Eun Koo, Gaoxing Ding
1003190254, 1002975569

ECE241 Final Project Report – **Beatmaker**

Introduction

For our project, we created an audio beat player, customized with 8 different samples, which can both record and playback a combination of beats. The motivation for this project was inspired from MIDI-kits, the 808-Rhythm Composer, and other digital drum kits. With a simple interface, one can create a large variety of songs, melodies, depending on which samples they use.

The goals we prepared were:

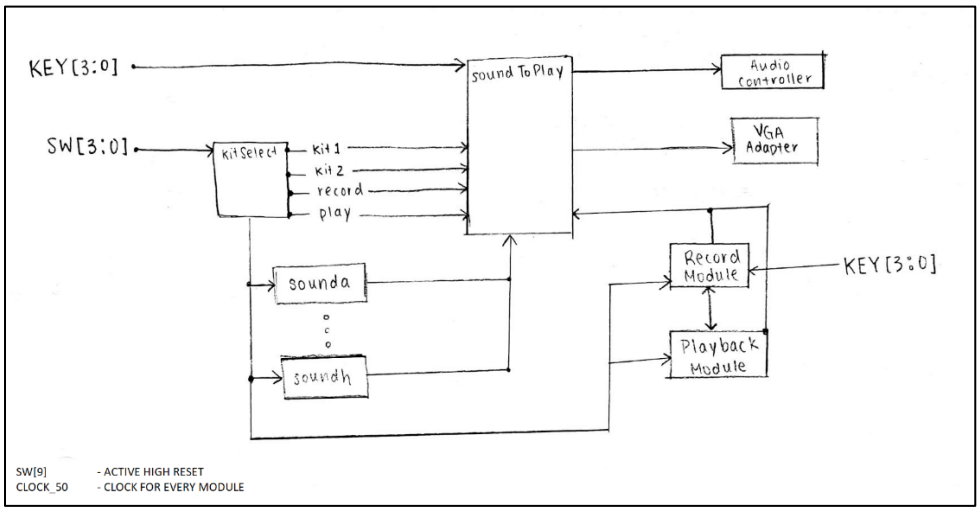
1. Prepare audio samples – have them play on keypress, and recording keypresses
2. Have all audio samples, recording working with all keys and playing back recordings
3. Connect keypresses and playback module to VGA, lighting up images if a sound is playing

During the process, we worked with implementing memory on the DE1-SOC Board on a much larger scale than we had before in our labs. Using the ROMs on the board, we were able to use converted audio files and pictures in our design.

A short demo of the project is available here:
https://drive.google.com/file/d/1JlrNIlYyaDrnijkrWSHl_wOkfAwdN5bG/

The Design

The design for the Beatmaker consisted of multiple modules, shown below in a simplified block diagram:



[Appendix B]

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Description of each component:

- **Top Module** [Appendix C]:
  - Connects all components/modules together
  - Contains wiring connecting soundToPlay Module to the audio controller as well as the VGA Adapter

- **kitSelect Module** [Appendix D]:
  - Multiplexor which determines state of circuit depending on which switches are on:
    - Which kit the user is using for playing sound (kit1, kit2)
    - Whether the user is recording, playing back, or neither

- **Sound Modules** [Appendix E]:
  - Contains ROM with '.mif' raw audio data stored
  - Has counter to send out ROM data when triggered with its respective 'play' signal
  - There are 8 of these modules, each for a different sound sample

- **soundToPlay Module** [Appendix F]:
  - Takes in all keypresses and decoded playback signals
  - Determines which Sound Module to send 'play' signals, to play their respective sound when triggered
  - Outputs are connected to the audio controller directly, which plays the sounds

- **Record Module** [Appendix G]:
  - Records keypresses every tenth of a second, using a counter with the built-in clock
  - Encodes keypresses for each possible sound to store into a 4-bit RAM:
    - 9 different codes for 8 sounds and 1 for no sound
    - Encoding method can be seen in the Appendix code
    - Used for playback of sounds in the same manner
  - Plays metronome click every half-second for timekeeping
  - Has two channel settings for 12 or 24 seconds of recording
  - Connects to HEX displays to show timer (in base 10)

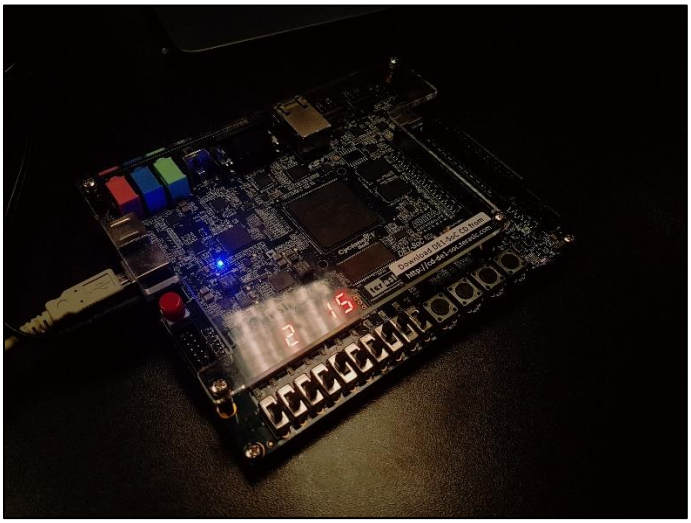
- **Playback Module** [Appendix H]:
  - Has counter which sifts through recorded data and decodes it
  - Sends playback signals for each sound back to soundToPlay Module

- **VGA Modules** [Appendix I] [Appendix J]:
  - Displays an intro screen (160x120) with numbers 1-8 for each sound sample
  - Prints record and playback images
  - Each image will light up when pressed/played/toggled (Did not work in final demo)
  - Outputs signals for position and color to the VGA adapter

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Report on Success

The development of the main components for our project was successful. The main portion of our project worked as intended: a functioning audio beat player, which can play 8 different samples, as well as both record and playback a combination of beats:



Project on Record mode 2nd Channel – Counting to 24 seconds

We were able to finish these parts of the design in the first two weeks, and, as a result, we were able to add an extra milestone for an additional feature to the game: using the VGA for playback and record images, as well as numbers 1-8 which would show the player which settings were toggled, and which sounds were playing [Appendix I] [Appendix J]. Unfortunately, this feature was not fully implemented – mainly due to bugs and time constraints.

The audio components were simple to work with at first, using the audio demo provided. Most of the difficulty for the audio controller was connecting different sounds, converting audio into a raw binary format, which would be plugged into a 1-PORT ROM. The ROMs were read with counters triggered from keypresses or recording data. In addition, we were able to add extra features, including two channels for recording (12 or 24 seconds) and a metronome to play during the recording state [Appendix G].

The main problem that we had during the building of our project was that the features implemented on the VGA could not be completed in time. We were attempting to use a separate FSM and control path to select and change the color of each number printed on the screen, which would indicate when each sound or beat was being played.

Eun Koo, Gaoxing Ding
1003190254, 1002975569

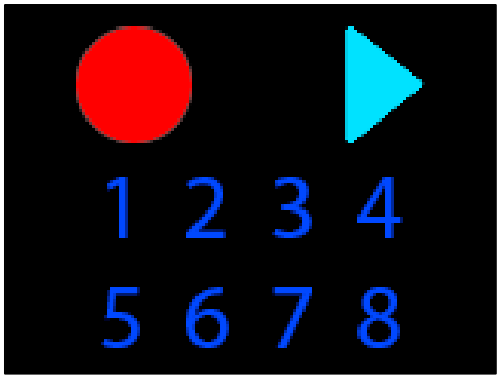


Image of VGA background

When toggling the record or playback switches, the matching image on the screen would have changed color. The same would occur for when the sounds were playing. We speculated that the main bugs were with the handshake signals, which were supposed to be passed between the audio and VGA modules. While we passed in signals for recording, playing back, and each sound, the monitor did not respond – nothing was printed onto the screen. In the end, we felt this was both an issue of time and improper testing, with the complexity of the VGA portion of the design.

What could have been done differently

Reflecting on how we could done this project differently, our team definitely agreed time management and communication was an issue. The work was not split properly, as discussed in the meeting with the TA, which led to delayed testing of much of the VGA portion of the project. One member (Gaoxing Ding) was responsible for implementing the VGA [Appendix I] [Appendix J], while the other (Eun Koo) was responsible for the audio implementation, recording/playback, and, for the third week, extra features, such as the metronome, timer, and recording channels [Appendix C] [Appendix D] [Appendix E] [Appendix F] [Appendix G] [Appendix H].

Our team decided to implement the VGA feature to our project after the second week of development. This was mainly due to us finishing the record and playback features much quicker than expected. If we had planned for a fully integrated VGA with our audio, it could have been possible to add a better user interface, and possibly support for other features, such as adding more sounds with a keyboard, while finishing on time.

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Most of the work for the VGA feature was done by one of our team members, as mentioned above, while the other worked on adding more features for the audio portion. Unfortunately, the coding process during the last week was rushed. Over 500 lines of code (partly recycled from previous lab assignments) were written [Appendix I] [Appendix J]. Initial testing of the code was only done after everything was written. We could have improved on this process by testing the code as we went along, instead of doing it all close to the deadline.

Another possible change was that we felt that if we had decided on our goals at an earlier point in time, this problem could have been avoided. One of the main reasons we did not prepare for this was because the record and playback features of the design were expected to take much longer to develop. Being our first design project in the course, and digital logic in general, there was a lot to learn from a project management perspective which resulted in a rush to complete the project before the deadline. With some more time, we felt could have successfully added animations and moving components onto the screen for the VGA portion. Ultimately, we compromised by displaying an intro screen for the different features of our design (record, playback, 8 sounds).

In addition, an extra feature we wanted to work with was an SD card for storage. This way, we could record the audio and have the data saved onto an external memory, which could be used elsewhere. This would have made our design much more practical and interesting. Unfortunately, our team agreed that using the SD card was going to be too difficult after researching various methods for implementation, many of which required the use of Qsys and coding in languages other than Verilog.

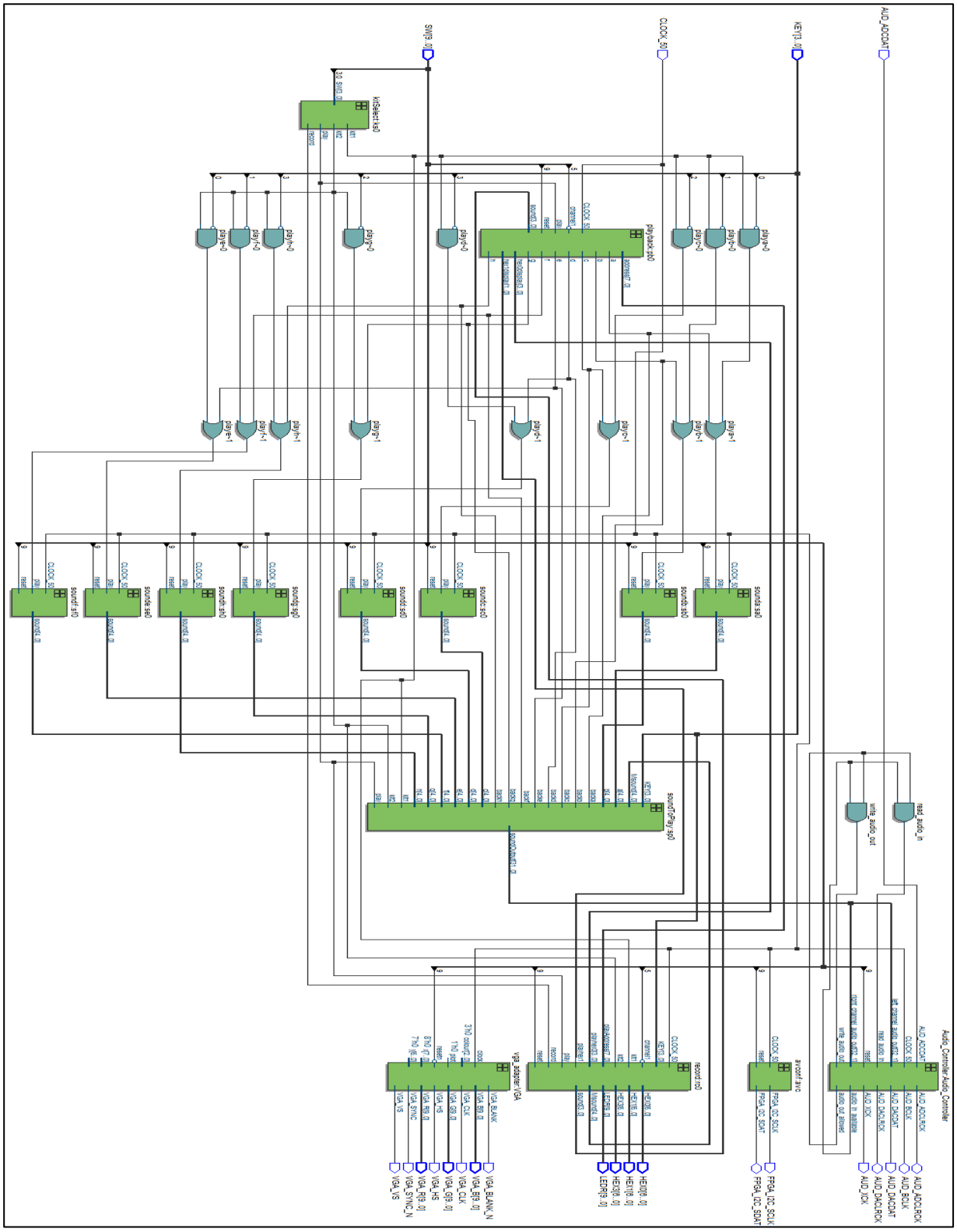
Conclusion

The project successfully demonstrated use of the DE1-SOC Board's memory, audio and VGA controllers with the implementation of a functioning audio beat player, which can play 8 different samples, as well as both recording and playing back a combination of beats. In the process, we learned how to work with memory cells for both the audio and VGA portions, and with abstraction on a large scale, with multiple FSMs, data paths, and multiplexors all intertwined and interacting in the process.

Eun Koo, Gaoxing Ding
1003190254, 1002975569

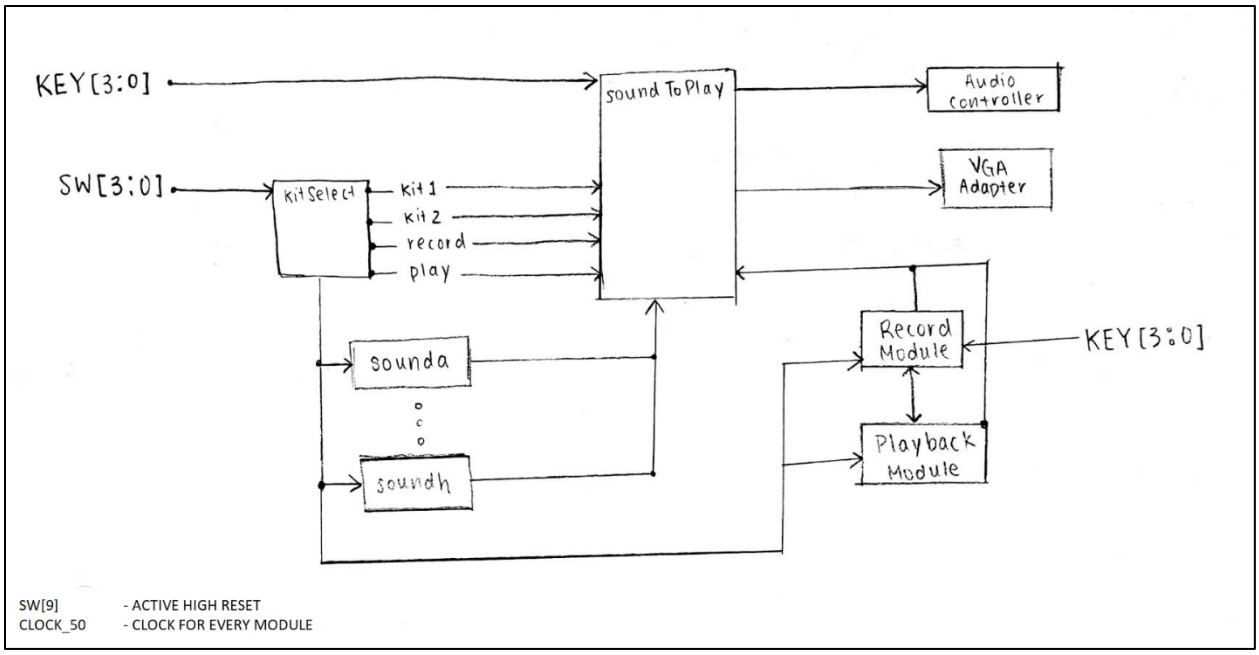
Appendix – Diagrams, Schematics, Code

Appendix A – Quartus Schematic for design



Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix B – Simplified Schematic / Block Diagram



Appendix C – Top Module code (Audio, VGA demo, redundant code was deleted – marked by ***_***)

```
module top (*AUDIO CODE*, *VGA CODE*, ***REST OF INPUTS MARKED DOWN BELOW***);

        ***AUDIO AND VIDEO INPUTS/OUTPUTS***

        // Inputs
        input                                   CLOCK_50;
        input               [3:0]               KEY;
        input               [9:0]               SW;
        output [6:0] HEX0, HEX1, HEX3;
        output [9:0] LEDR;

        // Internal Wires
        wire                                    audio_in_available;
        wire                [31:0]              left_channel_audio_in;
        wire                [31:0]              right_channel_audio_in;
        wire                                    read_audio_in;

        wire                                    audio_out_allowed;
        wire                [31:0]              left_channel_audio_out;
        wire                [31:0]              right_channel_audio_out;
        wire                                    write_audio_out;

        // My inputs and outputs
        wire kit1, kit2, record, play;
        kitSelect ks0(SW, kit1, kit2, record, play);

        // Sound Modules
        wire playa, donea;
        assign playa = ((kit1 & ~KEY[0]) | backa);
        wire [4:0] sounda;
        sounda sa0(CLOCK_50, SW[9], playa, sounda, donea);

        wire playb, doneb;
        assign playb = ((kit1 & ~KEY[1]) | backb);
        wire [4:0] soundb;
        soundb sb0(CLOCK_50, SW[9], playb, soundb, doneb);
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
        ***REDUNDANT CODE***

        wire playh, doneh;
        assign playh = ((kit2 & ~KEY[3]) | backh);
        wire [4:0] soundh;
        soundh sh0(CLOCK_50, SW[9], playh, soundh, doneh);


        wire [31:0] soundOutput;
        soundToPlay sp0(KEY, kit1, kit2, play, backa, backb, backc, backd, backe, backf, backg,
backh, sounda, soundb, soundc, soundd, sounde, soundf, soundg, soundh, Msound, soundOutput);


        // Record Module
        wire [3:0] recSound;
        wire [4:0] Msound;
        record rc0(CLOCK_50, SW[9], KEY, record, play, kit1, kit2, !SW[5], playAddress,
playhex0, playhex1, recSound, HEX0, HEX1, HEX3, LEDR, Msound);


        // Playback Module
        wire [7:0] playAddress;
        wire [3:0] playhex0;
        wire [1:0] playhex1;
        wire backa, backb, backc, backd, backe, backf, backg, backh;
        playback pb0(CLOCK_50, SW[9], play, recSound, !SW[5], backa, backb, backc, backd,
backe, backf, backg, backh, playAddress, playhex0, playhex1);


        // VGA Modules
        ***DID NOT END UP WORKING***
        wire clearSignal;
        wire [7:0] ld;
        wire vgaClear, waiting;
        vgaFSM vf0 (CLOCK_50, SW[9], KEY, kit1, kit2, play, {backa, backb, backc, backd, backe,
backf, backg, backh}, drawDone, clearSignal, ld, vgaClear, waiting);


        wire vgaWrite;
        wire drawDone;
        wire [2:0] vgaOutput;
        wire [7:0] x;
        wire [6:0] y;
        datapath dp0(CLOCK_50, reset, ld, vgaClear, waiting, vgaWrite, vgaOutput, drawDone,
clearSignal, x, y);


        assign read_audio_in                = audio_in_available & audio_out_allowed;

        assign left_channel_audio_out       = soundOutput;
        assign right_channel_audio_out      = soundOutput;
        assign write_audio_out              = audio_in_available & audio_out_allowed;

        ***AUDIO CONTROLLER AND VGA ADAPTER***

endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix D – kitSelect Module code (Redundant code was deleted – marked by ***_***)

```
module kitSelect (SW, kit1, kit2, record, play);
          input [3:0] SW;
          output reg kit1, kit2, record, play;

          // Manages different states for switches - record, play, kit1-2
          always @(*) begin
                    if (SW[0] & !SW[1] & !SW[2] & !SW[3]) begin
                              kit1 = 1;
                              kit2 = 0;
                              record = 0;
                              play = 0;
                    end

                    else if (SW[1] & !SW[0] & !SW[2] & !SW[3]) begin
                              kit1 = 0;
                              kit2 = 1;
                              record = 0;
                              play = 0;
                    end

                    else if (SW[2] & !SW[0] & !SW[1] & !SW[3]) begin
                              kit1 = 0;
                              kit2 = 0;
                              record = 1;
                              play = 0;
                    end

                    ***REDUNDANT CODE***

                    else begin
                              kit1 = 0;
                              kit2 = 0;
                              record = 0;
                              play = 0;
                    end
          end
endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix E – Sound A Module (Same format for all sound modules)

```
module sounda (CLOCK_50, reset, play, sound, done);
	input CLOCK_50;
	input reset;
	input play;
	output [4:0] sound;
	output reg done;

	reg [11:0] counter;
	reg [8:0] address;
	reg start;

	localparam		final = 9'b100111100;

	// Counter for extracting audio data
	always @(posedge CLOCK_50) begin
		if (play)
			start <= 1;

		if (reset) begin
			counter <= 0;
			address <= 0;
			done <= 0;
			start <= 0;
		end

		else if (done & start) begin
			counter <= 0;
			address <= 0;
			done <= 0;
			start <= 0;
		end

		else if (counter == 12'b100000100100 & start) begin
			counter <= 0;

			if (address != final)
				address <= address + 1;

			else
				done <= 1;

		end

		else if (start)
			counter <= counter + 1;

	end

	a	a_inst (
	.address ( address ),
	.clock ( CLOCK_50 ),
	.q ( sound )	// Add 20 - 23 0s at the end
	);

endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix F – soundToPlay Module code

```
module soundToPlay (KEY, kit1, kit2, play, backa, backb, backc, backd, backe, backf, backg,
backh, a, b, c, d, e, f, g, h, Msound, soundOutput);

          input [3:0] KEY;
          input kit1, kit2, play;
          input backa, backb, backc, backd, backe, backf, backg, backh;
          input [4:0] a, b, c, d, e, f, g, h, Msound;
          output [31:0] soundOutput;

          reg [4:0] sounda;
          reg [4:0] soundb;

          always @(!play) begin
                    case (~KEY)
                    4'b0000: sounda = Msound;
                    4'b0001: sounda = kit1 ? a : kit2 ? e : 0;
                    4'b0010: sounda = kit1 ? b : kit2 ? f : 0;
                    4'b0100: sounda = kit1 ? c : kit2 ? g : 0;
                    4'b1000: sounda = kit1 ? d : kit2 ? h : 0;
                    default: sounda = 0;
                    endcase
          end

          always @(play) begin
                    if (backa)
                              soundb = a;

                    else if (backb)
                              soundb = b;

                    else if (backc)
                              soundb = c;

                    else if (backd)
                              soundb = d;

                    else if (backe)
                              soundb = e;

                    else if (backf)
                              soundb = f;

                    else if (backg)
                              soundb = g;

                    else if (backh)
                              soundb = h;

                    else
                              soundb = 0;
          end

          assign soundOutput = {play ? soundb : sounda, 21'b0};
endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix G – Record Module code

```
module record (***INPUTS WHICH ARE EITHER REDUNDANT/LISTED BELOW***);

          input record, play, kit1, kit2;

          // Two recording channels determined by this input
          input channel1;
          input [7:0] playAddress;
          input [3:0] playhex0;
          input playhex1;

          output [3:0] sound;
          output [6:0] HEX0, HEX1, HEX3;
          output [9:0] LEDR;

          // Hold sound data to be recorded
          reg [3:0] keySound;
          reg [3:0] recKey;
          reg recDone;

          // RateDividing / Counter
          reg [22:0] counter;                    // Counts to 10011000100101101000000 = 5 mill.
          reg [7:0] address;

          // For Metronome feature
          reg [2:0] Mcounting;
          reg Mplay;
          output [4:0] Msound;

          // Timer for time keeping in base10
          reg [3:0] hex0timer;
          reg [3:0] hex0display;
          reg [1:0] hex1display;

          // Counts to 12 or 24
          localparam final = channel1 ? 8'b01111000 : 8'b11110000, Mfinal = 12'b110111001000;

          // Assigns 4-bit for every sound
          always @(*) begin
                    case (~KEY)
                              4'b0001: keySound = kit1 ? 4'b0001 : 4'b1001;
                              4'b0010: keySound = kit1 ? 4'b0010 : 4'b1010;
                              4'b0100: keySound = kit1 ? 4'b0100 : 4'b1100;
                              4'b1000: keySound = kit1 ? 4'b0111 : 4'b1111;
                    endcase
          end

          always @(posedge CLOCK_50) begin

                    if (reset) begin
                              counter <= 0;
                              address <= 0;
                              recDone <= 0;
                              recKey <= 0;
                              hex0timer <= 0;
                              hex0display <= 0;
                              hex1display <= 0;
                              Mcounting <= 0;
                              Mplay <= 0;
                    end

                    else if (record & !recDone) begin

                              // 0.1 second
                              if (counter == 23'b10011000100101101000000) begin
                                        counter <= 0;
                                        hex0timer <= hex0timer + 1;
                                        Mcounting <= Mcounting + 1;

                                        if (Mplay)
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                                        Mplay <= 0;

                                else if (Mcounting == 3'b101) begin
                                        Mcounting <= 0;
                                        Mplay <= 1;
                                end


                                if (address == (channel1 ? 8'b01111000 : 8'b11110000))
                                        recDone <= 1;

                                else
                                        address <= address + 1;

                                recKey <= 0;
                        end

                        else begin
                                counter <= counter + 1;

                                if (~KEY)
                                        recKey <= keySound;
                        end

                        if (hex0timer == 4'b1010) begin
                                hex0timer <= 0;
                                hex0display <= hex0display + 1;
                        end

                        if (hex0display == 4'b1010) begin
                                hex0display <= 0;
                                hex1display <= hex1display + 1;
                        end

                end
        end

        // Metronome Component
        reg Mstart, Mdone;
        reg [11:0] Mcounter;
        reg [11:0] Maddress;

        always @(posedge CLOCK_50) begin
                if (Mplay)
                        Mstart <= 1;

                if (reset) begin
                        Mcounter <= 0;
                        Maddress <= 0;
                        Mdone <= 0;
                        Mstart <= 0;
                end

                else if (Mdone & Mstart & record) begin
                        Mcounter <= 0;
                        Maddress <= 0;
                        Mdone <= 0;
                        Mstart <= 0;
                end

                else if (Mcounter == 12'b100000100100 & Mstart & record) begin
                        Mcounter <= 0;

                        if (Maddress != Mfinal)
                                Maddress <= Maddress + 1;

                        else
                                Mdone <= 1;

                end
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                    else if (Mstart & record)
                              Mcounter <= Mcounter + 1;

          end



          // Memory module with 128 words x 1-bit
          RecordMemory       recordmem (
                    // Make this use the same address counter - repeitive code
                    .address (record ? address : playAddress),
                    .clock (CLOCK_50),
                    .data (record ? recKey : 0),
                    .wren (record ? 1 : 0),
                    .q (sound)
          );

          metronome mn0 (
                    .address (Maddress),
                    .clock (CLOCK_50),
                    .q (Msound)
          );


          // Display for timekeeping while recording
          sevensegment hex0(record ? hex0display : playhex0, HEX0);
          sevensegment hex1(record ? {2'b00, hex1display} : {2'b00, playhex1}, HEX1);
          sevensegment hex3(channel1? 4'b0001 : 4'b0010, HEX3);

          assign LEDR[3:0] = sound;
          assign LEDR[7:4] = recKey;
          assign LEDR[9] = recDone;
endmodule


// HEX Display
module sevensegment(INPUTS,OUTPUTS);
          input [3:0] INPUTS;
          output [6:0] OUTPUTS;
          wire w, x, y, z;

          assign w = INPUTS[3];
          assign x = INPUTS[2];
          assign y = INPUTS[1];
          assign z = INPUTS[0];

          assign OUTPUTS[0] = (~w&~x&~y&z)|(~w&x&~y&~z)|(w&~x&y&z)|(w&x&~y&z);
          assign OUTPUTS[1] =
(~w&x&~y&z)|(~w&x&y&~z)|(w&~x&y&z)|(w&x&~y&~z)|(w&x&y&~z)|(w&x&y&z);
          assign OUTPUTS[2] = (~w&~x&y&~z)|(w&x&~y&~z)|(w&x&y&~z)|(w&x&y&z);
          assign OUTPUTS[3] = (~w&~x&~y&z)|(~w&x&~y&~z)|(~w&x&y&z)|(w&~x&y&~z)|(w&x&y&z);
          assign OUTPUTS[4] =
(~w&~x&~y&z)|(~w&~x&y&z)|(~w&x&~y&~z)|(~w&x&~y&z)|(~w&x&y&z)|(w&~x&~y&z);
          assign OUTPUTS[5] = (~w&~x&~y&z)|(~w&~x&y&~z)|(~w&~x&y&z)|(~w&x&y&z)|(w&x&~y&z);
          assign OUTPUTS[6] = (~w&~x&~y&~z)|(~w&~x&~y&z)|(~w&x&y&z)|(w&x&~y&~z);

endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix H – Playback Module (encoding keypresses/audio data)

```
module playback (CLOCK_50, reset, play, sound, channel1, a, b, c, d, e, f, g, h, address,
hex0display, hex1display);

	input CLOCK_50;
	input reset;
	input play;
	input [3:0] sound;
	input channel1;

	output reg a, b, c, d, e, f, g, h;
	output reg [7:0] address;

	// Counter components
	reg [22:0] counter;
	reg done;

	// Timer for time keeping in base10
	reg [3:0] hex0timer;
	output reg [3:0] hex0display;
	output reg [1:0] hex1display;

	localparam final = channel1 ? 8'b01111000 : 8'b11110000; // counts to 12 or 24 seconds

	always @(posedge CLOCK_50) begin

		if (reset) begin
			counter <= 0;
			address <= 0;
			done <= 0;
			hex0timer <= 0;
			hex0display <= 0;
			hex1display <= 0;
		end

		else if (done) begin
			counter <= 0;
			address <= 0;
			done <= 0;
			hex0timer <= 0;
			hex0display <= 0;
			hex1display <= 0;
		end

		else if (play & !done) begin

			// 0.1 second
			if (counter == 23'b10011000100101101000000) begin
				counter <= 0;
				hex0timer <= hex0timer + 1;

				if (address == (channel1 ? 8'b01111000 : 8'b11110000))
					done <= 1;

				else
					address <= address + 1;
			end

			else
				counter <= counter + 1;

			if (hex0timer == 4'b1010) begin
				hex0timer <= 0;
				hex0display <= hex0display + 1;
			end

			if (hex0display == 4'b1010) begin
				hex0display <= 0;
				hex1display <= hex1display + 1;
			end
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                end
        end

        // Decoding recorded data
        always @(play) begin
                case (sound)
                        4'b0001: begin
                                 a = 1;
                                 {b, c, d, e, f, g, h} = 7'b0;
                        end

                        4'b0010: begin
                                 b = 1;
                                 {a, c, d, e, f, g, h} = 7'b0;
                        end

                        4'b0100: begin
                                 c = 1;
                                 {a, b, d, e, f, g, h} = 7'b0;
                        end

                        4'b0111: begin
                                 d = 1;
                                 {a, b, c, e, f, g, h} = 7'b0;
                        end

                        4'b1001: begin
                                 e = 1;
                                 {a, b, c, d, f, g, h} = 7'b0;
                        end

                        4'b1010: begin
                                 f = 1;
                                 {a, b, c, d, e, g, h} = 7'b0;
                        end

                        4'b1100: begin
                                 g = 1;
                                 {a, b, c, d, e, f, h} = 7'b0;
                        end

                        4'b1111: begin
                                 h = 1;
                                 {a, b, c, d, e, f, g} = 7'b0;
                        end

                        default: {a, b, c, d, e, f, g, h} = 8'b0;
                endcase
        end


endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix I – VGA Control Module (FSM) code (Redundant code was deleted – marked by ***_***)

```
module control(sound, kit1, kit2, KEY, resetn, clk, clear, donedraw, record, playback, go, hold,
sinitial, sclear, ld, ld_record, ld_playback);

	***INPUTS AND OUTPUTS SHOWN ABOVE***

	output reg sinitial;
	output reg sclear;
	output reg [7:0] ld;
	output reg ld_record;
	output reg ld_playback;
	output reg hold;


	reg [1:0] current, next;

	localparam		A= 2'b00,
				B= 2'b01,
				C= 2'b10,
				D = 2'b11;

	// State Table

	always@(*) begin
		case (current)
				A: begin
					if (!go)
						next = A;
					else
						next = B;
				end
				B: begin
					if (go) begin
						if (donedraw)
							next = C;
						else
							next = B;
					end

					else
						next = D;

				end

				C: begin
					if(!donedraw)
						next = C;
					else
						next = D;
				end
				D: begin
					if(clear)
						next = D;
					else
						next = A;
				end

				default: next = A;
		endcase
	end
	// Datapath controls
	always@(*) begin
		// Setting all signals to 0 at first
		sinitial = 1'b0;
		ld = 8'b00000000;
		ld_record = 1'b0;
		ld_playback = 1'b0;
		sclear = 1'b0;
		hold = 1'b0;
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                    case (current)
                              A: begin
                                        sinitial = 1'b1;
                              end

                              B: begin
                              if (kit1  && ~(KEY[0])) begin
                                        ld[0] = 8'b00000001;
                              end
                              else if (kit1 && ~(KEY[1])) begin
                                        ld[1] = 8'b00000010;
                              end

                              ***REDUNDANT CODE***

                              else if (kit2 && ~(KEY[3])) begin
                                        ld[7] = 8'b10000000;
                              end
                              else if (record) begin
                                        ld_record = 1'b1;
                                        if (kit1  && ~(KEY[0])) begin
                                                  ld[1] = 8'b00000001;
                                        end
                                        else if (kit1 && ~(KEY[1])) begin
                                                  ld[2] = 8'b00000010;
                                        end

                                        ***REDUNDANT CODE***

                                        else if (kit2 && ~(KEY[3])) begin
                                                  ld[8] = 8'b10000000;
                                        end

                              end
                              else if (playback) begin
                                        ld_playback = 1'b1;
                                        if (sound == 4'b0001) begin
                                                  ld[1] = 8'b00000001;
                                        end
                                        else if (sound == 4'b0010) begin
                                                  ld[2] = 8'b00000010;
                                        end

                                        ***REDUNDANT CODE***

                                        else if (sound == 4'b1111) begin
                                                  ld[8] = 8'b10000000;
                                        end

                              end
                              end
                              C: begin
                                        hold = 1'b1;
                              end

                              D: begin
                                        sclear = 1'b1;
                              end

                    endcase
          end

          always@(posedge clk) begin
                    if (!resetn_image)
                              current <= A;
                    else
                              current <= next;
          end
endmodule
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

Appendix J – VGA Data Path Module code (Redundant code was deleted – marked by ***_***)

```
module datapath(CLOCK_50, resetn_image, INITIAL, clear, donedraw, load, hold, load_playback,
load_record, x_out, y_out);

	***INPUTS LISTED ABOVE***

	input load_playback;
	output reg donedraw;
	input load_record;
	output reg x_out;
	output reg y_out;


	wire [7:0] x_initial1 = 8'b00000000;
	wire [5:0] y_initial1 = 6'b110000;
	reg [5:0] xcounter1;
	reg [6:0] ycounter1;
	wire [2:0]color1;
	reg [11:0] coordinates1;

	wire [6:0] x_initial2 = 7'b0000000;
	wire [5:0] y_initial2 = 6'b110011;
	reg [6:0] xcounter2;
	reg [6:0] ycounter2;
	wire  [2:0]color2;
	reg [12:0] coordinates2;

	***REDUNDANT CODE***

	wire [6:0] x_initial_playback = 7'b1011101;
	wire [6:0] y_initial_playback = 7'b0000000;
	reg [6:0] xcounter_playback;
	reg [5:0] ycounter_playback;
	wire [2:0]color_playback;
	reg [11:0] coordinates_playback;

	wire [6:0] x_initial_record = 7'b0000000;
	wire [6:0] y_initial_record = 7'b1001000;
	reg [6:0] xcounter_record;
	reg [5:0] ycounter_record;
	wire [2:0]color_record;
	reg [11:0] coordinates_record;

	reg [7:0] background_xcounter = 8'b00000000;
	reg [6:0] background_ycounter = 7'b0000000;
	wire [2:0] color_background;
	reg [14:0] coordinates_background;

	mem1 m1(
		.address(coordinates1),
		.clock(CLOCK_50),
		.data(),
		.wren(1),
		.q(color1)
		);
	mem2 m2(
		.address(coordinates2),
		.clock(CLOCK_50),
		.data(),
		.wren(1),
		.q(color2)
		);

	***REDUNDANT CODE***

	mem_record mr(
		.address(coordinates_record),
		.clock(CLOCK_50),
		.data(),
		.wren(1),
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                .q(color1)
                );
        mem_playback mp(
                .address(coordinates_playback),
                .clock(CLOCK_50),
                .data(),
                .wren(1),
                .q(color_playback)
                );
        mem_background mb(
                .address(coordinates_background),
                .clock(CLOCK_50),
                .data(),
                .wren(1),
                .q(color_background)
                );

        always @ (posedge CLOCK_50) begin

                if (!resetn_image) begin
                        xcounter1 <= 0;
                        ycounter1<= 0;
                        xcounter2 <= 0;

                        ***REDUNDANT CODE***

                        ycounter_record<= 0;
                        background_xcounter <= 0;
                        background_ycounter <=0;
                        donedraw <= 0;
                end

                else begin
                        if (INITIAL) begin
                                donedraw <= 0;
                        end

                        if ( load || load_playback || load_record ) begin
                                if (donedraw && (load[0] == 8'b00000001)) begin
                                        donedraw <= 1'b0;
                                end
                                else if (!donedraw) begin
                                        xcounter1 <= xcounter1 + 1;
                                        if (xcounter1 >= 6'b110101) begin
                                                xcounter1 <= 0;
                                                ycounter1 <= ycounter1 + 1;
                                        end

                                        if (ycounter1 >= 7'b1000111) begin
                                                coordinates1 <= {xcounter1 + ycounter1};
                                                donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial1 + xcounter1;
                                        y_out <= y_initial1 + ycounter1;
                                        end


                                //if (done)//donedraw == 1'b1;


                                if (donedraw && (load[1] == 8'b00000010)) begin
                                        donedraw <= 1'b0;
                                end
                                else if (!donedraw) begin
                                        xcounter2 <= xcounter2 + 1;
                                        if (xcounter2 >= 7'b1010001) begin
                                                xcounter2 <= 0;
                                                ycounter2 <= ycounter2 + 1;
                                        end
```

```
                                        if (ycounter2 >= 7'b1000100) begin
                                        coordinates2 <= {xcounter2 + ycounter2};
                                        donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial2 + xcounter2;
                                        y_out <= y_initial2 + ycounter2;
                                        end

                                if (donedraw && (load[2] == 8'b00000100))begin
                                        donedraw <= 1'b0;
                                end
                                else  if (!donedraw)begin
                                        xcounter3 <= xcounter3 + 1;

                                        if (xcounter3 >= 7'b1001111) begin
                                        xcounter3 <= 0;
                                        ycounter3 <= ycounter3 + 1;
                                        end

                                        if (ycounter3 >= 7'b1000100) begin
                                        coordinates3 <= {xcounter3 + ycounter3};
                                        donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial3 + xcounter3;
                                        y_out <= y_initial3 + ycounter3;
                                        end

                                ***REDUNDANT CODE***

                                if (donedraw && (load[7] == 8'b10000000))begin
                                        donedraw <= 1'b0;
                                end
                                else if (!donedraw) begin
                                        xcounter8 <= xcounter8 + 1;

                                        if (xcounter8 >= 6'b110100) begin

                                        xcounter8 <= 0;
                                        ycounter8 <= ycounter8 + 1;
                                        end
                                        if (ycounter8 >= 6'b100100) begin
                                        coordinates8 <= {xcounter8 + ycounter8};
                                        donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial8 + xcounter8;
                                        y_out <= y_initial8 + ycounter8;
                                        end


                                if (donedraw && load_playback == 1'b1) begin
                                        donedraw <= 1'b0;
                                end
                                else if (!donedraw)begin
                                        xcounter_playback<= xcounter_playback + 1;

                                        if (xcounter_playback >= 7'b1000010) begin

                                        xcounter_playback <= 0;
                                        ycounter_playback <= ycounter8+ 1;
                                        end

                                        if (ycounter_playback >= 6'b110010) begin
                                        coordinates_playback <= {xcounter_playback +
ycounter_playback};
```

Eun Koo, Gaoxing Ding
1003190254, 1002975569

```
                                        donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial_playback + xcounter_playback;
                                        y_out <= y_initial_playback + ycounter_playback;
                                        end


                                if (donedraw && load_record == 1'b1) begin
                                        donedraw <= 1'b0;
                                end
                                else  if (!donedraw) begin
                                        xcounter_record<= xcounter_record + 1;

                                        if (xcounter_record >= 7'b1000001) begin
                                        xcounter_record <= 0;
                                        ycounter_record <= ycounter_record + 1;
                                        end
                                        if (ycounter_record >= 6'b101111) begin
                                        coordinates_record <= {xcounter_record +
ycounter_record};
                                        donedraw <= 1'b1;
                                        end
                                end
                                        if (!donedraw) begin
                                        x_out <= x_initial_record + xcounter_record;
                                        y_out <= y_initial_record + ycounter_record;
                                        end
                        end



                        if (hold) begin
                                donedraw <= 0;

                        end

                        if (clear) begin
                                background_xcounter <= background_xcounter +1;
                                if (background_xcounter >= 8'b10100000) begin
                                        background_xcounter <= 0;
                                        background_ycounter <= background_ycounter +1;
                                end
                                if(background_ycounter >= 7'b1111000) begin
                                        coordinates_background <= {background_xcounter +
background_ycounter};
                                end
                                        x_out <= background_xcounter;
                                        y_out <= background_ycounter;
                        end
                end

                if (donedraw) begin
                        xcounter1 <= 0;
                        ycounter1<= 0;
                        xcounter2 <= 0;

                        ***REDUNDANT CODE***

                        ycounter_playback<= 0;
                        xcounter_record <= 0;
                        ycounter_record<= 0;
                        background_xcounter <= 0;
                        background_ycounter <=0;

                end

end
endmodule
```