Piano Dance Revolution

6.111 Spring 2006
Final Project Report

Group #14:
Helen Liang, Wendi Li, David Meyer, Lucia Tian

Primary TA:
Theodoros Konstantakopoulos

Abstract:
Piano Dance Revolution is an amalgamation of the popular game Dance Dance Revolution and the famous FAO Schwartz piano in New York City, a giant floor piano played by touching the keys with ones feet. The system involves the design and implementation of four logic blocks: one that takes video input and analyzes the input signal to determine based on color filtering the location of the user's feet, a second that determines based on the location of user's feet the control logic that governs the overall system finite state machine, an audio control component, and a fourth block that holds logic to project a series of six screens onto the ground. Three modes of operation were implemented. In play mode the user can play notes on the projected keyboard; record mode allows the user to record songs he or she plays; game mode is similar to a piano version of DDR. While each individual block was implemented and functional, the overall system was unfortunately not integrated in the end due to time constraints. Nevertheless, the project was an educational success.

Table of Contents

I. Introduction 3
II. System Block Diagram 5
III. Video Input 5
A. Overview 5
B. Block Descriptions 6
IV. Step Interpretation 7
A. Overview 7
B. Block Descriptions 8
C. Testing and Debugging 10
V. Audio Control 12
A. Overview 12
B. Block Descriptions 12
C. Testing and Debugging 19
VI. Projection Output 20
A. Overview 20
B. Screen Descriptions 20
C. Block Descriptions 22
D. Testing and Debugging 26
VII. Game Mode 26
A. Overview 26
B. Block Descriptions 26
VIII. Interkit Communication 28
IX. Conclusions 28

Tables & Figures

Figure 1.1: Screenshot of Dance Dance Revolution Game 3
Figure 1.2: Screenshot of FAO Schwartz Piano 3
Figure 1.3 Piano Dance Revolution Logic Block Diagram. 4
Figure 2.1: Total System Block Diagram. 5
Figure 3.1: Video Input Block Diagram. 6
Figure 3.2: Two Camera Set-up. 6
Figure 3.3: Video Detection Images 7
Figure 4.1: Step Interpretation Block Diagram. 8
Figure 4.2: Overall system state transition diagram. 9
Figure 4.3: Final ModelSim simulation demonstrating state transitions and input pulses. 11
Figure 4.4: Final ModelSim simulation demonstrating key press detection. 12
Figure 5.1 Overall Audio Control System Diagram. 14
Figure 5.2 Audio Control Module Block Diagram. 15
Figure 5.3 Audiogen Block Diagram. 16
Figure 5.4 Record State Transition Diagram 18
Figure 5.5 Playback State Transition Diagram. 19
Figure 5.6 Sine Wave Debugging Waveform. 20
Figure 6.1: Welcome Screen 21
Figure 6.2: Mode Selection 21
Figure 6.3: Keyboard 21
Figure 6.4: Song Selection Screen 21
Figure 6.5: Enter Song Title Screen 22
Figure 6.6: Beat Selection Screen 23
Figure6.7 System Block Diagram of Projection Output Block 23
Figure 6.8 System Block Diagram for the Projection Output Display Logic Block 25
Figure 7.1 Game Mode State Transition Diagram 27
Figure 7.2: Game Console for Piano Dance Revolution 27

Table 5.1 Note Conversion Table 17

I. Introduction

The introduction of the blockbuster video game Dance Dance Revolution (DDR) into society revolutionized the entire concept of video gaming. In moving ones feet to the pattern of lighted pads and to the music of the DDR program, a generation of video gamers rose from their couches and integrated routine exercise and training in not only hand and eye but also feet coordination with the traditional act of gaming.

Figure 1.1: Screenshot of Dance Dance Revolution Game

Figure 1.2: Screenshot of FAO Schwartz Piano

In a parallel time frame, thousands of families visit the FAO Schwartz megastore in New York City each year. Children and adults alike enjoy the experience of walking across the floor length mechanical piano on the upper level, watching the keys light up as they step across, and hearing the keys they step on played in the store loudspeakers.

Piano Dance Revolution (PDR) fuses the concept of DDR and the FAO Schwartz piano into a single system that serves a three-fold purpose: song playing, sound recording, and video gaming. In the PDR system, a large three-octave, stacked piano is projected onto the ground.

In one mode of operation, Play Mode, a user wearing a bright red band around his or her ankles is able to step on any of the projected keys. The key being stepped on at the time will then light up and the note corresponding to the key will sound from the system speakers, similar to the function of the FAO Schwartz piano. No key will illuminate and no sound will play if the player's feet are lifted far enough from the ground so it seems to the system that the player is not stepping on any keys.

The second mode of operation, Record Mode, allows the user to record and store songs into memory to be retrieved later. The user selects a name under which the song to be recorded is stored, and a certain beat rate for the song. Then he or she steps on the series of keys that compose the desired song, which will be stored to memory as they are pressed. To stop recording, the user steps on the "Return" button, projected onto the ground next to the keyboard.

Game Mode, the last mode of operation, prompts the PDR system to function effectively as a piano version of DDR. One song stored in the PDR memory from Record Mode operation is selected by the user, and a desired beats per minute is chosen as well. The selected song is played through the loudspeaker and corresponding notes will illuminate on the projected piano. The gamer must now pay attention to which key is lit, either by looking at the projection on the ground, or looking at the gaming console on the monitor, and seek to step on the key while it is lit. PDR will keep track of how many keys the user steps on that are lit at the time of the step, and award a single point per correct step. The user will know he or she has stepped on the right key in the allotted time by either watching the score counter increment on the gaming console, or watching the portion of the gaming console that displays the current key stepped on by the gamer.

Modes of operation are interfaced to the user through a system of seven menus or screens, all projected onto the ground and interacted with by the user through stepping. The screens include a greeting screen, Main Menu, Game Menu, Record Menu, Beat Selection Menu, a game console, and the keyboard itself.

The entire PDR system is accomplished through four major logic blocks.

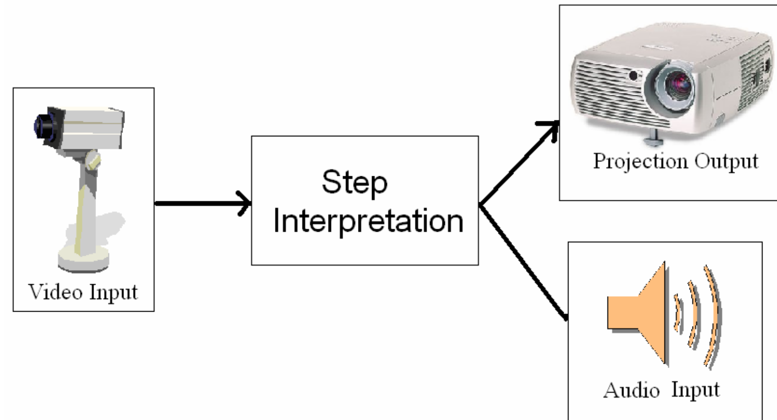


Figure 1.3 Piano Dance Revolution Logic Block Diagram

The video input block takes, processes, and stores input from video cameras. Two cameras are used to input an image of the user's feet from two directions perpendicularly. From there the input images are analyzed by the signal processing block, which determines based on the location of the bright red band on the screen where the user's feet are, and whether a button is being pressed. Based on the output of the signal processing block, the projection and audio blocks will control the computer monitor display and floor projection, and the audio output through system loudspeakers.

This document describes in detail the design of the system, the process of implementation and troubleshooting, and the results of the project. Improvements in the design process and in the system itself are presented at the end of this document.

II. System Block Diagram

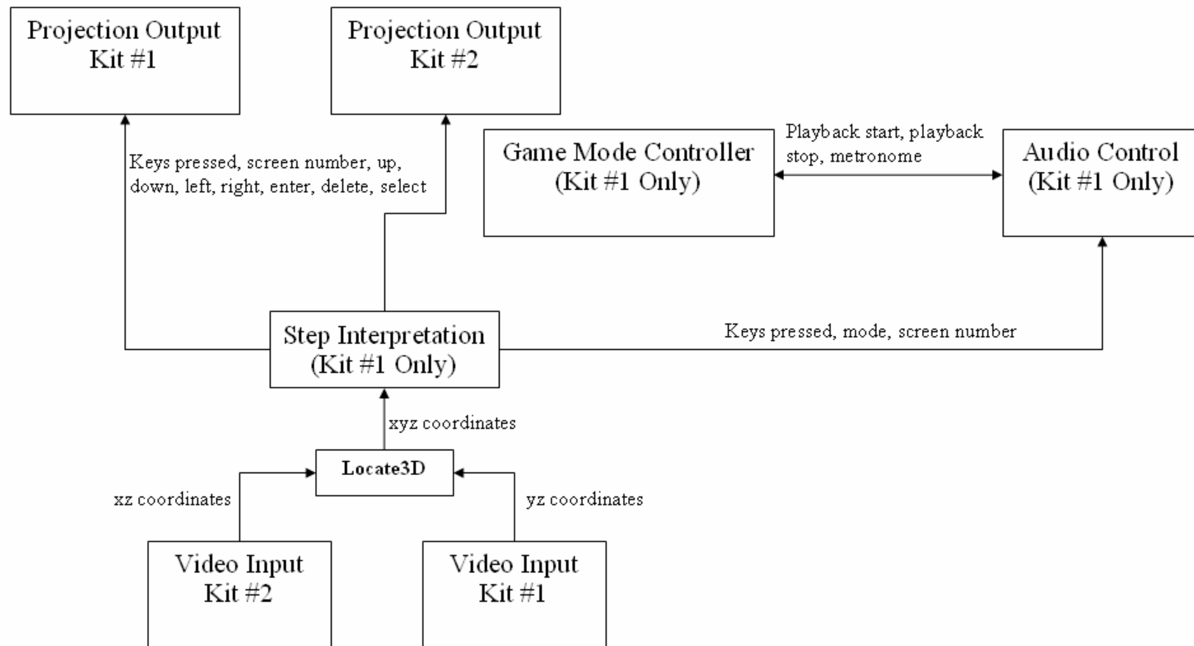


Figure 2.1: Total System Block Diagram

III. Video Input

A. Overview

The video input logic block is necessary to detect the location of the user's two feet in order to determine the xyz coordinates to output to the step interpretation block. The method of determining the location of feet is color filtering. One of the user's feet will be bound in a red band, and the other in a green band. The video block then takes as input two NTSC camera signals, one from each of two labkits, and analyzes the image to determine where the center of the red and green bands are, or if they are not detected at all. That center point or a non-detection signal is the vital output of the block.

The block diagram representing the video input block is shown in Figure 3.1

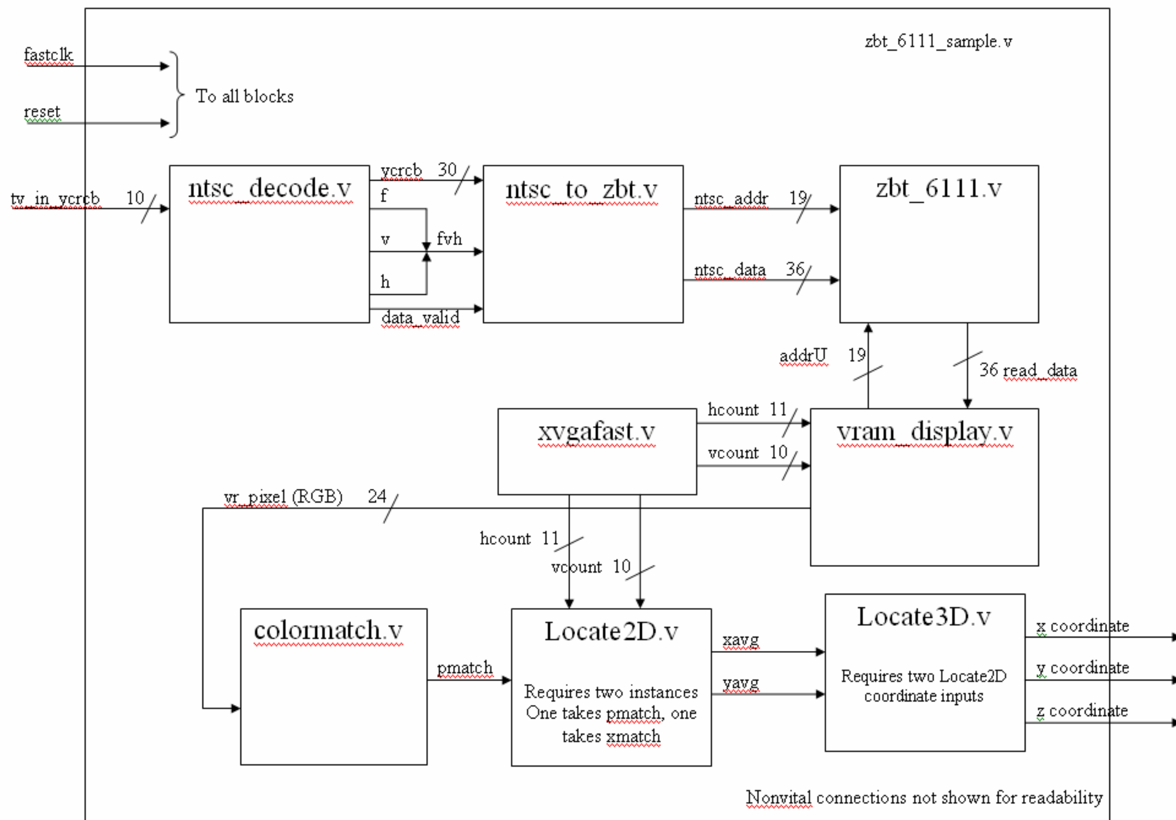


Figure 3.1: Video Input Block Diagram

The video input block is accomplished through six main modules. A number of support and debugging modules were also used.

B. Block Descriptions

On the top level is zbt_6111_sample.v which instantiates vital submodules of the video block, including zbt_6111.v, xvgafast.v, vram_display.v, ntsc_to_zbt.v, ntsc_decode.v, cameramatch.v, Locate2D.v, boxavg.v, and Locate3D.v.

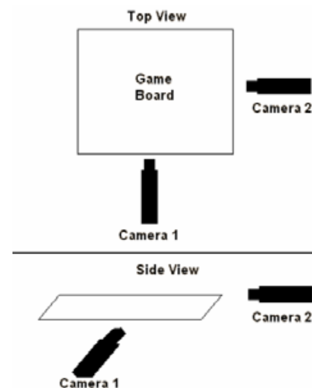


Figure 3.2: Two Camera Set-up

The two camera setup is shown in Figure ___. One camera determines the x-z coordinates of the user's feet while the other determines the y-z coordinates.

Input from the cameras is taken by ntsc_decode.v, which interprets the signals and passes the resulting Y, Cr, Cb signals to ntsc_to_zbt.v. The ntsc_to_zbt module converts the pixel values to an 18-bit RGB value in, which is then written to ZBT memory. Since ZBT memory has a width of 36-bits, two pixels can be stored in a single memory location with a single write. The RGB values are passed to zbt_6111, which is the ZBT interface that stores the image for use by the remainder of the system. Changes have been made to original sample zbt_6111 code to register the input and output.

The image stored into memory must then be analyzed to extract useful information regarding user intentions. Vram_display reads the RGB values from ram once every two clock cycles and outputs the RGB values read to the VGA output and to the processing modules. Image processing is accomplished in parallel to VGA output, and is performed by colormatch.v, Locate2D.v, boxavg.v, and Locate3D.v.

Colormatch takes as input RGB values from vram_display and determines whether the current pixel is part of either the red or green user ankle bands. To determine if a pixel is a part of the red band, the following RGB value ratios must be met: $5B < 10G$, $10G < 8B$, $10B < 7R$, $10G < 6R$, and $R > 100$. To determine if a pixel is a part of the green band, the following RGB value ratios must be met: $10R < 9G$, $9B < 10R$, $10R < 14B$, $13B < 10G$, $G \geq 71$. These values were determined experimentally. A 16x16 square was drawn onto the VGA output of the image, and the RGB values of all the pixels within the box were averaged, and the average RGB value was presented on the LED display. By then taking the camera and placing the red and green band within the drawn box, the average RGB values were acquired. The outputs of colormatch are *xmatch* and *pmatch*, which are high when a pixel is determined to be either red or green.

The Locate2D module takes as input *hcount*, *vcount*, and *xmatch* or *pmatch* from colormatch to then average all red and green pixel coordinate numbers. The final average of red pixels and green pixels are deemed the center points of the user's ankle bands and sent as output to Locate3D. Locate3D takes two sets of coordinates from two instantiations of Locate2D and determines the xyz coordinates of each ankle band. These xyz coordinates are the central outputs of the video input block, and are sent for further processing by the step interpretation logic block. Pictures of the detection mechanism and screens are shown in Figure 3.3 below.

Figure 3.3: Video Detection Images

IV. Step Interpretation

A. Overview

The step interpretation block comprises the control logic that links the input and output blocks, and directs the state transitions of the overall system. This is achieved through six modules: one top level module boardsel.v, and five minor FSMs mainmenu.v, gamemenu.v, recordname.v, beatselect.v, and keyboard.v. The block diagram representing the step interpretation logic block is shown in Figure 4.1.

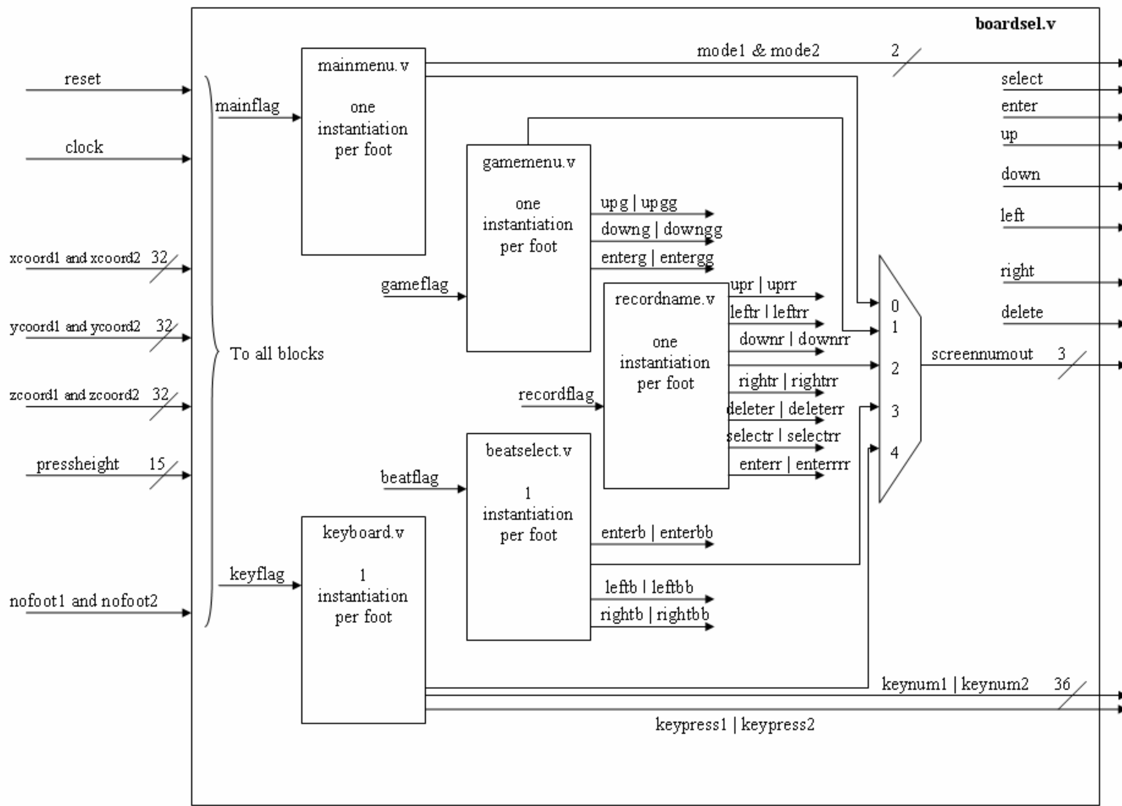


Figure 4.1: Step Interpretation Block Diagram

B. Block Descriptions

1. Board Selection

The overall PDR system can be thought of as one large FSM. The state transition diagram for the overall FSM is shown in Figure 4.2.

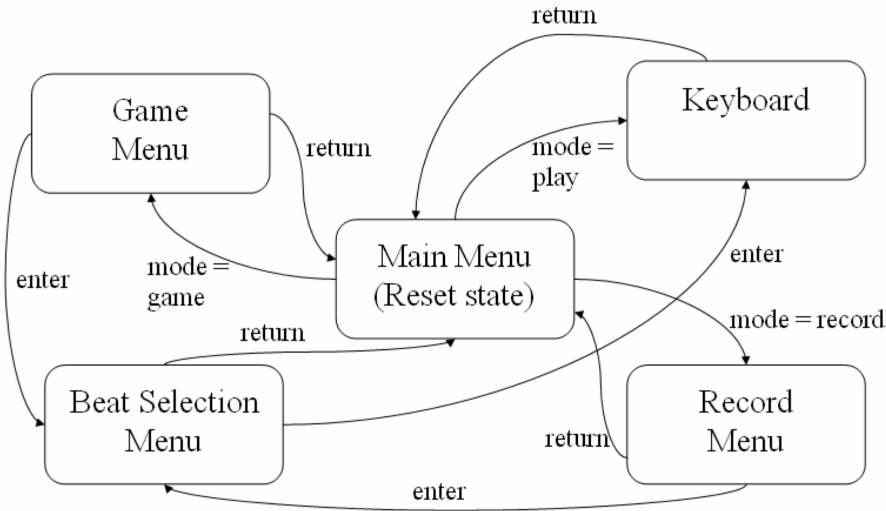


Figure 4.2: Overall system state transition diagram

The states of the overall FSM correspond to the different menus projected onto the ground. The states are:

1. **Main Menu**
   The Main Menu state serves as the default state where each set of transitions begin. The Main Menu state can be transitioned to from any of the other four states by a high *return* signal from each screen, or by overall system reset through button input from the FPGA labkit. Three different states can be reached directly from Main Menu. A logic high *play* signal switches the system to Keyboard. A logic high *game* signal transitions the system to Game Menu. A logic high *record* signal transitions the system to Record Menu.
2. **Game Menu**
   The Game Menu state is transitioned into only from a logic high *game* signal from Main Menu. There are two outward transitions: Beat Selection Menu given a logic high *enter* signal, or Main Menu given a logic high *return* signal.
3. **Beat Selection Menu**
   The Beat Selection Menu state is transitioned into from a logic high *enter* signal from either the Game Menu state or Record Menu state. There are also two outward transition options: Keyboard given a logic high *enter* signal, or Main menu given a logic high *return* signal.
4. **Record Menu**
   The Record Menu state is transitioned into only from a logic high *record* signal from Main Menu. There are two outward transitions: Beat Selection Menu given a logic high *enter* signal, or Main Menu given a logic high *return* signal.
5. **Keyboard**
   The Keyboard state is transitioned into from either a logic high *enter* signal from Beat Selection Menu, or a logic high *play* signal from Main Menu. The only outward transition state is Main Menu, accessed through a high *return* signal.

Transition control between the states is controlled by the top level module boardsel.v. Boardsel takes as input signals from the video input block telling the module whether each of the user's two feet have been detected, along with the x-y-z coordinates of both feet. Based on the current state of the system FSM, stored in *screennumout*, boardsel will send an enable flag to the appropriate minor FSM to retrieve the next state of the system and user input signals (*up, down, left, right, delete, enter*, key playing signals).

The system is designed to accommodate two feet input, and as such two instantiations of each minor FSM are necessary since each takes only a single set of x-y-z foot location coordinates. The final output signals from boardsel are the bitwise OR of corresponding output signals from the minor FSMs of each foot.

2. Minor FSMs

The minor FSMs of the step interpretation block, mainmenu, gamemenu, boardselect, recordname, and keyboard, provide the logic that determines whether it is the intention of the user to change the projected menu or activate a key on the keyboard. Each of the minor FSMs takes as input one set of x-y-z coordinates of one foot, along with a signal indicating whether that foot has been detected. Each also takes as input an enable flag, which is steady logic high whenever the state of the overall system matches the function of the minor FSM. The outputs of each are the next state of the overall system and user input signals.

There are several factors to consider when determining whether it is the intention of the user to be stepping on any of the input channels. It is necessary to check to ensure the user's feet have been detected, which is accomplished through the *nofeet* input from the video input block. It is also necessary to ensure the z coordinate of the located foot is below the key press threshold value, set by labkit switch. Finally, a physical hazard to consider is the potential for the projected menu to change, and in the next screen, for the user's foot to be stepping on a button in the next screen before the user has an opportunity to remove his or her foot. To resolve this issue, the logic in the minor FSMs requires that a key be "enabled" before it can be stepped on. The enabling process requires the user to lift his feet above 100 pixels above the step threshold directly above one of the buttons for any of the buttons to function as desired. Since the user must lift his foot to such a high distance above the threshold to activate the buttons, the system will know the user was not simply shifting his feet when a button is pressed.

After activation of buttons and determination of step, the minor FSMs determine, based on x and y coordinates of the user's feet, whether the user is stepping on a button. If so, the output signal for that button will set. If the particular button stepped on corresponds to a state changing button, such as *enter* or *return*, the next state of the system will also be determined and output to the major FSM.

C. Testing and Debugging

The step interpretation block was tested using ModelSim. The block was tested in two ways. First, the block was given an additional input *screennum*, which set the current state of the system. In this way, the state could be dictated and individual buttons in the state could be tested. After all buttons were determined to generate the correct output signal, *screennum* was removed, and replaced with *screennumout*, so state transitions could be tested. That is, given a screen and the right button, the screen will change and the system can drive itself.

The debugging process resulted in a number of features added to the original skeletal control logic. The enable flag system of boardsel was added after discovering that while the top level module was not registering changes to other states while in a certain state, the internal signals of minor FSMs continued to change while in other states. This presents the potential that when a state is returned to, the signals will have random initial values, and the buttons could already be activated. The flag system provides not only a way to stop the changing of internal signals, but also a way to zero all minor FSM signals while it is not enabled.

The enter flag system of the minor FSMs also resulted from the debugging process. It was shown in ModelSim that for the signal *enter*, due to the fact that it changes the state of the overall system, holds at steady high after being set since the state changes before the signal can be zeroed. This poses problems in that the next state, should it have an *enter* signal, may register an immediate enter and switch states before the user has the opportunity to react. With the enter flag, the *enter* signal can be zeroed before the next state, but ensure that the state still changes as desired.

Other minor problems were also discovered, such as the need for *mode* to be zeroed before a new mode can be stored due to the logic behind the bitwise OR, and the need for the next state variable to be set to the current state in each minor FSM before the state is reached again.

All bugs were corrected and the final simulation showed all desired state transitions and logic on signals. Final ModelSim results are shown in Figures 4.3 and 4.4.

Figure 4.3 shows desired state transitions (*screennumout*) in response to appropriate signals, along with user input signals set given appropriate x-y-z coordinates. Notice from this Figure that no signals pulse high until after the z coordinate input is a value greater than 100 pixels more than the threshold step value, set for this simulation at 20. Also notice that signals pulse in response to both feet.

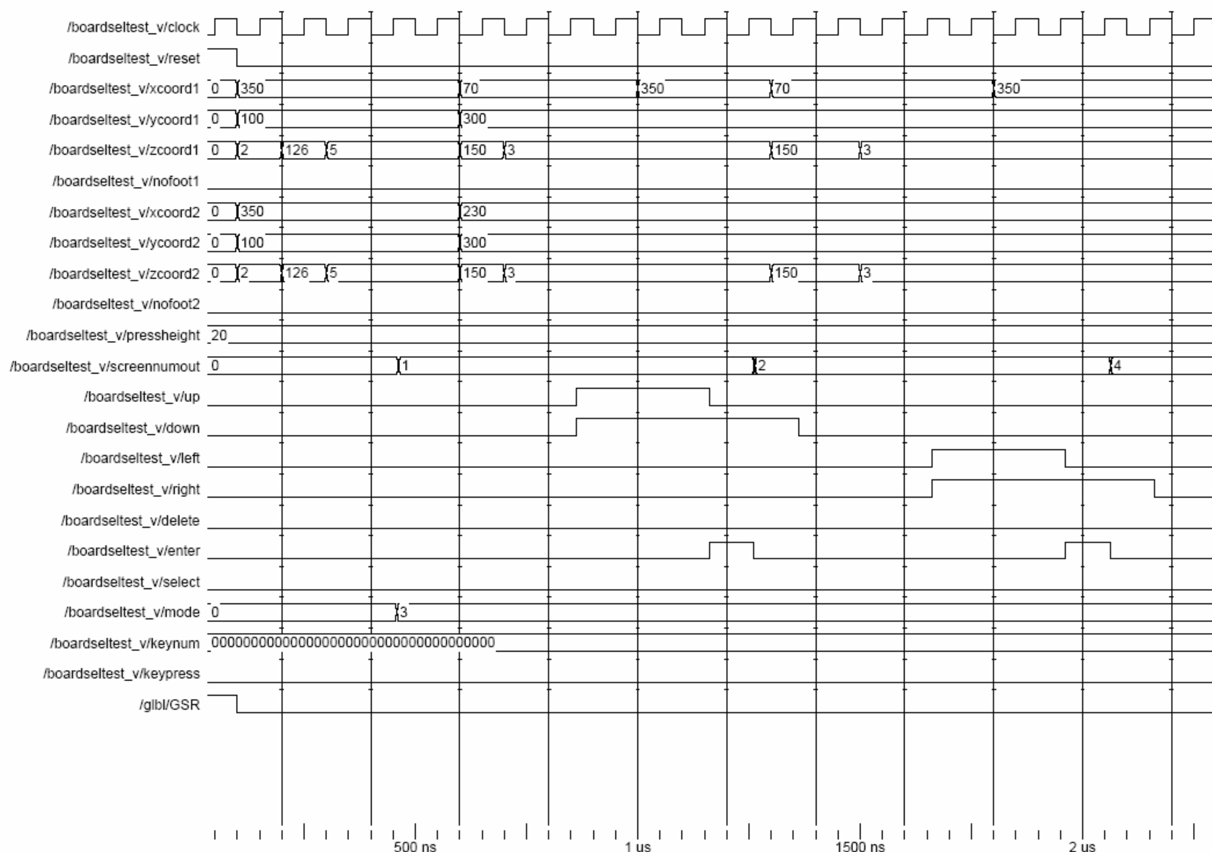


Figure 4.3: Final ModelSim simulation demonstrating state transitions and input pulses

Figure 4.4 shows a ModelSim simulation of the system in the Keyboard state with two different set of user steps. In the first set, two on bits are circled. This corresponds to the two notes played based on the

two given set of x-y-z coordinates. In this case, both feet have been detected and have been located on the keyboard. In the second set, one foot has been removed from its key, and is used to activate the return button then step on it. The end of the simulation shows the transition from state 4, Keyboard, back to state 0, Main Menu.

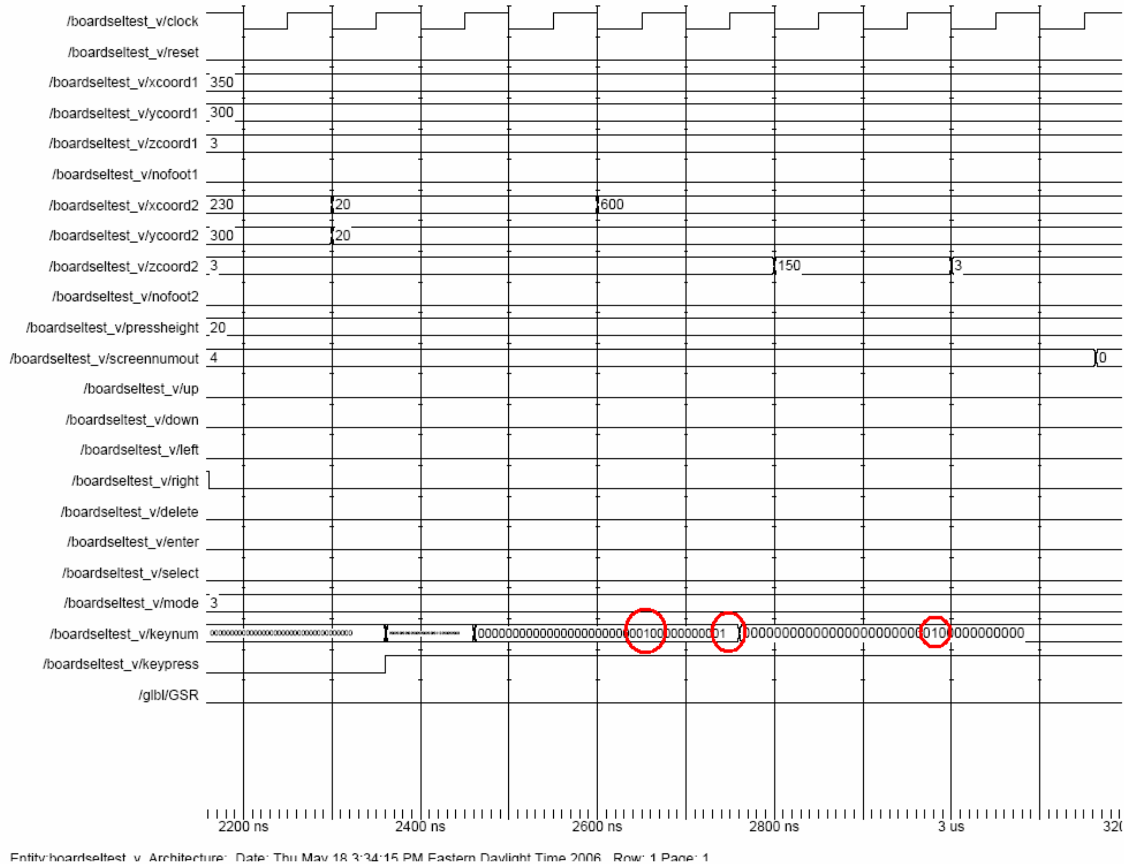


Figure 4.4: Final ModelSim simulation demonstrating key press detection

These simulation screens demonstrate the full desired functionality of the step detection logic block.

V. Audio Control

A. Overview

The audio control block provides all audio functions for the system. It acts as a link from the user, the step interpretation, projection output and game mode blocks to the AC97 and SRAM interfaces, which generate and store sound respectively. Audio control can play up to two notes simultaneously, generate metronome beats upon request and save songs to playback later. The overall system diagram connecting the audio control block to the integrated system is shown in Figure 5.1.

B. Block Descriptions

The detailed block diagram for the audio control module is shown in Figure 3.2. The module is constructed of four separate blocks – audiogen, record_mode, song_playback, and metronome. Audiogen is the core audio output module; it receives note and metronome inputs and generates sound through

interaction with the AC97 chip in the FPGA. The record_mode and song_playback modules handle the logic for sound storage and playback, respectively, and interface with the songmem module which handles reading and writing from SRAM. The metronome module generates a slow and fast metronome based on a BPM (beats per minute) count from the user.

1. Audiogen

   The audiogen module lies at the heart of the audio output portion of the system. Its function is to generate the correct signals to interface with the AC97 chip to generate sound. It can play two notes and a metronome simultaneously and can be expanded to play more notes very easily. A block diagram of its two-level complexity is shown in Figure 5.3.

   a) SFX Module

   The sfx block in the right half of Figure 5.3 is the portion of the audio system that generates the sound. The modules ac97 and ac97_commands are adapted versions of the AC97 driver by Nathan Ickes. Given 20-bit data values for the left and right sound channels, the system generates the correct timing and data signals to interface with the AC97. This is the only portion of the system which uses the 48kHz AC97 bit clock. The rest of the system is timed on the 65mHz system clock.

   The left_out_data and right_out_data values fed into the ac97 module are generated by the sound_fsm module. Sound_fsm contains two identical three-state finite state machines to generate a pure sine-wave note and a metronome beat. Please refer to Figure 3.4 for the state transition diagram. To generate the pure sine wave tones for up to two notes, the module contains two instantiations of the sin_fx module. More instantiations would enable the module to produce more simultaneous notes. When a note is played, the two modules count through a sine lookup table at the correct rates note1_N and note2_N to output samples of the sine table at 48kHz. The samples are averaged and fed to both audio channels. The metronome sound, timed on the metronome input, generates sound in much the same way, but uses a slash wave sound effect instead of the sine look up table. The one-bit metronome signal pulses high for one clock cycle each time the metronome is to be heard.

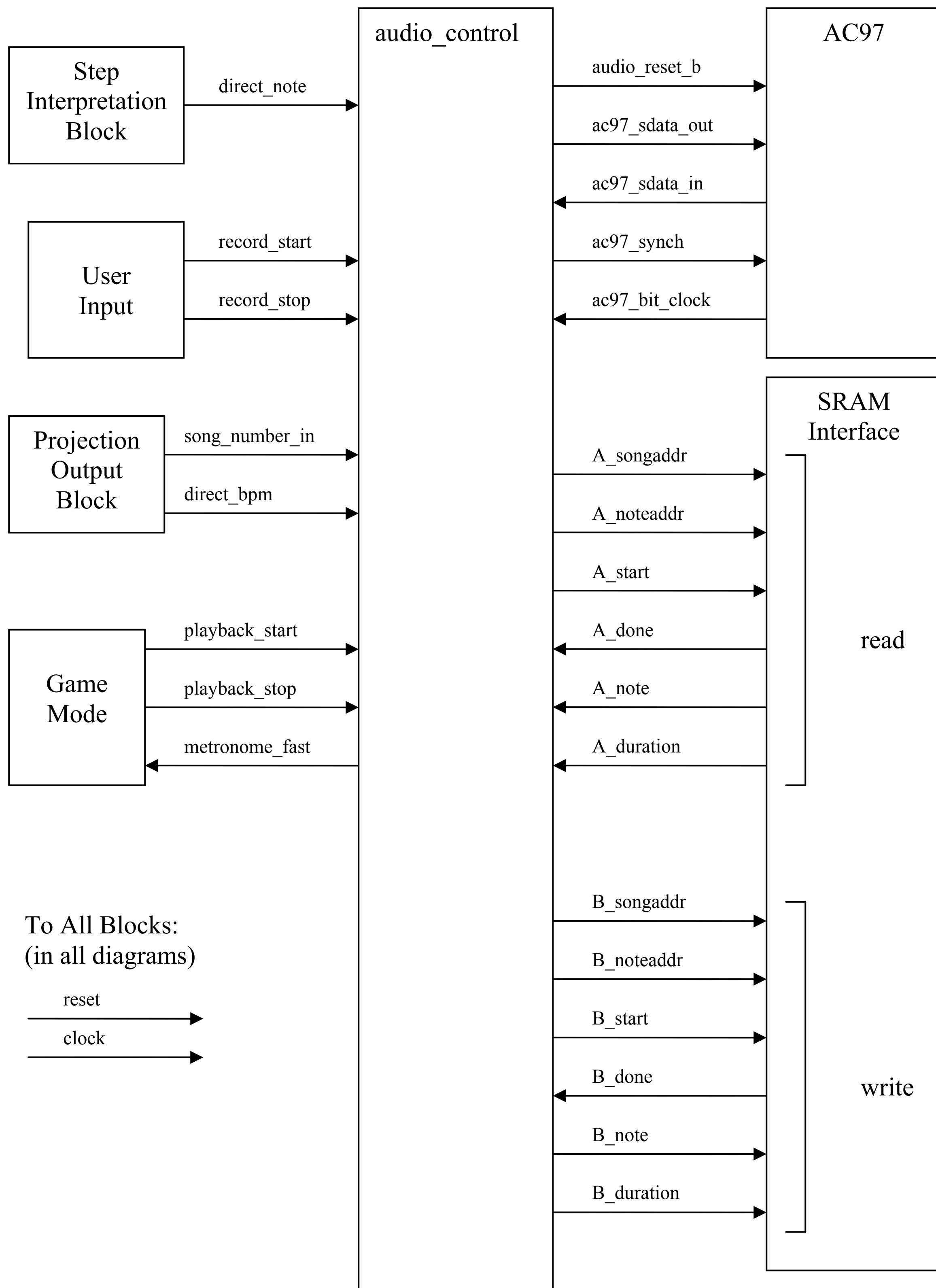


Figure 5.1 Overall Audio Control System Diagram

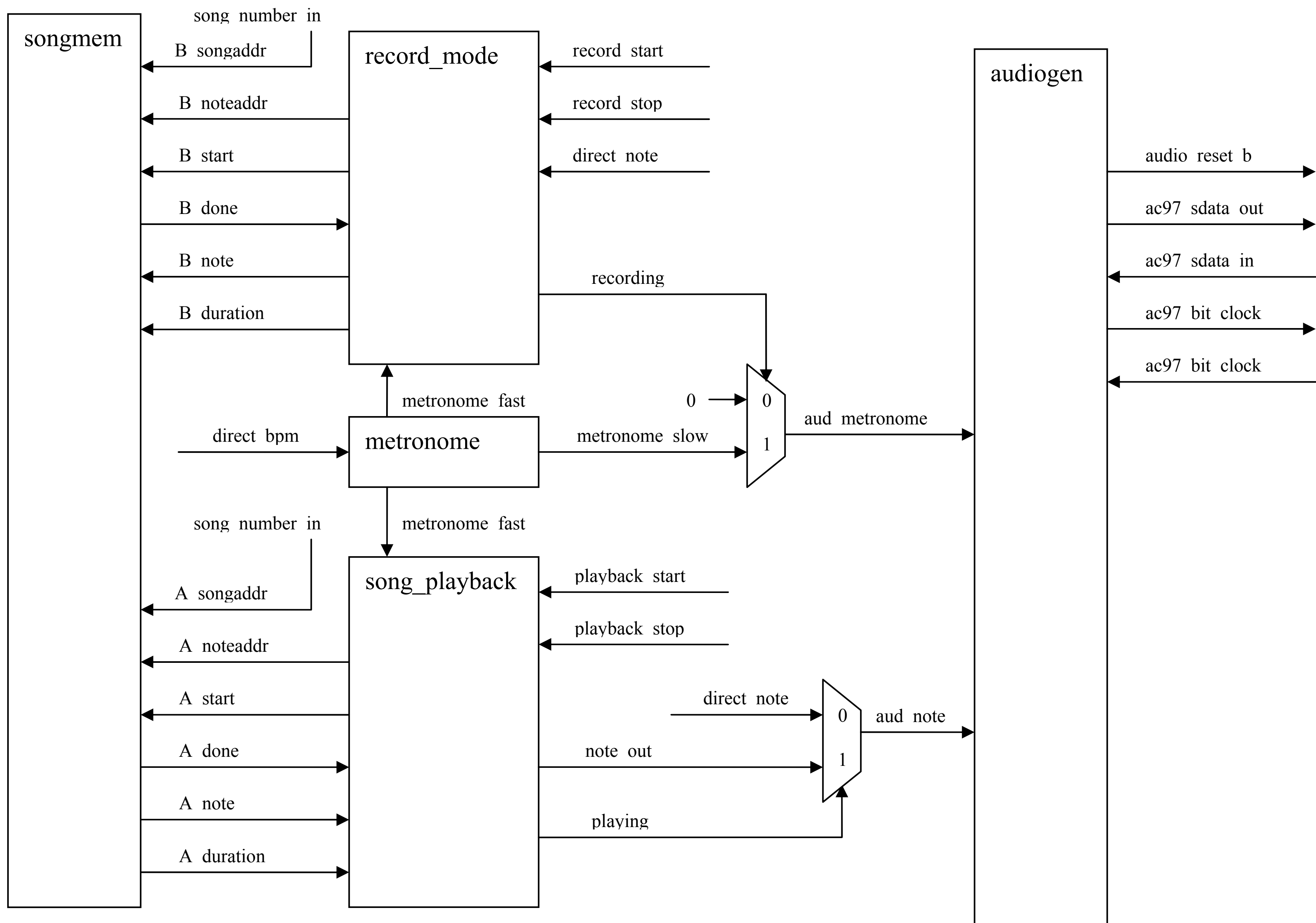


Figure 5.2 Audio Control Module Block Diagram

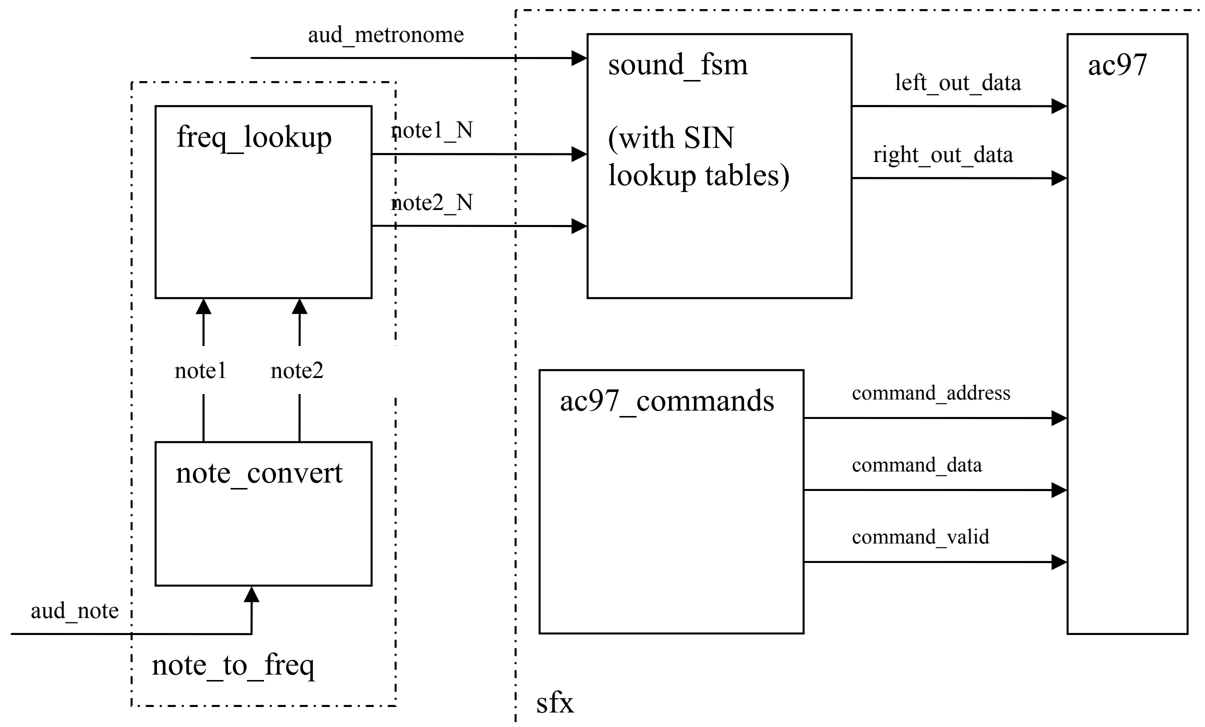


Figure 5.3 Audiogen Block Diagram

b) Note to Frequency Conversion
Audiogen receives a 36 bit note number based on the encoding scheme from the projection output block. There are 36 musical notes in all, from C natural below middle C (C3) to the second B natural above middle C (B5). Each note is represented by a single bit in the note number, which is high when the note is played and low when not. This is the simplest way to make the system expandable to play two or more notes.

The note_to_freq module (left in Figure 3.3) internally converts the 36-bit note number into two 6-bit note numbers representing the active notes, and then converts these again in a frequency look up table. The system matches the notes up to the right 'N,' the count number, calculated by the following formula:

$$N = \frac{f \cdot 2^{16}}{48000}$$

This number represents the number of clock cycles of the 48kHz audio bit clock that must be counted through before sampling the sine table again for any given audio frequency f if we have a sine table with 2^16 entries. In IP CoreGen, we were only able to simulate a sine table with 256 entries, so taking the top 8 bits of the count was sufficient to generate the note. This sampling method led to a ~1Hz resolution on the audio output signal, sufficiently small to go undetected by the human ear.

Table 5.1 documents the note to sound conversion for the 36 available notes.

Note	36-bit Representation	6-bit Representation	f (Hz)	N
REST	36’b0	6'd0	0	0
C 3	36'b000000000000000000000100000000000000	6'd15	130.81	179
C#/Db 3	36'b000010000000000000000000000000000000	6'd32	138.59	189
D 3	36'b000000000000000000001000000000000000	6'd16	146.83	200
D#/Eb 3	36'b000100000000000000000000000000000000	6'd33	155.56	212
E 3	36'b000000000000000000010000000000000000	6'd17	164.81	225
F 3	36'b000000000000000000100000000000000000	6'd18	174.61	238
F#/Gb 3	36'b001000000000000000000000000000000000	6'd34	185.00	253
G 3	36'b000000000000000001000000000000000000	6'd19	196.00	268
G#/Ab 3	36'b010000000000000000000000000000000000	6'd35	207.65	284
A 3	36'b000000000000000010000000000000000000	6'd20	220.00	300
A#/Bb 3	36'b100000000000000000000000000000000000	6'd36	233.08	318
B 3	36'b000000000000000100000000000000000000	6'd21	246.94	337
C 4	36'b000000000000000000000000000010000000	6'd8	261.63	357
C#/Db 4	36'b000000000100000000000000000000000000	6'd27	277.18	378
D 4	36'b000000000000000000000000000100000000	6'd9	293.66	401
D#/Eb 4	36'b000000001000000000000000000000000000	6'd28	311.13	425
E 4	36'b000000000000000000000000001000000000	6'd10	329.63	450
F 4	36'b000000000000000000000000010000000000	6'd11	349.23	477
F#/Gb 4	36'b000000010000000000000000000000000000	6'd29	369.99	505
G 4	36'b000000000000000000000000100000000000	6'd12	392.00	535
G#/Ab 4	36'b000000100000000000000000000000000000	6'd30	415.30	567
A 4	36'b000000000000000000000001000000000000	6'd13	440.00	601
A#/Bb 4	36'b000001000000000000000000000000000000	6'd31	466.16	636
B 4	36'b000000000000000000000010000000000000	6'd14	493.88	674
C 5	36'b000000000000000000000000000000000001	6'd1	523.25	714
C#/Db 5	36'b000000000000001000000000000000000000	6'd22	554.37	757
D 5	36'b000000000000000000000000000000000010	6'd2	587.33	802
D#/Eb 5	36'b000000000000010000000000000000000000	6'd23	622.25	850
E 5	36'b000000000000000000000000000000000100	6'd3	659.26	900
F 5	36'b000000000000000000000000000000001000	6'd4	698.46	954
F#/Gb 5	36'b000000000000100000000000000000000000	6'd24	739.99	1010
G 5	36'b000000000000000000000000000000010000	6'd5	783.99	1070
G#/Ab 5	36'b000000000001000000000000000000000000	6'd25	830.61	1134
A 5	36'b000000000000000000000000000000100000	6'd6	880.00	1201
A#/Bb 5	36'b000000000010000000000000000000000000	6'd26	932.33	1273
B 5	36'b000000000000000000000000000001000000	6'd7	987.77	1349

Table 5.1 Note Conversion Table

2. Metronome
   The metronome module serves two functions. First, it provides a “slow,” audible metronome to audiogen to guide the user when recording songs. Metronome_slow ranges between 10 to 250 beats per minute. The “fast” metronome drives the record and playback modes of the audio control system. It runs at 60 times the speed of the slow module. In the record mode, it determines the temporal resolution of the note durations stored and in the playback mode, it determines how long each memory note is played. Changing the metronome also enable the playback function to playback songs at different tempos than that originally recorded.

   The metronome functions by counting up each clock cycle to a beat_count value. The beat count for metronome_fast is determined by the following formula:

$$beat_count = \frac{clock_frequency}{BPM}$$

To generate the slow metronome, 60 fast beats are counted in between each slow beat.

3. Record Mode
   The record_mode module defines the logic necessary to store a song played by the user. Each note is stored to a memory location defined by a 4-bit song address, allowing the user to store a maximum of sixteen songs, and a 12-bit note address, allowing the user to store 4096 notes per song. Each memory location stores both a 36-bit note number and an 8-bit duration number which specifies the number of metronome counts the note is to be held out for.

   The state transition diagram for the record mode is shown below in Figure 5.4.

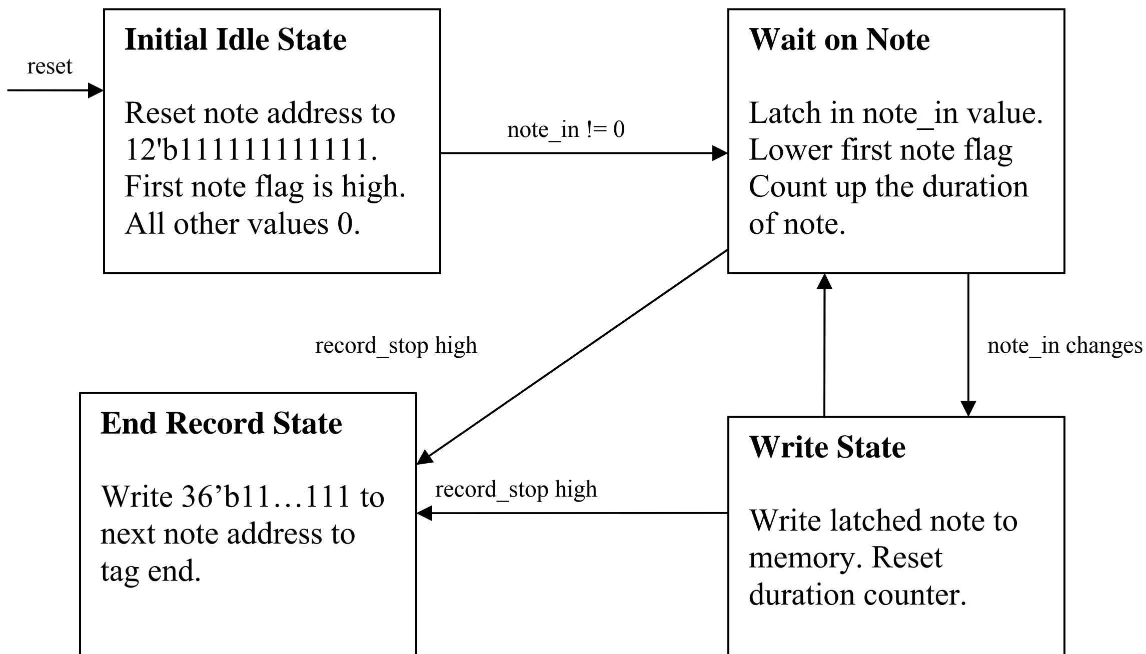


Figure 5.4 Record State Transition Diagram

4. Song Playback Mode
   The song_playback module is the complementary module to the record module. It reads out data in the same format as the songs have been stored and also uses the fast metronome to determine how long each note is held. It sends the note information directly to the audiogen module.

   The state transition diagram for the playback mode is shown below in Figure 5.5.

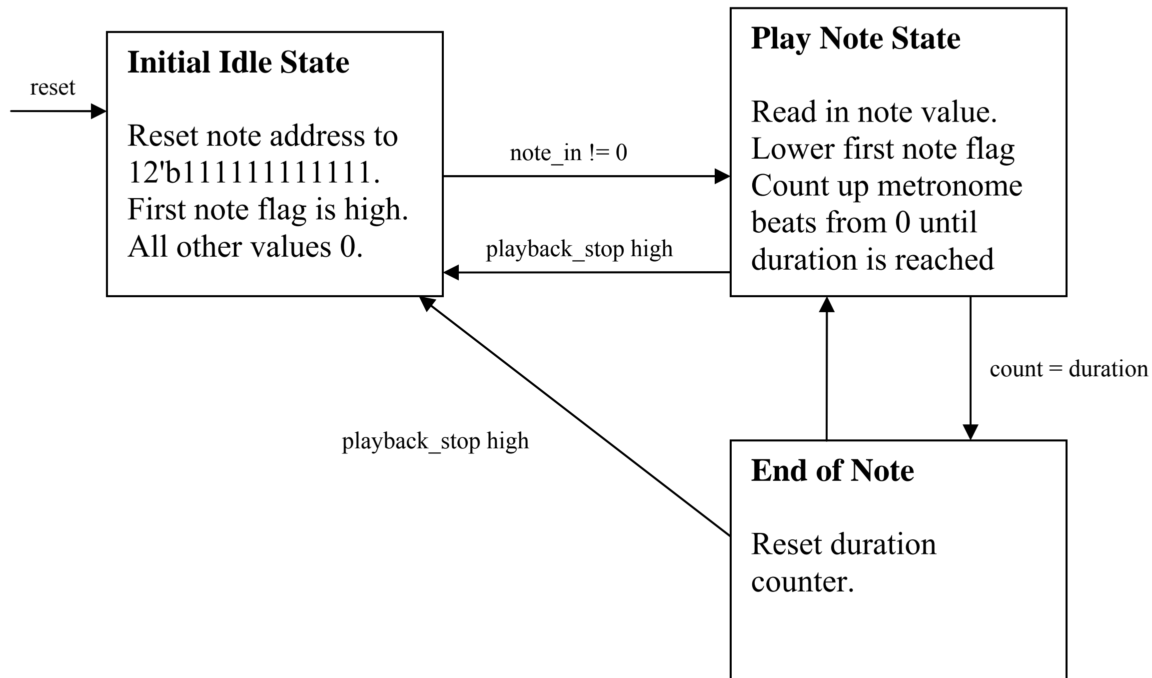


Figure 5.5 Playback State Transition Diagram

C. Testing and Debugging

The audio control system was constructed in two stages. First, we wrote and tested the audiogen block. To test the 36 different frequencies we intended to generate we tied the swtiches on the labkit to a 6-bit note number. From the beginning it had no problem generating notes, using our sine lookup method. The initial sound, however, was extremely noisy. After attempting anumber of potential solutions, such as registering and delaying the sound through multiple clock cycles, we looked at the output sine wave on the logic analyser. A sample screen is shown below in Figure 5.6. We noticed that the sine wave would periodically jump to a different location and begin again-the ready signal from the ac97 turned out to be glitchy. An additional register solved our problem.

We constructed the record and playback modes simultaneously and tested them together. Various switches were designated for song address selection and beat selection, and the buttons were tied to a limited selection of notes. We used the logic analyzer to observe the behaviour of all signals to and from the SRAM, and were able to catch a number of minor bugs before the system was fully functional.

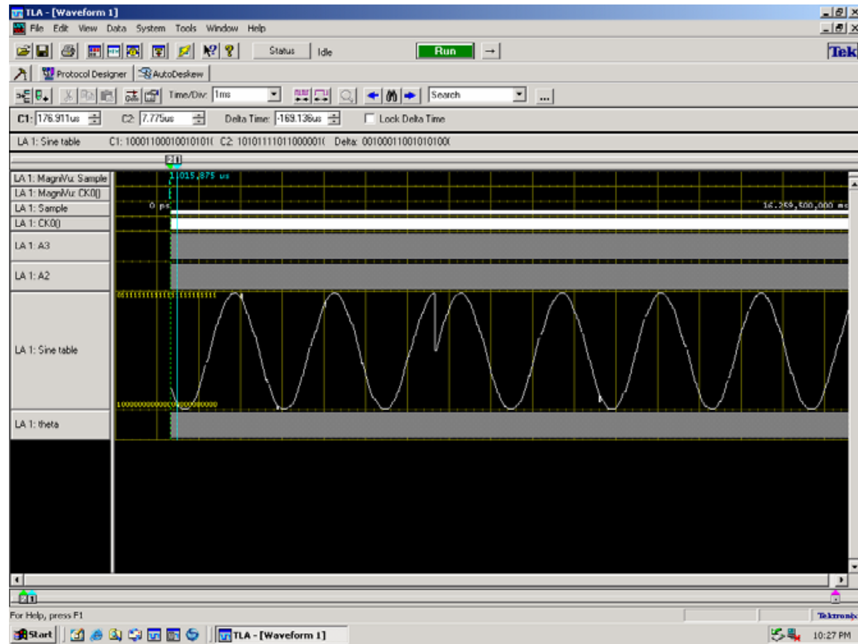


Figure 5.6 Sine Wave Debugging Waveform

VI. Projection Output

A. Overview

The projection output block is responsible for generating the VGA signals to the projectors. It generates the piano keyboards, as well as a series of menus which the user can use to control the system.

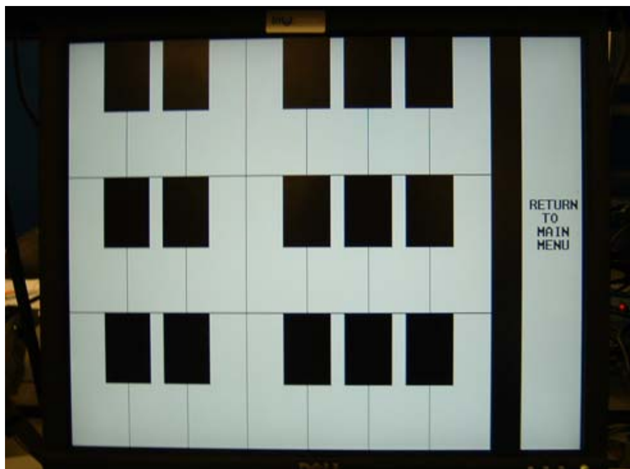
B. Screen Descriptions

There are a total of six screens projected onto the ground. The first screen that users see is the welcome screen, which simply says "welcome to piano dance revolution" on top of a Charlie's Angels background. This screen will last for five seconds when the game first starts or when the user decides to reset the system (shown in Figure 6.1). After five seconds, the display atomically goes to the mode selection screen (shown in Figure 6.2), which includes buttons "play mode", "game mode" and "record mode" that the users could step on to determine which mode he/she wants to go into.

Figure 6.1: Welcome Screen

Figure 6.2: Mode Selection

If the user stepped on the play mode, it will lead him/her directly to the keyboard. As shown in Figure 6.3, the keyboard has three rows, each row is an octave. This is done so user could jump across octaves more easily. Whenever the user steps on a key, it will light up to a different color. To the right of the keyboard, there is a return button, which will bring the user back to the mode selection screen.



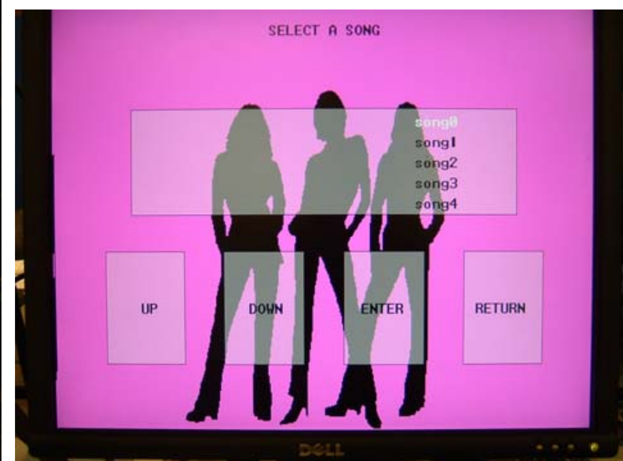


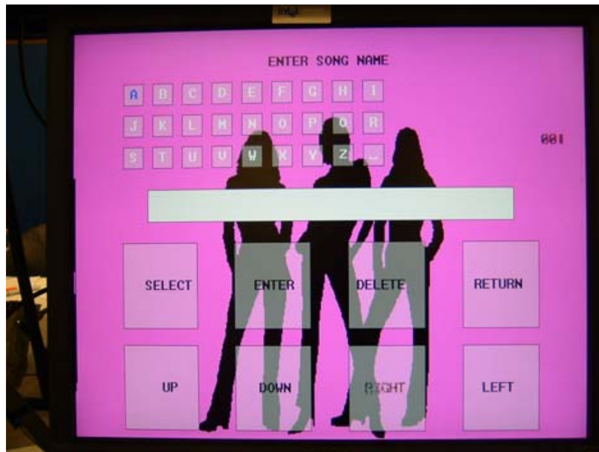
Figure 6.3: Keyboard

Figure 6.4: Song Selection Screen

If the user selected game mode, it will lead him/her to a song selection screen, shown in Figure 6.4. A maximum of 16 songs can be stored into the system for game mode. The up and down buttons will help the user navigate through a list of these previously stored songs. Once a desired song is found, the user could step on the enter button to select the song and go to the beat selection screen. Finally, the return button will bring the user back to the mode selection screen.

If the user selected record mode, the system will lead him/her to the enter song title screen, shown in Figure 6.5. This screen allows the user to enter a title for the song that he/she is about to record. The user uses the up, down, right, left and select buttons to navigate through the mini-keyboard located on the upper left corner of the screen. The user could use also the delete button available to make adjustment to

the title. Once the title is selected, the user could hit enter to store the title and go to the next beat selection screen.



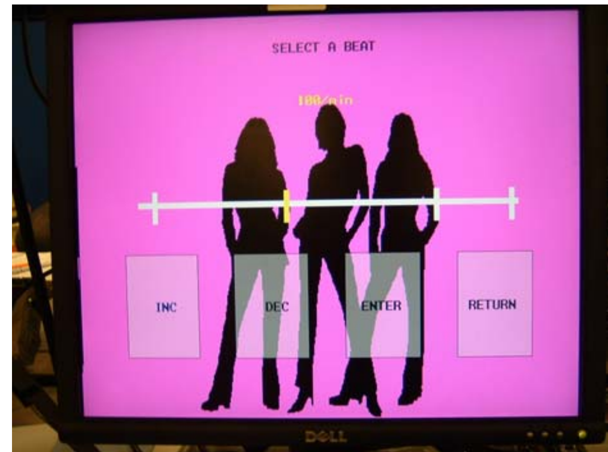


Figure 6.5: Enter Song Title Screen

Figure 6.6: Beat Selection Screen

The beat selection screen allows the user to control the speed of the song, shown in Figure 6.6. The user uses increase and decrease buttons to adjust the beat, and the result will be shown on the beat meter located in the center of the screen, along with the numbered beat value above the meter. Once the desire beat is reached, the user then steps on the enter button and go to the piano keyboard display.

C. Block Descriptions

The project output block is controlled by two major sub-blocks: the projection output control logic block and the projection out display logic block. These two blocks interact with each other, as well as the step interpretation logic block, the SRAM1 interface to generate the correct VGA outputs. A detailed block diagram is shown below in Figure 6.7.

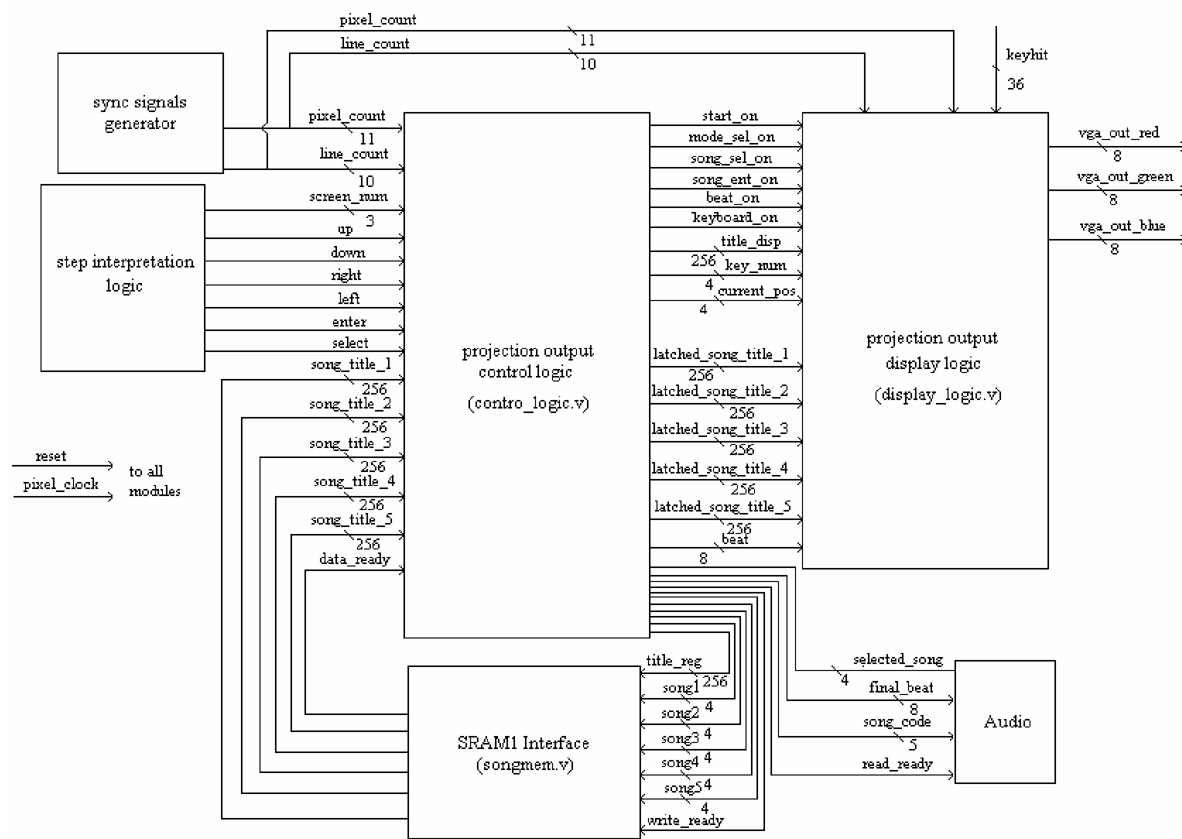


Figure6.7 System Block Diagram of Projection Output Block

As shown above, the projection output control logic is the major FSM that calculates the screen that should be generated and all the necessary data that should be displayed on the screen, and sends them to the projection output display logic, the minor FSM, which outputs the corresponding color values for each pixel. The step interpretation logic block is responsible for detecting with buttons has been stepped on by the user, and sends the corresponding signals the projection output control logic block. It also generates a *screen_num*, which codes for the screen should be on at any given moment.

1. Projection Output Control Logic Block

This block includes four modules: control_logic, song_sel_cal, song_ent_cal, and beat_sel_cal. Following is the detailed description of each of these modules.

a) Control Logic

This is the main module which converts *screen_num* to six flag signals which turn on and off the six display screens. It sends these flag signals to the display logic block, as well as the other three modules in the control logic block.

b) Song_sel_cal

This module is on when the song selection screen is displayed. It interacts with the SRAM1 interface and generates a list of five songs to be displayed onto the song selection screen. When turned on, it sends five song numbers to the SRAM1 and gets back the title of these songs. The five song numbers to SRAM1 will change as the user scroll up and down the song list with the up and down buttons displayed on the screen. The five titles will remain same as previous until a pulsed signal data_ready is

received from the SRAM, indicating that all of the title are stable and could be passed to the display. Also, the *song_sel_cal* keeps track of which of the five songs displayed is being selected by the user and sends *current_pos* to the display unit to highlight that particular song. Again, this position will change as the user moves up and down through the song list.

c) Song_ent_cal
This module is on when the entering song title screen is displayed. It is responsible for generating a number for the song that the user going to record (*song_code)*, keeping track of the letter that the user is currently highlighting (*key_num)*, the title that the user has entered so far (*title_disp)*. These data are sent to the display logic block to be displayed on the entering song title block. The *song_ent_cal* block also writes the complete song title (*title_reg*) to SRAM1 when the user steps on enter, which *song_code* as its corresponding address. The *song_code* is also send to the audio portion of the system so the notes played could be stored under the same address. The system could store a maximum of 16 songs. The last song will be overwritten if the maximum number is reached and the user still decides to store a song. All the recorded song titles will be lost if the user chooses to reset the system.

d) Beat_sel_cal
This module is on when the beat selection screen is displayed. It calculates the current beat speed from the right and left inputs from user, and sends that number to the display logic block. The default beat is 100 beats per minute. The range of the beat is from 10 beats per minutes to 250 beats per minute, and each step on the buttons to adjust the beat by 10. When the user steps on the enter button, the beat number is latched and send to the audio portion of the project (*final_beat*).

2. Projection Output Display Logic
This block is takes all the display data from the projection output control logic block and convert them into VGA signals. This block includes six sub-blocks: open_slide, mode_selection, song_selection, song_enter, beat_selection, and the keyboard. Each of the sub-blocks is responsible for displaying one screen. A detailed block diagram is shown blow in Figure 4.8.

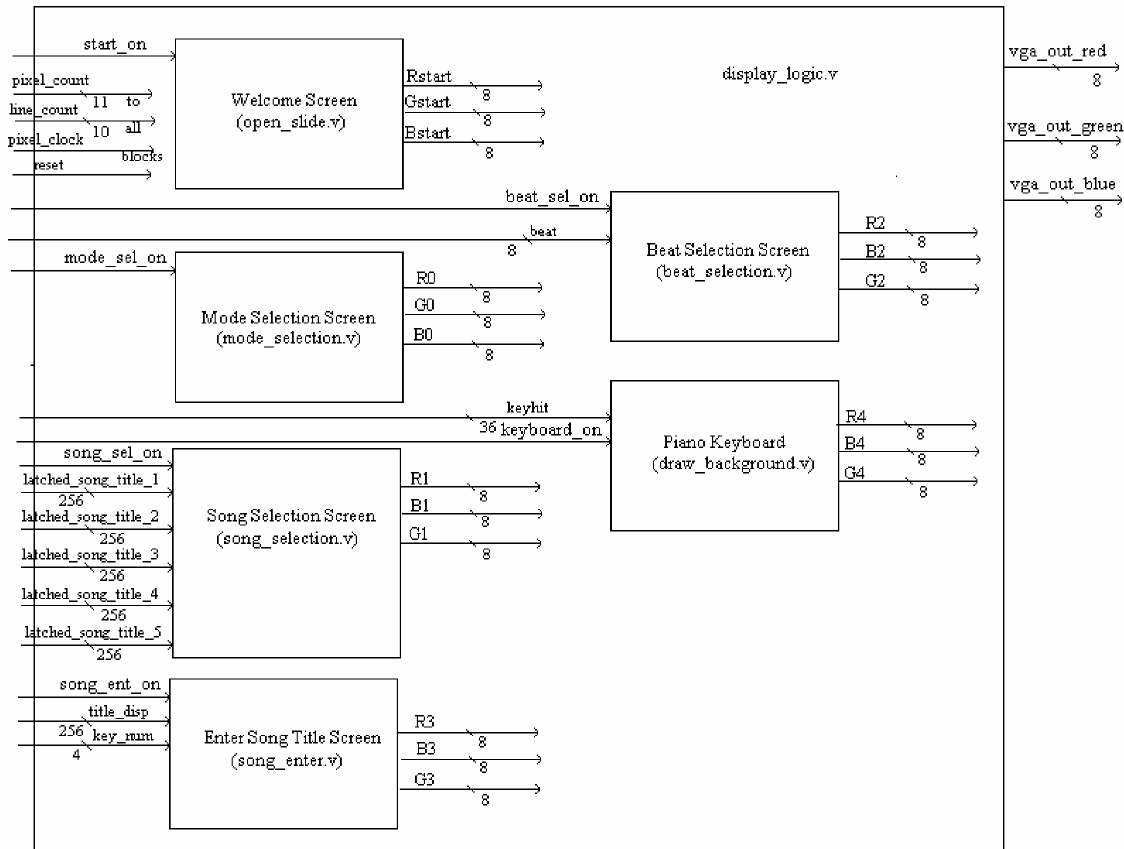


Figure 6.8 System Block Diagram for the Projection Output Display Logic Block

All of the modules mentioned above instantiates one or more of these three modules: the *rectangle* module, the *char_string_display* module, and the *vga_romdisp* module. The details of these modules are explained below.

a) Rectangle

The rectangle module takes in the pixel and line values of the upper left corner and the lower right corner of a rectangle, and outputs *in_rectangle* and *on_border*. For a given pixel, this indicates whether it's inside, on the border, or outside of the rectangle. These two signals are then utilized by other modules to assign the different color values to pixels depending on their position.

b) Char_string_display

This module is found on the Fall 2005 6.111 website. It was written by I. Chuang and C. Terman. It displays an ASCII encoded character string onto the VGA monitor at any pixel and line location. It reads from a font ROM which stores the fonts. Like the rectangle module described earlier, it outputs *cpixel*, which indicate whether a given pixel is within the string of letters. The original coefficient file used to generate the font ROM was modified in this project in an attempt to prettify the displays.

c) VGA_romdisp

This module is also found on the Fall 2005 6.111 website. It was originally written by I. Chuang, and it was modified in this project. The module reads an image from ROM, and outputs the image onto the VGA screen. Due to memory constrictions, the image is stored on ROM is reduced to 2 colors and has size 512 x 386. Therefore, the corresponding ROM has a width of 1 and a depth of 196608. Since the

VGA display is 1024 x 768, 4 pixels are stored and displayed per memory location. The initial coefficient file used to generate the image ROM is created with matlab. Like the other two modules described earlier, it outputs *pixel*, which indicate the color block that a given pixel belongs to.

Putting the above three modules together, each pixel on the screen have four flags or more, which determines the position of the pixel relative to the rectangles, the background figures, and the text displayed on the screen. The color values are then assigned to the pixel accordingly. The six different sub-blocks, along with the display for game mode scoring tracking, all utilize this method to generate different outputs. Then the display logic module uses the six different on signals to determine which screen should be displayed, and connect the VGA color outputs to the corresponding modules.

D. Testing and Debugging

The projection output portion of this project is rather easy to debug since all the outputs are visual. The simplest thing to do is to implement the Verilog code and see whether it works. From the monitor output, we could easily determine whether the modules are implemented right.

However, sometimes it's not always easy to figure out the source of the problem by just looking at the monitor alone. Debugging tools such as oscilloscope are used to sample the internal signals and thereby determining the source of the problems. When we first started the project, we wanted to set the display to be1024 x 768, however, the monitor always indicates that the sync and blanks signals are not correct. After we assigned all the important signals to user ports and tested these signals with oscilloscope, we found that for some reason the clock remained at 27mHz even though we used DCM module to speed it up to 65 mHz. We identified the source of the problem, but we unable to solve it until we realized that the labkit was used sometimes need power on reset to speed up its clock and display at a higher resolution.

VII. Game Mode

A. Overview

In addition to standard play, record and song playback functions, the system also has a game component. In game mode, a song is played back from memory and the keys are lit up in the correct order on the keyboard projection. The user must step on the keys in the correct sequence at the correct time to score points. The user's score is displayed on an additional vga display.

B. Block Descriptions

The game operates by sending the start and stop pulse signals to the song_playback module. It receives simultaneous note inputs from the user's steps and the playback module, and matches them against each other at every pulse of the fast metronome. For each pulse at which the notes match, the user will accumulate one point. The 8-bit score is sent to the score display portion of the system.

The state transition diagram for the game mode is shown below in Figure 7.1.

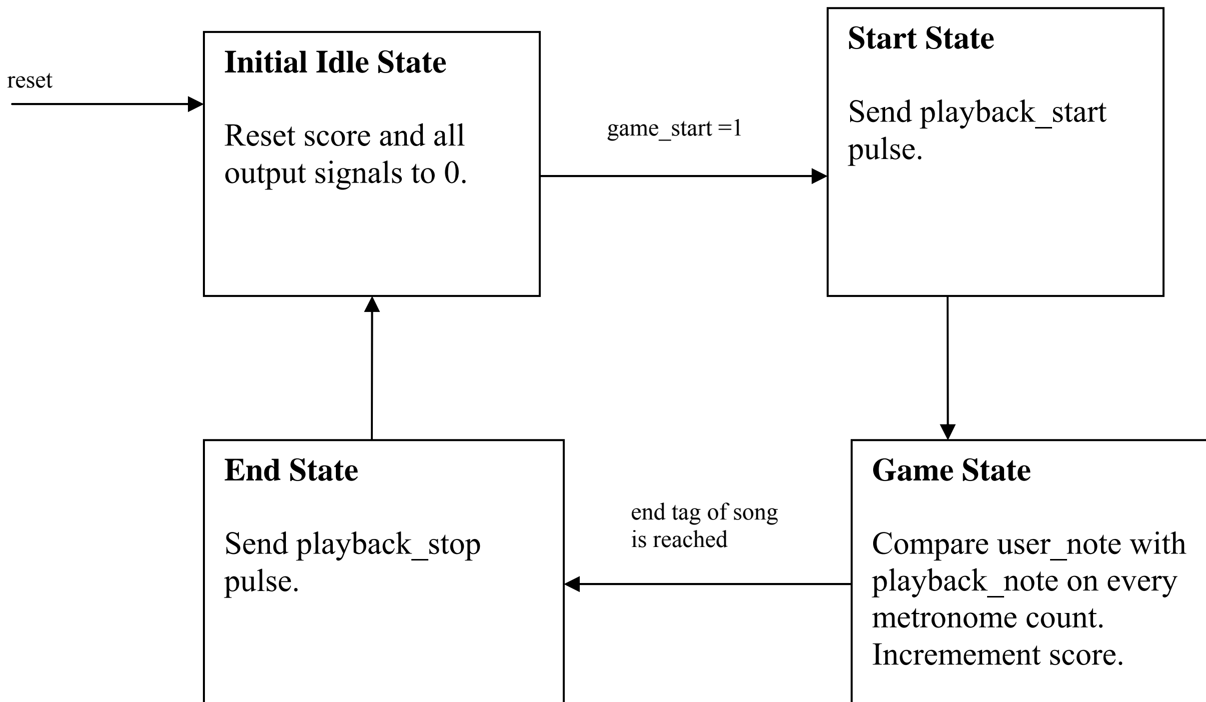


Figure 7.1 Game Mode State Transition Diagram

The game console allows user to see the score, the keys that he/she is stepping on at the moment, and the keys that he/she will have to step next. A picture of this is shown in Figure 7.2.

Figure 7.2: Game Console for Piano Dance Rovolution

The mini-keyboard on the left shows the user's steps in real time, while the keyboard on the right shows the steps that the user needs to step next. Whenever the user steps on a key, the background of the game console will flash purple. The screen is implemented with draw game background block, which implements a series of rectangle modules, char_string_display modules, and the vga_romdisp modules. These modules, along with the method used to generate the game console display, will be described later in the projection output session of the report.

VIII. Interkit Communication

In our project, we have three kits operating in tandem. One kit is designated the "master kit", and the other two are slave kits. The master kit maintains and updates the system state, while the other kits are controlled by the master kit.

Different parts of our project require multiple simultaneous communication lines. For example, both projectors must display the same image, so the master kit must send logic lines indicating what to display on each of the slave kits. The keyboard screen in particular requires 36-bits to determine which keys to light up. Another example is the video data output from Locate2D. The (x,y) coordinates from the camera connected to the slave kit must be transmitted as a continuous stream to the master kit.

In order to be able to transmit all this information back and forth without requiring hundreds of wires between the lab kits, we use a custom made serial communication protocol. The two modules responsible are kitcom_in and kitcom_out.

The kitcom_out module is a parallel to serial converter which accepts a 256 bit input stream, and outputs a continuous one wire serial communication which repeatedly transmits the 256 bit stream at 10 KHz. Each 256-bit transmission is separated by a stop bit (low) and a start bit (high). Resynchronization is achieved by a period of silence (ground) which lasts for over 256 bits. This period occurs every 32 transmissions.

The kitcom_in module is a serial to parallel converter, which converts the serial input (generated by kitcom_out on another labkit) into a 256 bit stream. The input signal is debounced (with a delay of 20 cycles) before being processed. Synchronization with the input signal is achieved by waiting for the silence followed by a high start bit, and then using the labkit's own clock to count the cycles of a 10 KHz clock. In addition, every time the input serial signal has a transition, the cycle count is reset to better synchronize the input signal.

The product of both modules working together is that with four wires between two labkits (one transmit, one receive, and two ground), we can transmit 256 bits of data from one kit to the other, and vice versa, independently.

IX. Conclusion

Piano Dance Revolution attempted the large scale design and implementation of a gaming and entertainment system aimed at fusing the concept of DDR and the FAO Schwartz Piano. This system involved the independent design of four major logic blocks and the subsequent integration of those parts.

The design of each logic block, video input, step interpretation, audio control, and projection output, was completed successfully, and after a substantial debugging process, were independently determined functional as desired. It was then unfortunate that the entire system failed to come together in the end due to a number of regrettable factors. The large scale redesign of the system between proposal and implementation greatly increased the implementation time and manpower. While the original system consisted only of play mode and game mode, and of a single keyboard projection, the final object system involves three modes and an entire system of five projection screens. This system upgrade required the utilization of a great deal of additional memory components and interfacing, along with a few thousand bits of additional information communicating between labkits. Step detection was completely remodeled to reflect control between newly introduced states, and audio control required change to incorporate a recording option. These improvements required a substantially greater amount of time than anticipated by

the designers of the system, and significantly abridged time allotted for final system integration. Additionally, a limited number of gaps in communication during the design of the individual logic blocks provided for some individual block redesigning during the implementation process.

Given an opportunity to repeat the project, significantly greater consideration will be given to time management and project scaling. Perhaps a more advantageous strategy would be to implement a functioning basic system before seeking improvements and upgrades. Also, interblock communication is vital, and all interblock signal and logic requirements should be made very clear at the on start, with modifications made very clear in a timely fashion.

While the final product was not integrated, the design and implementation process of the project was greatly profitable for each of the designers. It was satisfying to pursue an ambitious project and to encounter and resolve the problems along the way. Although the project would be pursued in a different fashion in the future, significant lessons were learned along the way and as an experience the project was an incredible success.