6.111 Final Project Proposal: Dance Dance Revolution

Andrea Bolivar, Grace Quaratiello

Introduction

We are implementing a version of the arcade game Dance Dance Revolution (DDR). The goal of the game is to score points by stepping in time with choreography that is displayed on a screen.

Hardware

A Dance Dance Revolution pad can be represented as a nine square grid where eight squares hold directional arrows and one square is the center "default" square (Figure 1).

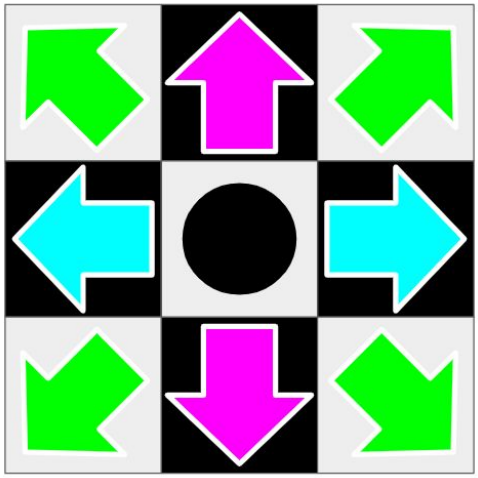

Figure 1: Dance Dance Revolution Example Pad

We will build a four foot by four foot enclosure comprised of four wooden rails. Two of the rails will each hold three lasers, and the other two rails will each hold three photodiodes. Each laser will be placed directly opposite to a photodiode that will detect whether or not the beam is being interrupted by the player's foot. This will create nine intersections of lasers at the centers of each of the squares in the grid, and we will be able to determine where the user is stepping at a given time based on the two (or more) lasers that are being interrupted.

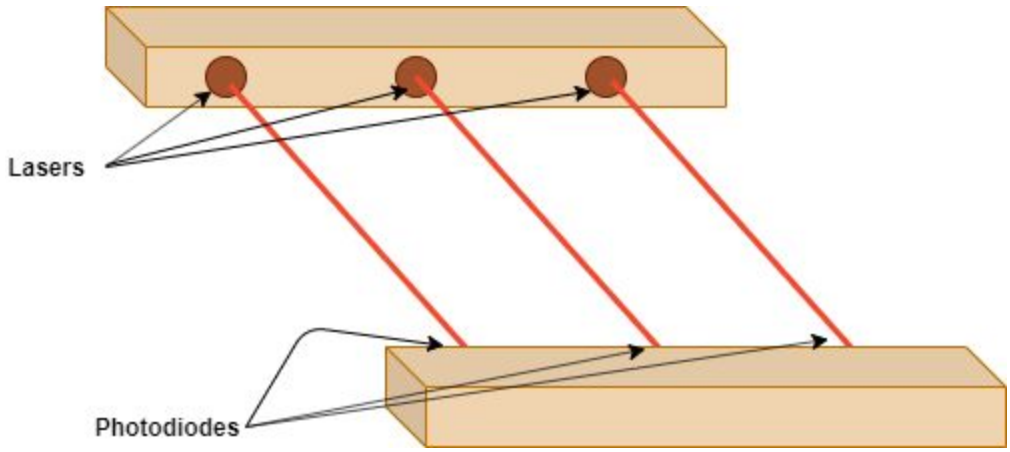

Figure 2: System of two rails, where three lasers are lined up with three photodiodes.

The photodiodes will measure the light transmitted from each laser in order to determine where the player is stepping at any given time. Each photodiode will be aligned with one laser beam (Figure 2). A player's foot can block a laser beam from reaching its corresponding photodiode, so we can use this set up to track where the player is stepping. We will use a voltage divider to measure the voltage drop across the photodiode and use a MOSFET as a buffer.

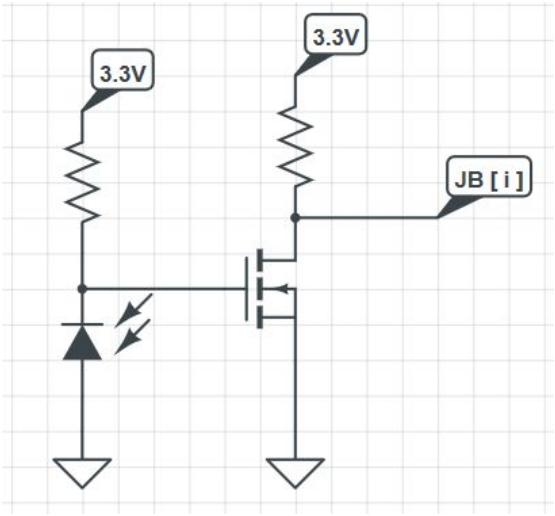

Figure 3: Photodiode Circuit Diagram

Block Diagram - Entire System

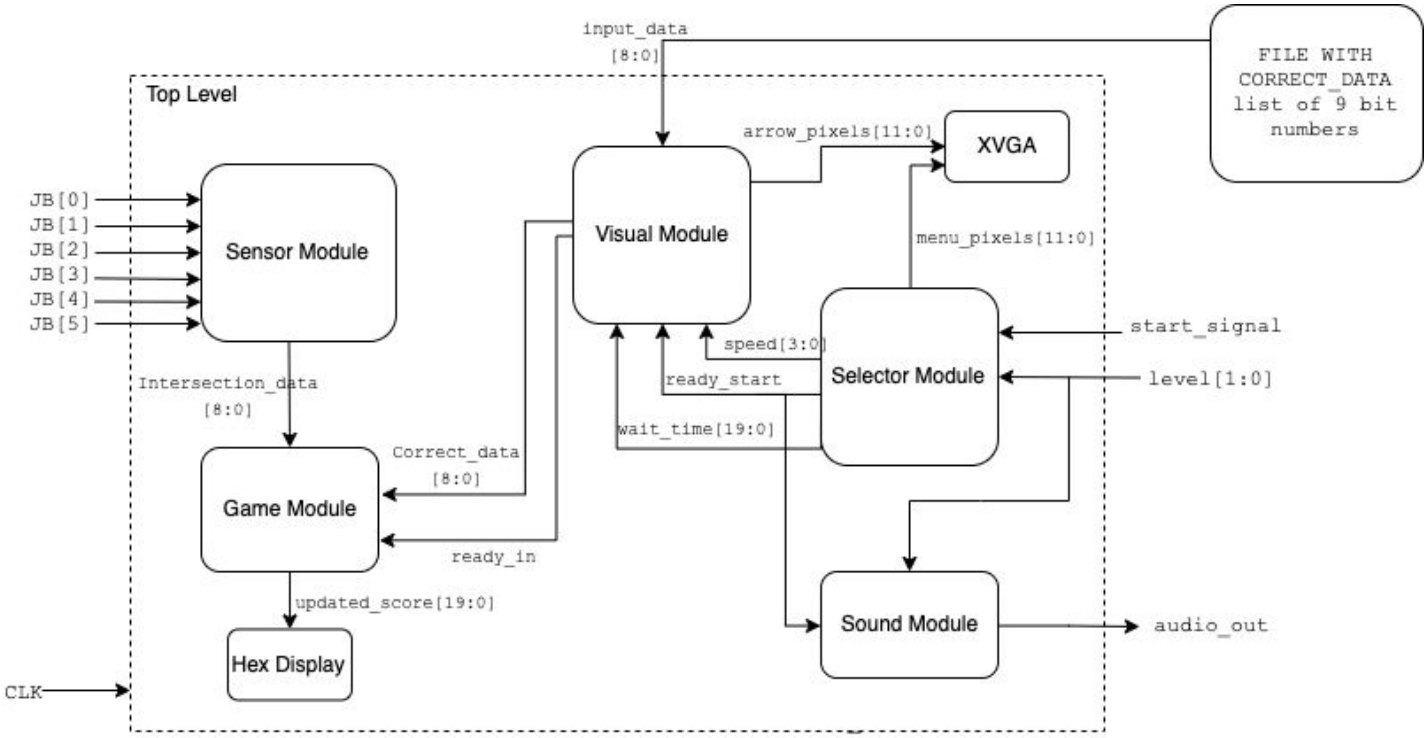

Figure 4: Block diagram for the top level module

Overview

The top level module will be comprised of the sensor, visual, selector, sound, and game modules. The sensor module will process the input signals coming from the JB ports based on the outputs from the six photodiodes to determine which squares a player is stepping in at a given time. This information will then tell the game module how to update the score once the `ready_in` signal is received from the visual module. The visual module will generate the arrows based on the correct data it reads in from an outside COE file with hard coded nine-bit numbers. Once the visual module determines that the arrows have reached the top of the screen, then that `correct_data` will be sent over to the game module to update the score along with the `ready_in` signal that tells the game module that it can adjust the score. The `speed` at which the arrows move up the screen and the handling of the start of the game will be evaluated in the selector module. The selector module will also generate the `menu_pixels` before the start button is pressed and determine the `speed` of the arrows based on the `level` input from the user with two switches. The `ready_start` signal along with the level will also be sent to the sound module to process the `audio_out` for the chosen level.

Sensor Module (Andrea)

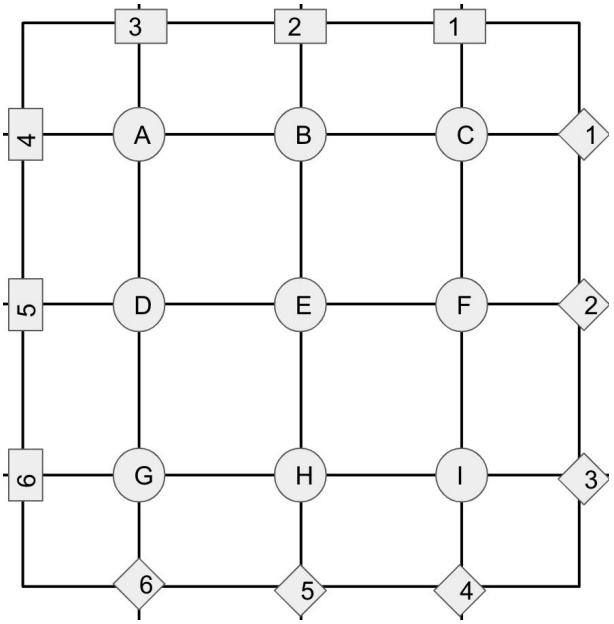

Figure 5: A bird's eye view of the setup where the rectangles are lasers, the diamonds are photodiodes, and the circles are intersections of lasers.

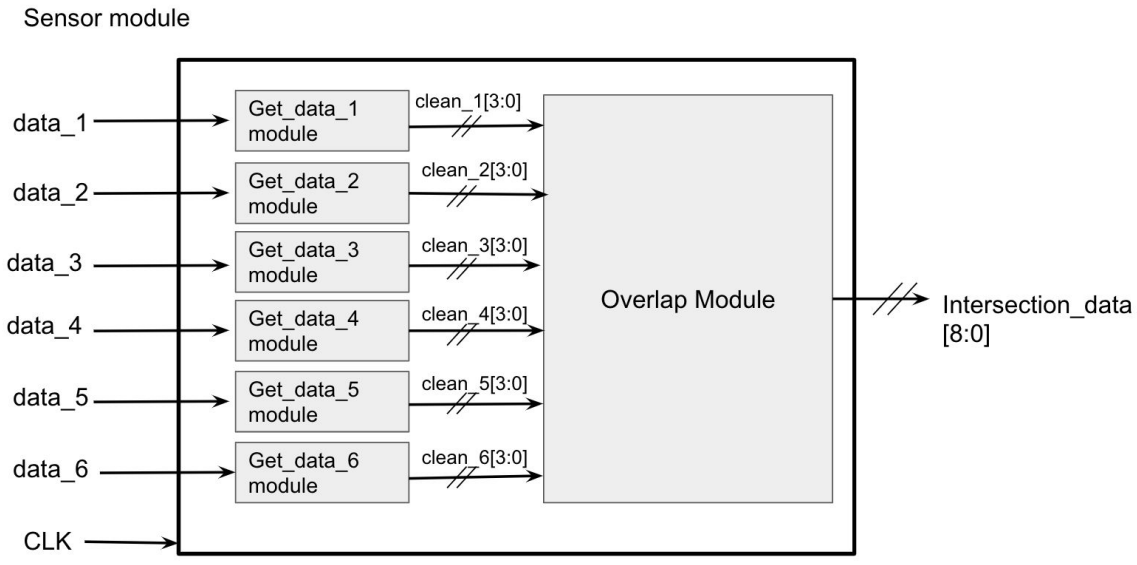

Figure 6: Block Diagram for sensor module

The sensor module will process the six inputs from each photodiode (`data_1, data_2`, etc.). The `get_data_1, get_data_2, get_data_3, get_data_4, get_data_5` and `get_data_6` modules will receive the incoming data from one of the corresponding JB ports and output 4 bits respectively. The first bit will be a 0 if light has passed and 1 if the light has been

blocked. This translates directly to a 1 if a player is stepping in the area since that will block the light. The last 3 bits of each output will determine which photodiode produced the output (as shown in the placement of the diodes in Figure 5). For instance, the two values that `get_data_1` can output are 0001 or 1001 since the first bit determines whether or not someone is stepping in that spot (A, B, C) and the last 3 digits are 001 to represent that this information is coming from the 1st photodiode.

The overlap module will take in six 4 bit inputs, and determine which of the nine sections have been stepped on. The output will be a 9 bit number with a 0 meaning someone's foot is not on that corresponding part and a 1 meaning that spot has someone's foot there. The number will be in the following format ABCDEFGHI depending on the value at each intersection of the lasers as shown in Figure 5**.**

This module can be tested by creating a test bench for the `get_data` modules to see if it is in fact creating the `clean_data` correctly based on the incoming data of 0's and 1's and the corresponding photodiode. There can also be a test bench written for the overlap module by passing in all possible combinations of `clean_data` to see if the expected `intersection_data` is valid.

Game Module (Grace)

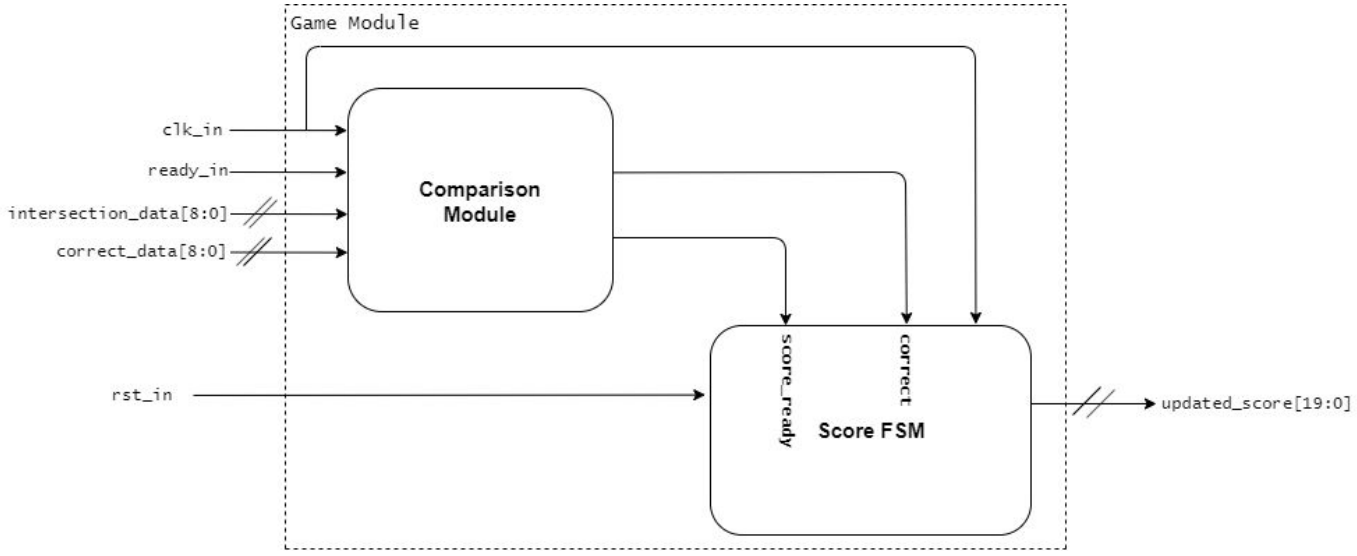

Figure 7: Block Diagram for the Game Module

This module takes in the processed data from the sensors and compares it to the correct data to determine if the player steps in the correct boxes at the right times. If `intersection_data` and `correct_data` are equal, then `score_ready` will be 1, indicating to the score FSM that the player was correct and should be awarded points. If `rst_in` is true, then

the score will reset to 0. Otherwise, the score will increment when the player is correct and stay constant when the player messes up.

To test this module we will create a testbench that will check how `updated_score` changes as the comparison module compares `intersection_data` and `correct_data.` We will simply input various `intersection_data` points along with the corresponding `correct_data` points and observe how the score changes. The `updated_score` output should increment when the inputs match, and stay constant when they don't. We can also test the `rst_in` button to make sure it resets the `updated_score` to 0.

Visual Module (Andrea/Grace**)**

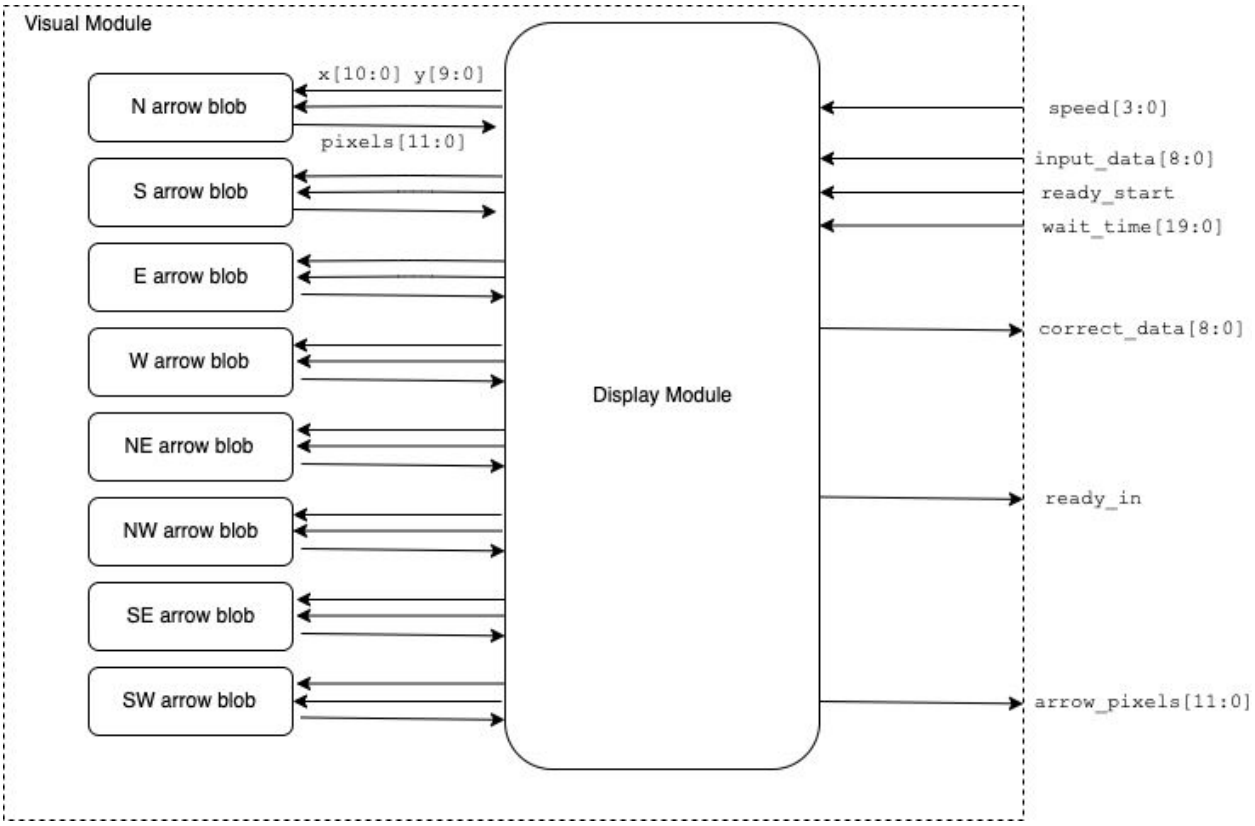

Figure 8: Block Diagram for the visual module

This module will handle the timing of checking for a correct step and determines which pixels should be displayed once the game has started. There will be 8 different arrow blob modules corresponding to the 8 different squares that a player can step on to earn points. The display module will read from the COE file (list of 9 bit numbers) of the correct combination of arrows in order to determine which arrow blobs to display, and it will read these values based on the `wait_time` that will reflect the level selected by the user.

This display module will also take in the current level of the game and the `ready_start` signal to know the speed the arrows should move with and when to start outputting pixels rather than 12 bits of 0, since the selector module should be providing the pixels at this point. Another input will be the `wait_time`, which is the number of clock cycles for which the display module must read incoming data from memory. This is an important step to ensure that data comes in at a constant pace and displays on the monitor with the correct speed and spacing. The arrows will start at the bottom of the monitor with fixed x coordinates, and their y coordinates will change (in the upward direction) based on the speed of the arrows. Once an arrow blob (or a combination of blobs) reaches a designated y-coordinate at the top of the screen, then the 9 bits of data associated with the combination of arrows will be sent over to the game module along with a `ready_check` signal to indicate that data is available and the game should check if the player made the correct step at the right time.

We will test this module by using a testbench with a set `speed` input value and a proper `wait_time` (with a smaller value than it would have in practice for timing reasons) for reading two 9 bit numbers representing defined arrow combinations. We can check to make sure that the visual module is only reading in an arrow combination after the defined time, and that the x and y coordinates reflect the speed at which they are moving up the screen, and that the `ready_in` signal is sent out once the combination of arrows reaches the correct vertical threshold.

Selector Module (Andrea)

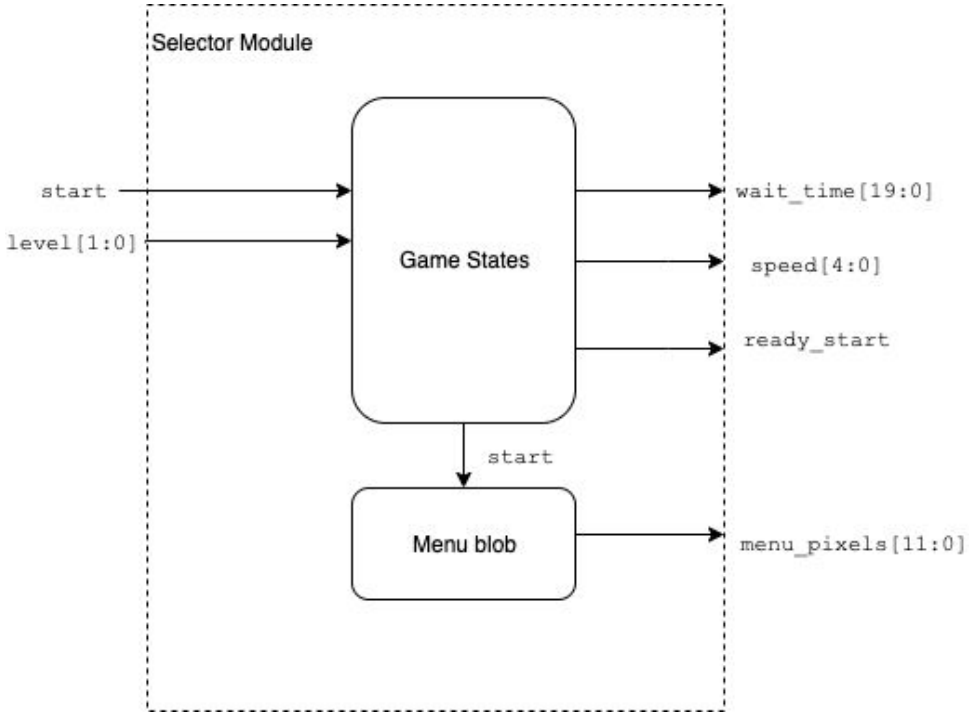

Figure 9: Block Diagram for Selector Module

The selector module will determine the `speed` at which the arrows move on the screen based on the `level` the user selects. It will also determine when the game has started. This module will take the `start` input (btnc) and a 2 bit number (from sw[1:0]) representing the level of the game (easy, medium, or hard). These two inputs will go into a smaller module known as game states that will determine the current state of the game and the values of important parameters that will define the game. The game state module will output the `speed` at which the arrows will move up on the screen, the `ready_start` signal, and `wait_time`, which will tell the display module how often to read from memory. Lastly, the start menu that will be displayed before the start button is pressed (indicating the beginning of the game) will be created in the menu blob. This smaller module will take in the `start` signal and determine the pixels to output. If the game hasn't started then it will output the menu pixels, otherwise, it will simply return 0's.

This selector module will be tested by creating a testbench where we can choose a level to start with and check if the appropriate `ready_start` signals, `wait_time`, and `menu_pixels` are produced based on when the start button is pressed.

Audio Module (Grace)

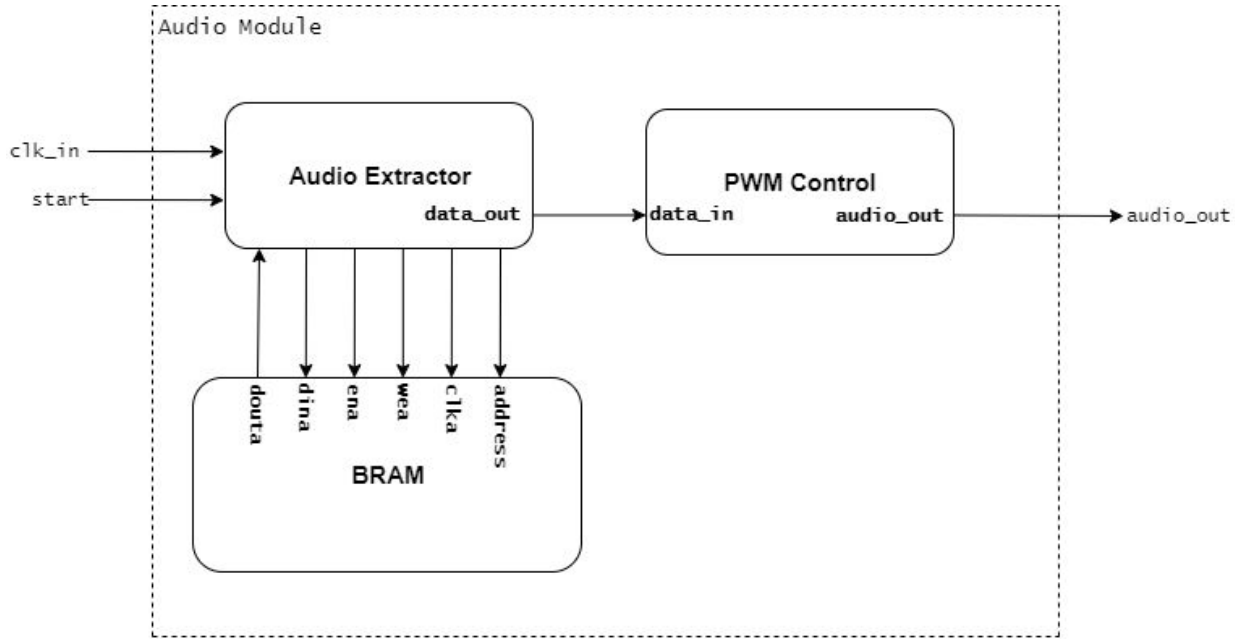

Figure 10: Audio Module

The audio module will handle the playing of the song attributed with the game being played. When the game starts, which will be indicated by a high signal from the start, the audio extractor module will start reading data from Block Random Access Memory (BRAM) and sending it to the pulse width modulation (PWM) module at a determined sampling rate. The address will be incremented by 1 every time the audio extractor reads data from BRAM, and `data_out` will remain constant in between reads. The PWM module will take in the digital data

and convert it to an analog audio output. This output will then play through a speaker connected to the audio jack on the FPGA. This module can primarily be tested by listening to the output after `btnc` is pressed.

Timeline

11/03	Finalized block diagram
11/06	Have the structure built and hardware tested
11/15	Complete and test sensor module and audio module
11/22	Complete and test game module, selector module, and visual module
11/29	System integration
12/6	Work toward stretch goals

Project Goals

Goals	**Stretch Goals**
- Create a functioning game with a start menu. - The score will count up normally without complex logic or states. - One song will be available with 3 difficulties that correspond to the speed of the game.	- Make the scoring FSM module more complex (streaks, bonuses, perfect vs. imperfect success) - Add more songs to memory and be able to select the desired song. - Option to pause the game during play. - Display the score on the monitor instead of the hex display

Challenges

- Building the structure for the lasers and photodiodes
- Lining up the laser with the photodiode since the laser has a small range
- The structure is secure and can not be accidentally kicked while playing the game