ECE241 | DE1-SoC Verilog Final Project Report
Chord Machine, Synthesizer and Sequencer
Ian Webster
12/2/19

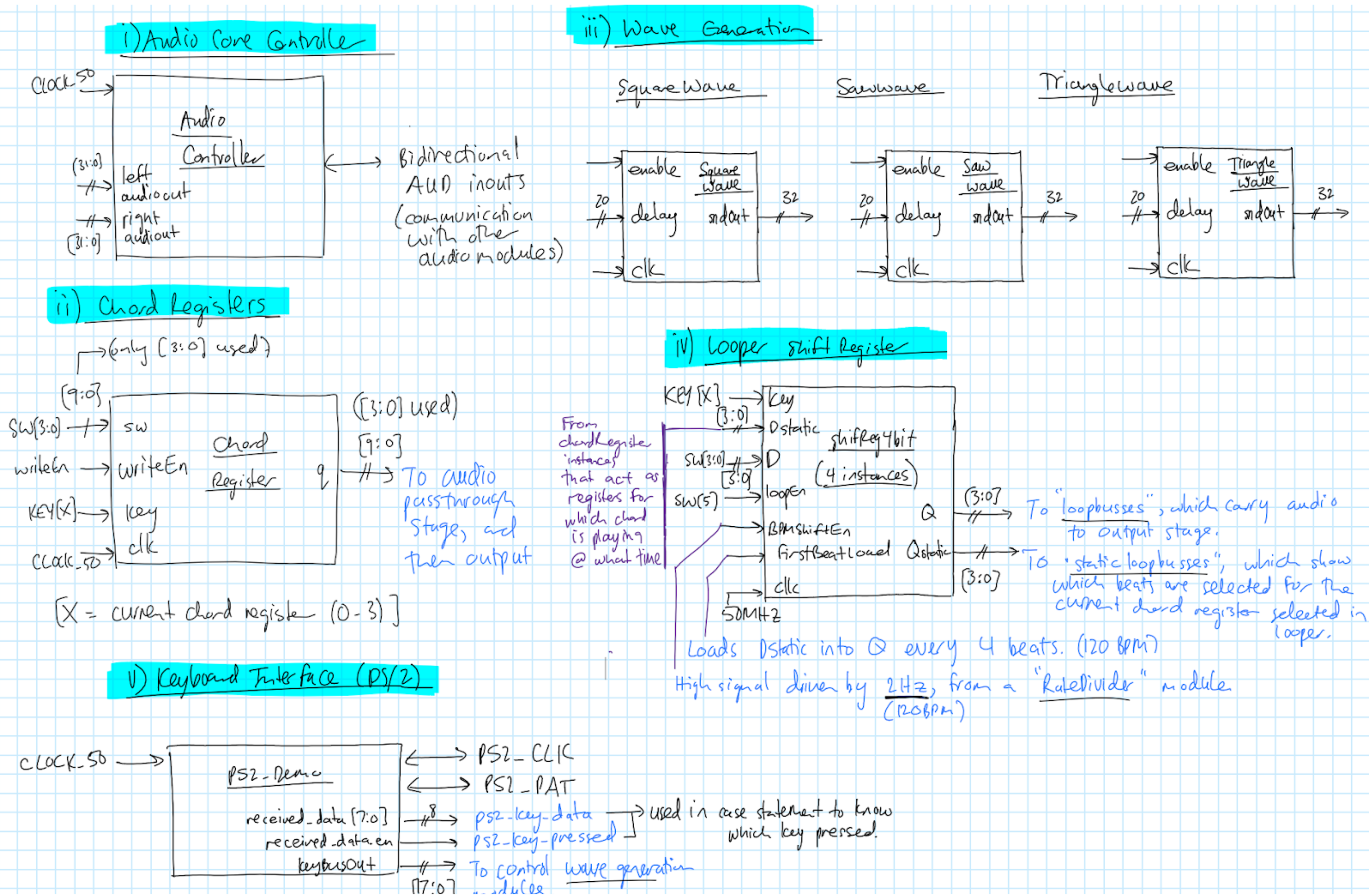
1. Introduction

This project aims to implement the Altera University Program's Audio IP Core to create a chord synthesizer, player, and sequencer using the DE1-SoC. This project is motivated by a passion for music and implementing technology with musical performance.

2. Design

Block Diagram of Major Parts/Subparts:

i) Audio Core Controller
ii) Chord Registers
iii) Wave Generation
iv) Looper Shift Register
v) Keyboard Interface


i) Audio Core Controller
CLOCK_50
Audio Controller
[31:0]
left audioout
right audioout
[31:0]
Bidirectional AUD inouts (communication with other audio modules)
ii) Chord Registers
(only [3:0] used)
[9:0]
SW[3:0]
writeEn
KEY[X]
CLOCK_50
sw
writeEn
key
clk
Chord Register
q
([3:0] used)
[9:0]
To audio passthrough stage, and then output
[X = current chord register (0-3)]
iii) Wave Generation
Square Wave
Sawwave
Trianglewave
enable
Square wave
Saw wave
Triangle wave
delay
sndOut
clk
20
32
iv) Looper Shift Register
KEY[X]
key
[3:0]
Dstatic
shiftReg4bit
(4 instances)
SW[3:0]
D
[3:0]
SW(5)
loopEn
BPMShiftEn
FirstBeatLoad
clk
50MHZ
Q
Qstatic
[3:0]
[3:0]
From chordRegister 'instances' that act as registers for which chord is playing @ what time
To "loopbusses", which carry audio to output stage.
To "static loopbusses", which show which beats are selected for the current chord register selected in looper.
Loads Dstatic into Q every 4 beats. (120 BPM)
High signal driven by 2Hz, from a "RateDivider" module (120BPM)
v) Keyboard Interface (PS/2)
CLOCK_50
PS2_Demo
received_data [7:0]
received_data_en
keybusOut
PS2_CLK
PS2_DAT
8
ps2_key_data
ps2_key_pressed
used in case statement to know which key pressed.
To control wave generation modules
[7:0]

Functionality of Parts, Top Down explanation:

i) Audio Core Controller

- Implemented using the template audio core project provided from [1]
- 7 audio controller verilog files: Audio Bit Counter, Audio In Deserializer (this is unused - the line in and mic are irrelevant to this project), Audio Out Serializer, Clock Edge, SYNC FIFO, Audio Clock, and Audio Controller. The audio library was unaltered.
- Outputs from wave generation modules were sent to the left_audio_out and right_audio_out 32-bit busses of the audio controller.

ii) Chord Registers

- SW[3:0] used for selecting notes in chords ("pitch switches")
- 4-key rollover for playing 4 chords simultaneously (i.e. can play multiple chords, with every pitch in any chord sent to the output at the same time)
- Play Mode: all pitch switches set to 0
- Write Mode: pitch switches not all 0, and loop enable ("loopEn"/SW[5]) set to 0. Can select your chord, and preview it by hitting the key of the register you'd like to write it to.

iii) Wave Generation

Square Wave Generator Module

- Creates a square wave for any note frequency
- Alternates 32-bit output amplitude of 32'd20 000 000 or -32'd20 000 000 according to given frequency parameter
- An output register, sndOut, stores the 32-bit output amplitude, and it is assigned 32'd20 000 000 or -32'd20 000 000 every (1/frequency)*th* of a second. EX: A4 is 440Hz. Therefore, the sndOut alternates values every 1/400th of a second. The perceived sound is 440Hz, or A4 on a piano.

Sawwave Wave Generator Module

- Creates a sawtooth wave for any note frequency.
- The slope for the waveform's amplitude is based on the *frequency* of the note being generated: we use the formula *delay_cnt*(maxAmplitude/delay)*2* to determine output amplitude, (where maxAmplitude = 20000000).
- delay is the clock frequency (50000000) divided by a reference pitch, i.e. A4 = 440Hz.
- delay_cnt is the internal counter register. Want amplitude to range from 0 to 20000000. Using A4 as reference (440Hz), delay_cnt ranges from 0 to (50000000 / 440 = 113636). Therefore, in order for the output to have the same perceived volume or dynamic range as a square wave, its slope must be 20000000 / 113636 = 178. As a result, we multiply the output amplitude by 178. The *2 is used in the formula because the saw wave is half the amplitude as the square wave (due to not using an unsigned register for negative values.)

Triangle Wave Generator Module

- Alternate incrementing and decrementing counter, same volume formula as sawwave module.
- When delay_cnt hits the max value, it starts decrementing.
- When delay_cnt hits the min value, it starts incrementing.

Wave Synthesis

- Select active waveforms with SW[9:7] - 9 is square, 8 is saw, 7 is triangle.
- All outputs from waveform generation modules are OR'd together to emulate an additive synthesis engine.

Unison/Phase Mode for Wave Generators

- Activated with SW[6].
- Wave generators play 2 of the same pitches at once (per note in a chord), but have them out-of-phase by 180 degrees; adds fullness to the sound.
- How to achieve an out-of-phase signal? - Achieved by using a formula for the "delay" parameter, passed into the wave generation modules: freq + freq/phaseDiff; where freq is the clock rate (50MHz) divided by a reference pitch, for example, A4 = 440Hz. phaseDiff is set to 2, for 180 degrees of phase difference.

iv) Looper Shift Register (for looping the Chord Registers)

- Activated via SW[5] ("loopEn")
- Loops 4 beats in a time signature of 4/4, at a tempo of 120BPM.
- Instantiate 1 shift register per chord. (EX: 4 chords, therefore need 4 registers)
- LEDs cycle to show which beat is being read from. If a switch has been selected at a certain LED position (0-3), then the *chord register that the switch was selected for* will play sound when the LED cycles to that position.
- Will loop chords at a base tempo of 120BPM - this is a constant parameter in the code

v) Keyboard Note-Playing Module

- Using the keyboard core sample from [2], we were able to interface the PS/2 keyboard with the wave-generation modules to create sound that *layers* on top of the chord register audio.
- Real world implication: the goal is to play solo on keyboard while chord register loops in background, acting as a backing track.

3. Report on Success

<u>What worked?</u>

- According to our original milestones, everything in our project was fully functional. (i.e. 1: Proof of concept with LEDs instead of audio, 2: Test audio sample code; 3: Implement audio with the first milestone.)
- In addition, the looper worked on a 4-beat loop at a tempo of 120 BPM. The notes were placed incorrectly at times (this is explained below).
- The keyboard interface worked as expected, however, our keyboard FSM could not handle simultaneous key presses.
- The wave generation worked smoothly (with occasional audio artifacts - pop/crackles) and notes were stored in and recalled from registers smoothly.

<u>What did not work?</u>

- The looper did not fully work in the end. 2 bits were reversed. Even though 4 *identical* shift register modules were instantiated, and each was controlled with 1 of the 4 keys, they did not perform as expected during looper operation.
- For example, beats loaded on registers 0 and 3 would load correctly and play correctly, but beats loaded on 1 and 2 would only *load* correctly. This may be a careless mistake, or it may have to do with incorrect wiring - however, significant effort was made to fix this error.
- Looper plays consecutive beats as 1 note, instead of >1 separate rhythmic units. This could be fixed in the future by doubling the beats per bar and writing and reading from only every other register. (There would always be a “rest” between sound outputs.)

4. What would we do differently?

- If we were to redo our project, we would consider varied milestones that are more representative of what we would be capable of in the given time period. The milestones set were too simple. We could’ve included features such as waveform synthesis, looper functionality, and keyboard interfacing within the milestones.

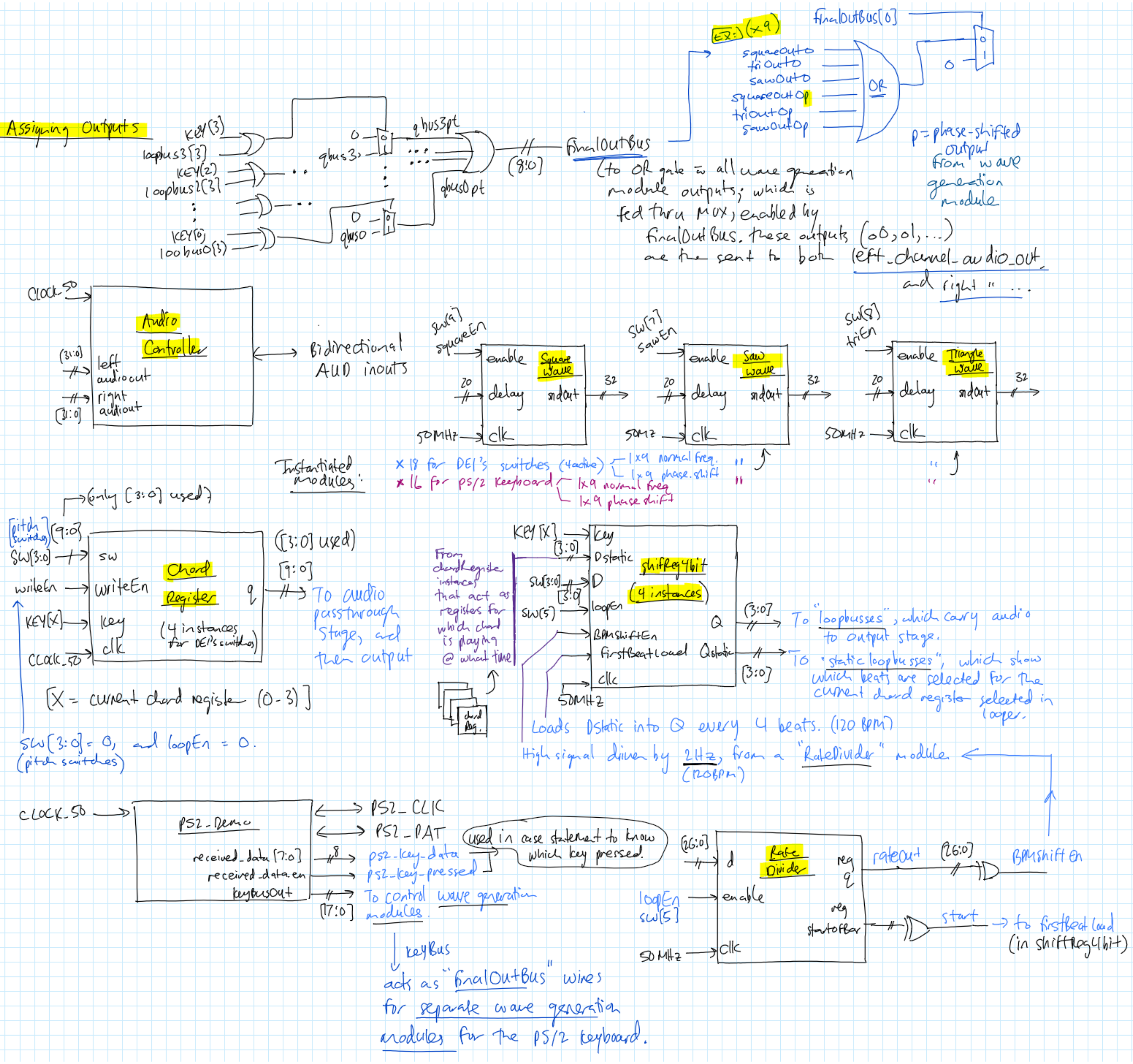
Extensions of Functionality for the Future

- Control volume, BPM, and other functionality via keyboard and T-Flip Flops
- Have octave switch: play bassline (lower octave), or solo (higher octave)
- Volume (pass in maxAmplitude as a parameter to all the wave generation modules. Change value with switches)
- Show BPM on HEX display; increment/decrement with the keyboard
- Could extend looper to have 8x2 registers. Once LEDs light up to end of one set of 8 beats, go to the next.

5. Appendices - Next Page

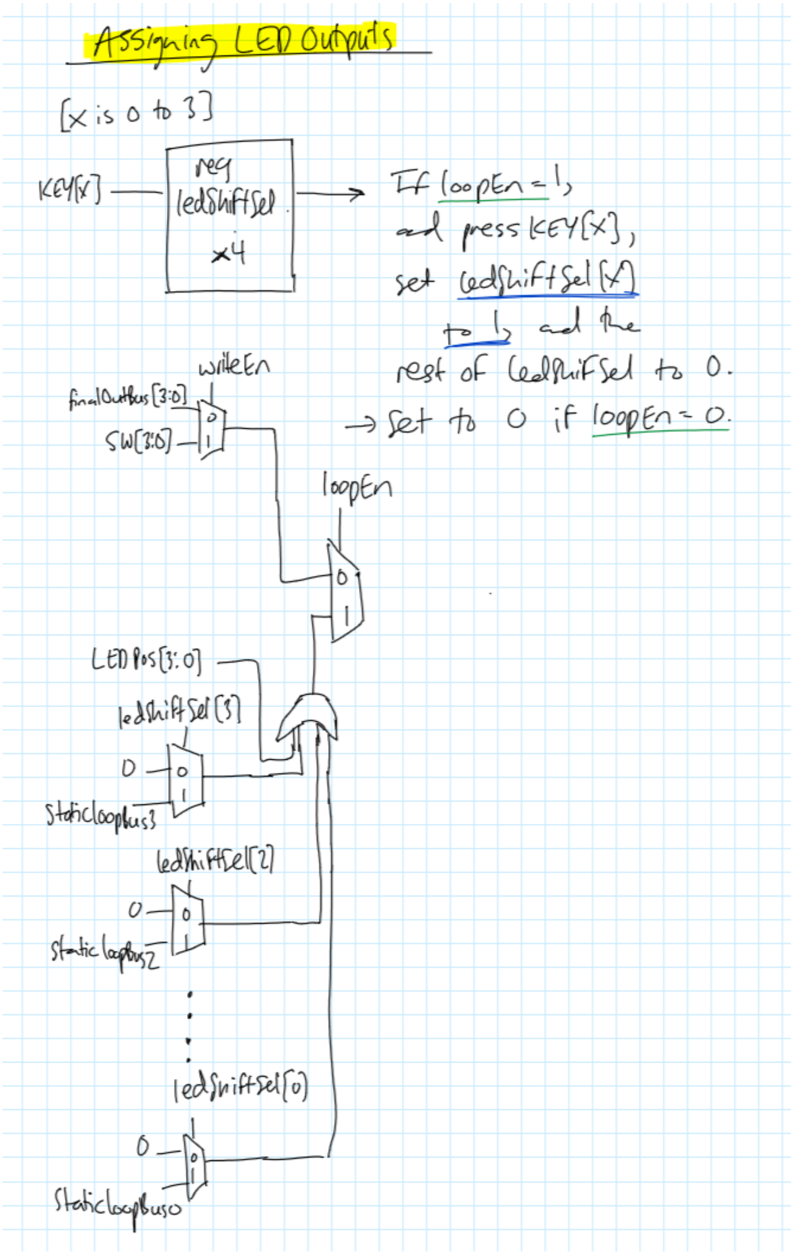
Appendix A: Schematics

Fig 1: Main Components and Connections


Assigning Outputs
KEY[3]
loopbus3[3]
KEY[2]
loopbus2[3]
KEY[0]
loopbus0[3]
0
qbus3
0
qbus0
qbus3pt
qbus0pt
[8:0]
finalOutBus
EX:) (x9)
finalOutBus[0]
squareOut0
triOut0
sawOut0
squareOut0p
triOut0p
sawOut0p
OR
0
1
0
p = phase-shifted output from wave generation module
(to OR gate w/ all wave generation module outputs; which is fed thru MUX, enabled by finalOutBus. these outputs (o0, o1, ...) are then sent to both left_channel_audio_out and right " ...
CLOCK_50
Audio Controller
[31:0]
left audioout
right audioout
[31:0]
Bidirectional AUD inouts
sw[9]
squareEn
enable
Square Wave
20
delay
sndOut
32
50MHz
clk
sw[7]
sawEn
enable
Saw wave
20
delay
sndOut
32
50MHz
clk
sw[8]
triEn
enable
Triangle Wave
20
delay
sndOut
32
50MHz
clk
Instantiated modules:
x 18 for DE1's switches (4 active) — 1x9 normal freq. / 1x9 phase.shift
x 16 for ps/2 keyboard — 1x9 normal freq / 1x9 phase shift
"
"
"
"
(only [3:0] used)
[pitch switches]
[9:0]
SW[3:0]
writeEn
KEY[X]
CLOCK_50
sw
writeEn
key
clk
Chord Register
q
(4 instances for DE1's switches)
([3:0] used)
[9:0]
To audio passthrough stage, and then output
[X = current chord register (0-3)]
SW[3:0] = 0, and loopEn = 0. (pitch switches)
From chordRegister instances that act as registers for which chord is playing @ what time
chord Reg.
KEY[X]
[3:0]
SW[3:0]
[3:0]
SW[5]
key
Dstatic
D
loopEn
BPMShiftEn
firstBeatLoad
clk
shiftReg4bit
(4 instances)
Q
Qstatic
50MHz
[3:0]
[3:0]
To "loopbusses", which carry audio to output stage.
To "static loopbusses", which show which beats are selected for the current chord register selected in looper.
Loads Dstatic into Q every 4 beats. (120 BPM)
High signal driven by 2Hz, (120BPM) from a "RateDivider" module
CLOCK_50
PS2_Demo
received_data [7:0]
received_data_en
keybusOut
PS2_CLK
PS2_DAT
8
ps2_key_data
ps2_key_pressed
used in case statement to know which key pressed.
[7:0]
To control wave generation modules.
keyBus
acts as "finalOutBus" wires for separate wave generation modules for the PS/2 keyboard.
[26:0]
d
loopEn
sw[5]
enable
50 MHz
clk
Rate Divider
reg q
reg startofBar
rateOut
[26:0]
BPMShift En
start
→ to firstBeat load (in shiftReg4bit)

Appendix A Continued:

Fig 2: LED Output Logic



If loopEn = 1, and press KEY[x], set ledShiftSel[x] to 1, and the rest of ledShiftSel to 0.

→ Set to 0 if loopEn = 0.