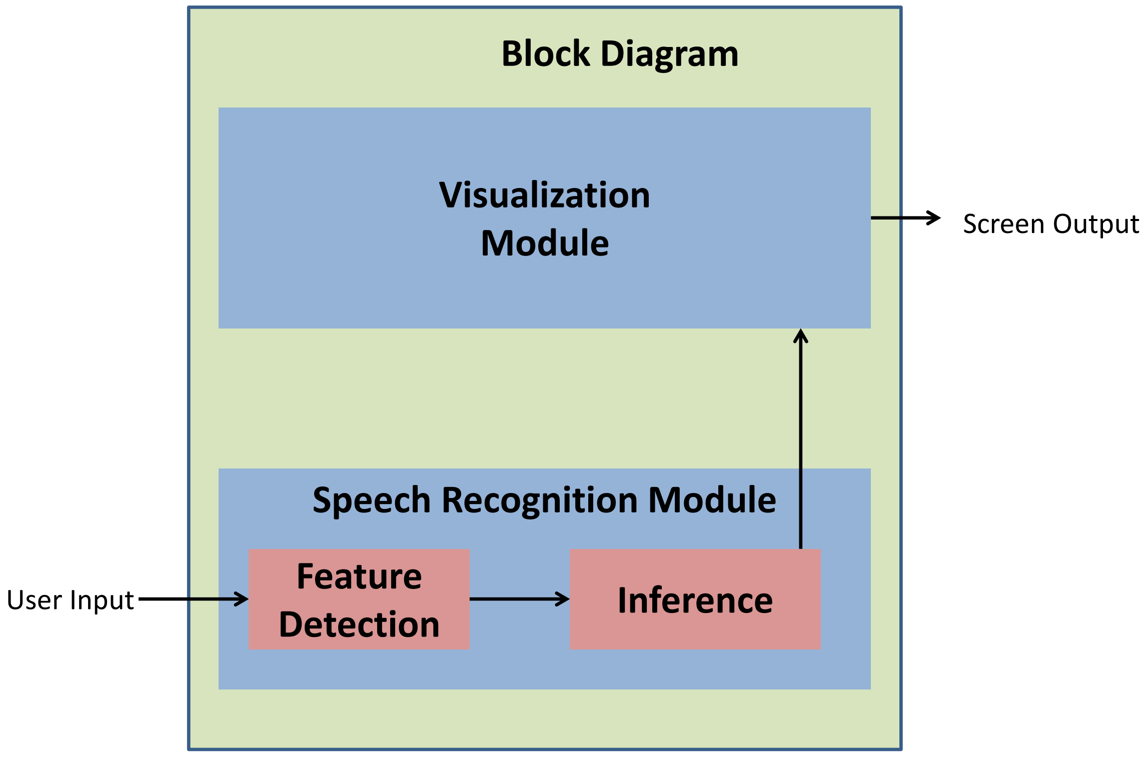

Figure 1: **Block Diagram Describing the Relationship between the Visualization and Speech Recognition Components**. The project uses a linear flow of data from the initial user input into the speech recognition component. Inside, the speech recognition component uses two main sub-components: one for detecting user input features, and the other for conducting inference on the user input using these features. The inference results are sent to the visualization component, which translates the identified command into an on-screen change in the car sprite's location.

Speech Recognition Module - led by Aakanksha Sarda

The speech recognition module is broken into two major submodules: the feature detection and the inference components (See Figure 2 for the wiring diagram). The feature detection component is responsible for taking audio input from the microphone, and converting it into a stream of feature vectors, which is an intermediate representation used by the inference component to decide which word was spoken.

Feature Detection

The feature detection component takes as input a signal indicating that the user is speaking (from a button press), and audio sampled at 44.1 KHz from the labkit mic input. The audio recording module is responsible for down-sampling the audio to 16 KHz (which is standard for human speech signals), breaking it up into chunks of 512 samples, and passing the chunks on to the FFT module. The chunks will be stored temporarily in the labkit BRAM, as lab 5. We will use a pre-existing implementation for the FFT module. The output of the FFT module (the amplitudes of the different frequencies) will also be temporarily stored in the labkit BRAM, and passed on to the peak detection module. The peak detection module will make use of a common comparator tree structure to detect the 6 highest-amplitude frequency components in the speech sample. The tuple of frequencies corresponding to the 6 peaks comprise the ”feature vector”, which is passed on to the inference module.

Inference

The inference component takes in a stream of feature vectors, and needs to output a recognized command. The feature vectors are first stored in a buffer, and from where they are accessed by the decision module. The decision module takes pre-calculated, hard-coded, parameters from the vocabulary module (which is just a ROM); these parameters correspond to each of the words in the vocabulary and will have been pre-calculated on a computer with MATLAB.

The decision module will pass on the received feature vectors and the the vocabulary parameters to the Gaussian Mixture Model PDF (GMM PDF) module. The GMM PDF module is contains a ROM from where it can look up a similarity score based on the feature vectors and the vocabulary parameters.

An optional extension (time permitting) will have a Hidden Markov Model module producing the similarity score instead. In either case, the similarity scores will be passed on to a common comparison tree module which will return the closest-matching word (the word with the highest similarity score). The decision module will then pass this matched word on to the Visualization module.

Testing

We will follow a hierarchical testing strategy for the Speech Recognition Module. First, the speech recognition algorithm will be prototyped and tested in MATLAB. The MATLAB implementation will use fixed point arithmetic and use no built-in functions in order to model the FPGA setup as closely as possible. The algorithm will then be translated into Verilog, and implemented on the FPGA. Each module (audio recording, FFT, peak detection etc) will be separately tested against the corresponding MATLAB implementation, to make sure that it produces the correct output. We will rely heavily on output to the Logic Analyzer in order to debug the modules. Then we will test the Feature Detection component, the Inference component, and finally the Speech Recognition Module as a whole. This integrated testing is vital because comparing each individual module with the MATLAB implementation will only verify the numerical correctness of the outputs, but not whether they are produced at the correct time. The timing could be verified by routing easily identifiable dummy data (say, a single-frequency sine wave) through the system, and verifying that a valid result is produced at every stage and at the end.

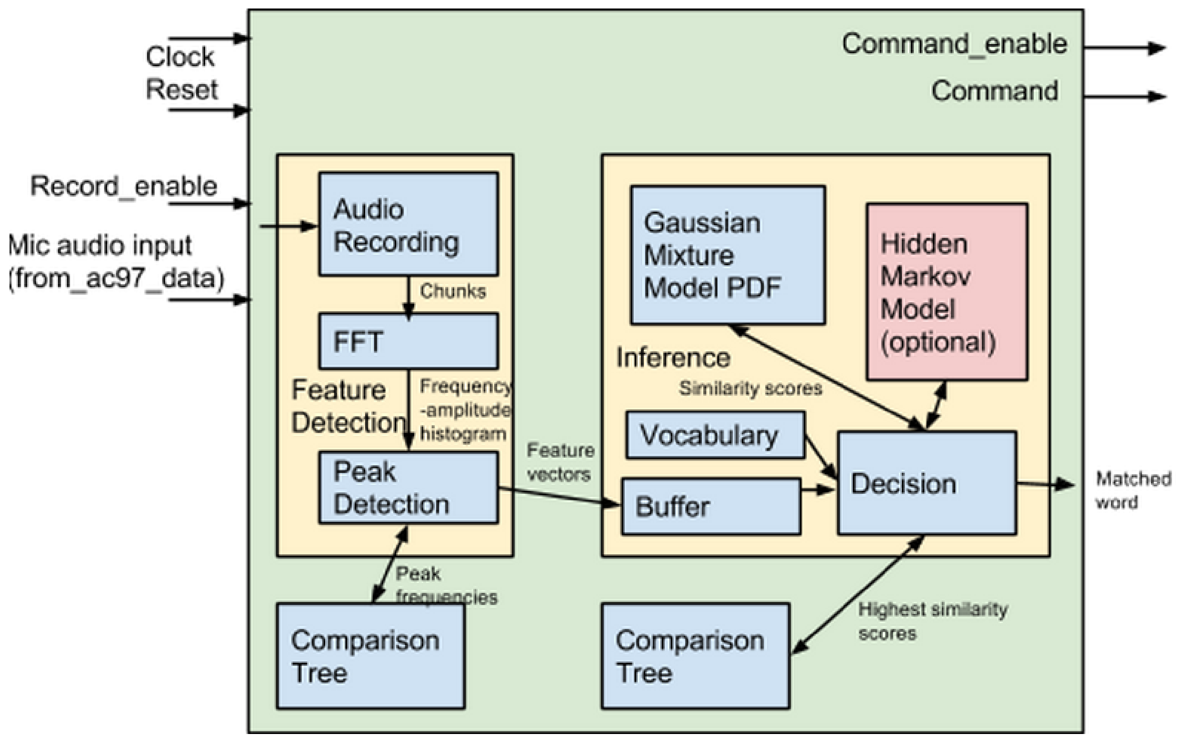

Figure 2: **Organization of the Speech Recognition Module into Feature Detection and Inference components**. The Feature Detection component is responsible for recording the audio input and processing it to produce an intermediate representation (corresponding to the dominant frequencies of the speech sample), which is used by the Inference module to decide which word from the vocabulary was spoken. The Comparison Tree module is a common parameterized module that will be instantiated by both the Feature Detection and Inference components (with different parameters).

Visualization Module - led by William Huffman

The visualization module is broken down into five major sub-components (See Figure 3 for the wiring diagram). The offsets module contains the simple state machine for transforming between commands from the speech recognition module and the state of the visual display. The collision module contains a set of rules for determining if a particular position of the car will result in a collision with a wall or the finish line, which is then sent back to the offsets module to be taken into account for determining the car's final position. The pixel logic module takes in an offset from the offsets module describing the final position of the car, and uses this to offset the display of the car and the map in the appropriate places. The car, map, and done modules each contain a pixel buffer corresponding to the size of that object to be displayed, which tells the pixel logic module what color should be displayed for each pixel on the screen. The VGA Display module takes the output from the pixel logic module, incorporates vsync and hsync signals, and sends the entire frame buffer off to the display using the VGA connection.

In addition, two minor modules control the timing for the visualization component. A 65MHz clock module provides a sufficiently fast clock signal for the visualization component to produce XVGA signals. An input sanitization module synchronizes the input from the speech recognition module to the 65MHz clock for use in the visualization module.

The offsets and collision modules

The offsets module is a relatively simple FSM that changes its state based on the speech recognition output. The state is then used to increment the offset output, which is a 20-bit bus that contains information concerning the position of the car and the state for the display: 10-bits for the horizontal position, 9-bits for the vertical position, and 1-bit for whether or not to display the done screen.

The offsets FSM has six states: five of which correspond to the five input commands recognized by the speech recognition module, and one of which corresponds to the done screen to be displayed after the car crosses the finish line. In the stop state, the car's offset does not change. In each of the four directional states, the car should update its position to have moved a number of pixels (based on a hardcoded velocity to be determined during testing - this value will be roughly 5-10) in that direction. This updated position is sent to the collision module, which responds with whether this new updated position is valid (if not, the updated position is reset to the previous position). Finally, the updated position is loaded into the offset module, which is sent off to the pixel logic module to display the car in the appropriate position onscreen. Finally, the sixth state is the done state, which should set the done bit in the offset bus high. The done state is triggered by the collision module detection a collision between the car and the finish line, and contains a counter which counts down five seconds, after which the offset module resets to the car's initial position in the stop state.

The collision module takes as input a 19-bit position (encoded in a similar manner to the offset bus described above), and gives a 2-bit output describing one of three possible outcomes: 0 is no collision, 1 is a collision with a wall, and 2 is a collision with the finish line. It contains a series of conditional statements checking whether the queried position is in a valid region or in the finish line area.

The offsets module will be initially tested independent of the collisions module in order to run the state machine through all of its state transitions. This is done by simulating the speech recognition output and checking whether the offset matches what is expected. Once this testing is completed, the collision module will be incorporated and the entire offset-collision combination will be tested through simulated sequences of commands testing the offsets FSM states and collisions with walls and the finish line.

The Pixel Logic; VGA Display; and Car, Map, and Done modules

The car, map, and done modules are all simply ROMs which contain information about pixel colors at different locations in the image. The car module contains pixel data for representing a car on screen, the map module contains pixel data for representing the map on the screen, and the done module contains pixel data for representing a done screen upon crossing the finish line. Each pixel location contains 8 bits for each of red, green, and blue. These modules require little testing beyond confirmation that they contain the correct data, as there is no actual logic involved, only data storage.

The pixel logic module is responsible for querying the car, map, and done modules to grab color data for each pixel to be drawn to the screen, and outputting a combination of these pixel colors to finally be drawn to the screen. As the car moves around the screen, the specific pixels queried for the car are offset from the pixels being drawn by an amount equal to the car's position, which is taken into account through the offset input from the offsets module. The car and map pixels will be combined via a simple OR, as the car and map will not overlap. The done pixels will override the car and map pixels if the done bit is asserted in the offset output, otherwise they will be ignored.

This pixel logic module will be tested initially in combination with the car, map, and done modules by checking its pixel output through a testbench which uses various offset inputs and expected pixel output values.

The VGA Display module is a basic module which takes in the combined pixel data from the pixel logic module, incorporates vsync and hsync signals for VGA communication, and outputs the final signal through the VGA Display. As this module is very similar to that used in lab three, testing should be minimal and will be reduced to running the pixel logic module output through the VGA Display module to ensure that the correct image is displayed on the monitor.

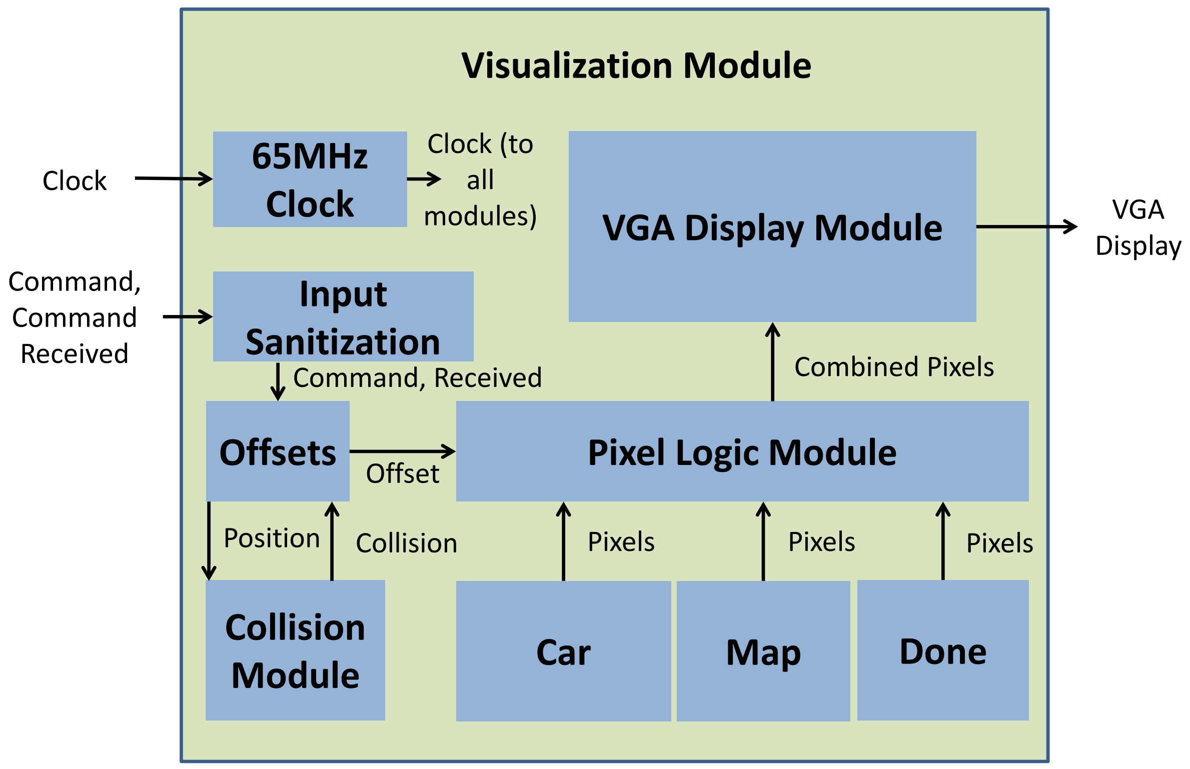

Figure 3: **Organization of the Visualization Component into Modules Handling Movement Logic and Display**. The input from the speech recognition component is sanitized and sent into the offsets module, which uses its internal FSM in conjunction with a collision detection module to determine where the car sprite should move. This position, in terms of an offset for the car sprite to be displayed on screen, is sent to the pixel logic and VGA Display modules to be rendered appropriately to the screen. The 65MHz clock module produces a sufficiently fast clock signal to output 1024x768 VGA frames.