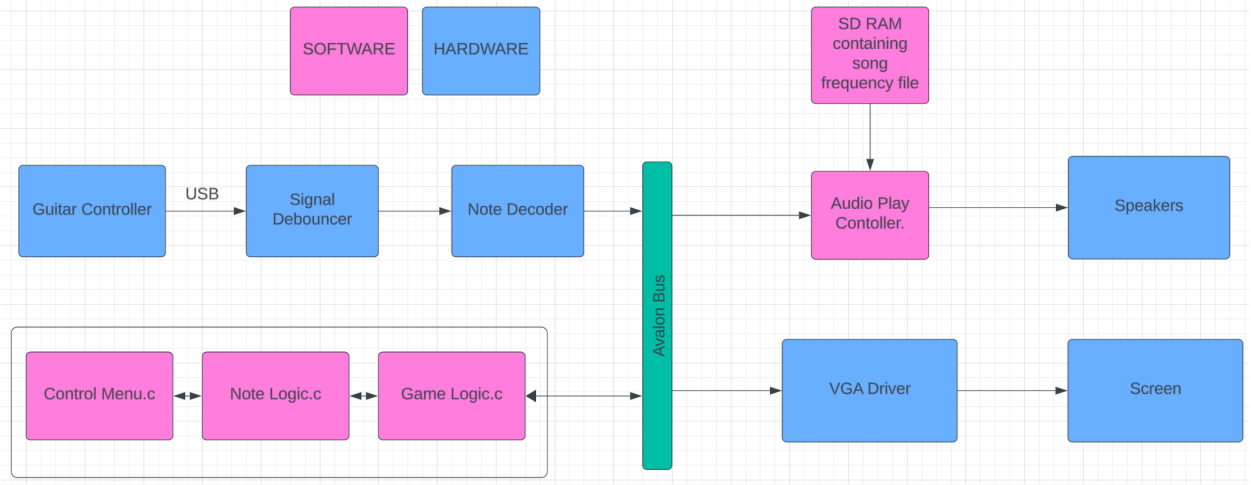
Guitar hero

Patrick Cronin pjc2192, Kiryl Beliauski kb3338, & Daniel Ivanovich dmi2115

Overview

For our final project, we are going to remake guitar hero, controlled by the original guitar-shaped adapter for the Wii controller. We will communicate with the guitar using a USB adapter for Wii accessories, which will require reverse-engineering the protocol used to encode the state of the guitar. The input from the player will be debounced and passed onto the game logic, which will also control our VGA and audio outputs. Our audio output will use the Wolfson CODEC to communicate with the speakers.

System Block Diagram



Algorithms

The guitar controller will be constantly logging inputs from the player. After reverse-engineering how our Wii accessory USB adapter encodes the state of the guitar, this processing logic will be written into a Note Decoder, which will interpret changes in the state of the guitar. The received state will then be passed to the Avalon bus to be read into the game logic. The Note Decoder will also work with a Signal Debouncing module to ensure we only update the game logic when there are meaningful changes in controller state.

The game logic will process player input like the original game. If the player strums too early, too late, or without all the right notes, they will miss and be penalized. If they strum correctly (within a certain time interval), they will hit. The main focus and challenge of the game logic will be coordinating the timing. The VGA output, guitar input, and audio all have to be synchronized, so delays in sending messages to the hardware components must be accounted for.

Regardless of the player input from the guitar, the vga will be continually displaying the current and following chords to play. These notes will enter from the top of the screen and proceed downward toward the play line at the bottom of the screen. The changing notes will appear along a constant background. The background of the display will be hardcoded in Verilog. There are 5 basic notes (represented by multicolor circles) and they move across the screen so we will be using sprites to depict the notes and center coordinates to control their velocity. The sprites will be preloaded into the FPGA SRAM and accessed for each note on the screen. The coordinates are updated in software and written to memory from which the VGA Driver will read the coordinates and display the notes at the given locations. The game logic will handle updates to these sprites and the queueing of future rows of sprites.

When the succeeding notes reach the play line they become the current expected note to be played. While within a certain range of the play line, the player will successfully hit the note. If the note is played incorrectly, the notes pass through the playline as if nothing happened. If the note is played correctly the note is changed from the basic sprite note to a highlighted sprite note and the user's score is updated.

Resource Budget

Classification	# of Elements	Pixels	Size(bits)
notes	5	80 * 20	192000
line	2	500 * 5	60000
number	10	20*20	96000
hit mark	1	80*20	38400
background music	1	N/A	24000000
Total			24234400

Hardware/Software Interface

The Game Logic is the central authority on the state of the game. Our main hardware components — those that process input from the Guitar, those that process audio output, and those that update the VGA's display — will all communicate with the Game Logic in only one direction. For example, the processed state of the guitar will always be passed to the Avalon Bus for the Game Logic to interpret, without any communication required from the Game Logic; the Game Logic will read the data as it sees fit, as it is the central timing authority. The Game Logic will also send the necessary data to the VGA and audio output to reflect the current state of the game and attempt to keep everything synchronized. Specifications for the communication between these components and the Game Logic follow.

For the Wolfson 8371 Audio CODEC:

REGISTER ADDRESS	BIT	LABEL	DEFAULT	DESCRIPTION
0000111 Digital Audio Interface Format	1:0	FORMAT[1:0]	10	Audio Data Format Select 11 = DSP Mode, frame sync + 2 data packed words 10 = I^2S Format, MSB-First left-1 justified 01 = MSB-First, left justified 00 = MSB-First, right justified
	3:2	IWL[1:0]	10	Input Audio Data Bit Length Select 11 = 32 bits 10 = 24 bits 01 = 20 bits 00 = 16 bits
	4	LRP	0	DACLRC phase control (in left, right or I^2S modes) 1 = Right Channel DAC data when DACLRC high 0 = Right Channel DAC data when DACLRC low (opposite phasing in I^2S mode) or DSP mode A/B select (in DSP mode only) 1 = MSB is available on 2nd BCLK rising edge after DACLRC rising edge 0 = MSB is available on 1st BCLK rising edge after DACLRC rising edge
	5	LRSWAP	0	DAC Left Right Clock Swap 1 = Right Channel DAC Data Left 0 = Right Channel DAC Data Right
	6	MS	0	Master Slave Mode Control 1 = Enable Master Mode 0 = Enable Slave Mode
	7	BCLKINV	0	Bit Clock Invert 1 = Invert BCLK 0 = Don't invert BCLK

Table 15 Digital Audio Interface Control

Data sent to the VGA controller and data sent to the Game Logic about the state of the guitar (very likely to change once we understand how their protocol works):

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Bit	0	1	2	3	4	5	6	7	8	9	10	11
2	Guitar State Packet:	Green Note	Red Note	Yellow Note	Blue Note	Orange Note	Strummed						
3	VGA Packet 0:	Green Pressed	Red Pressed	Yellow Pressed	Blue Pressed	Orange Presssed	Note Row 1 current height						
4	VGA Packet 1:	Green Present	Red Present	Yellow Present	Blue Present	Orange Present	Current Score Lower 8						
5	VGA Packet 2:	Green Present	Red Present	Yellow Present	Blue Present	Orange Present	Current Score Upper 8						
6	VGA Packet 3:	Green Present	Red Present	Yellow Present	Blue Present	Orange Present							
7	VGA Packet 4:	Green Present	Red Present	Yellow Present	Blue Present	Orange Present							
8	VGA Packet 5:	Green Present	Red Present	Yellow Present	Blue Present	Orange Present							
9	VGA Packet 6:	Green Incoming	Red Incoming	Yellow Incoming	Blue Incoming	Orange Incoming	Strum Hit Row 1						

This will be adjusted as we start implementing, but we plan to tell the VGA controller the current offset of the lowest row of notes on the screen and pass a bit for each note for each row. The VGA controller will use this data to calculate the coordinates of each needed sprite as each row will be a constant height. When the bottom row goes off screen, all rows above will be shifted down and the offset will be adjusted. The format of the data sent from the Guitar to the Game Logic will be completely dependent on the findings of our reverse-engineering, but the format illustrated above represents a best-case scenario in terms of ease of interpretation.