EE 108 Final Project Report

Music Synthesizer +

By Noor Fakih & Adrian Saldana

Overview

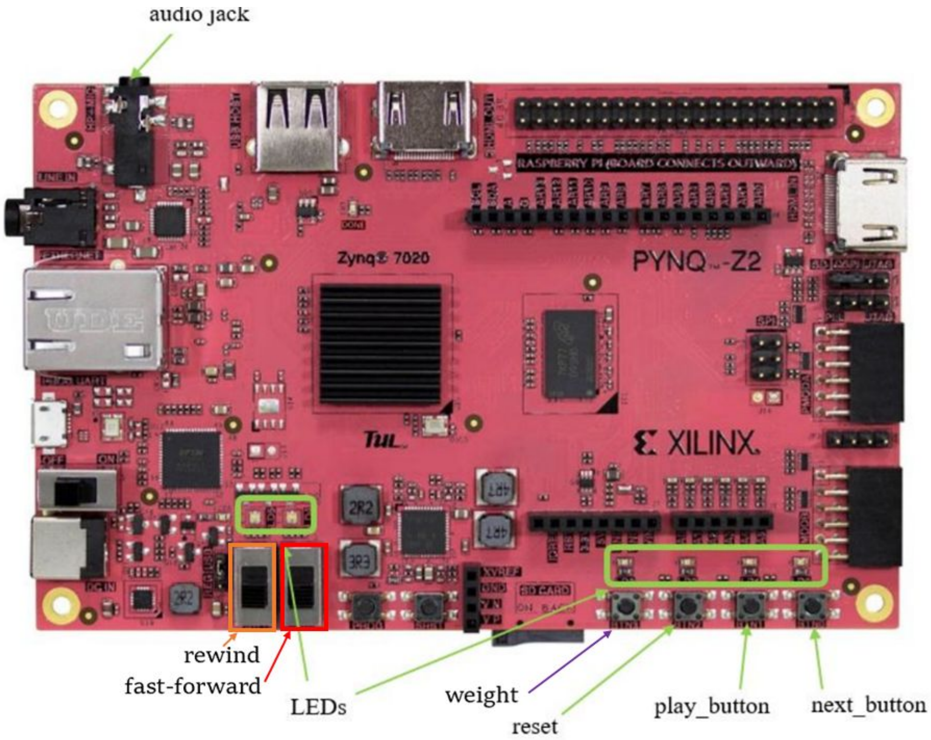
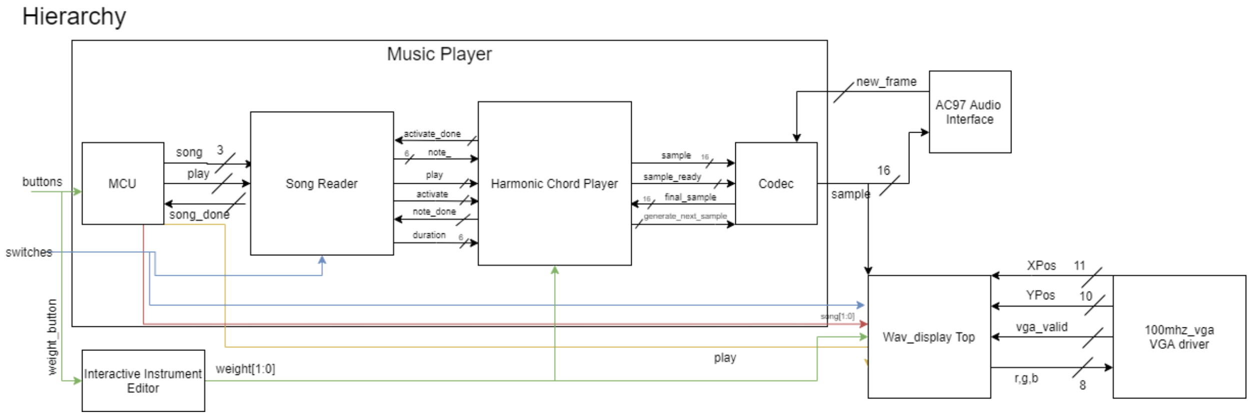


diagram of board's controls
rewind switch - r_switch1
Fast forward switch -
ff_switch0
bttn3 - weight_button



MCU and song_reader basically retain the same functionality as Lab 4&5. The only difference is in song_reader, it will half the duration for fast-forward and rewind. Harmonic Chord player will then decide which note_player channel to load the note it receives into. It will take the note and the user-set weight to add harmonics. The samples generated by each channel and their harmonics are then combined and sent to the codec.

If the user presses the weight button, then the interactive instrument editor will adjust the weight so that create_harmonics will add the correct amount of samples to create a harmony.

Wave_display_top functions as normal for the most part, except it takes in additional values in order to properly display icons.

Chords - 4 pts

Feature Description

Our music synthesizer has the ability to play up to 3 notes at the same time of varying length.

Implementation Description

*refer to table in Module IO Appendix and figure in Diagram Appendix
Song_reader passes the new note to harmonic chord player who then decides which of 3 note player channels it should load the note into. There's a 4th channel but it isn't a note_player channel - it's strictly reserved for when activate "notes" are passed in to ensure proper passage of time.

Testing

We created songs that utilize chords, and played 2 or 3 notes pretty often to testbench with. We kept an eye on the different channels and the notes/durations they were loading in to make sure they were correct. We also made sure that our counters for each channel worked properly and that each note was only played for its allotted duration. Another great test was comparing the waveform of the three combined notes and their individual waveforms to check that the samples remained sinusoidal and didn't take an unexpected shape.

When we first started, we only had three note_player channels and were running into an issue where we got stuck in the waiting state for song_reader. Song_reader would stop after loading the three noteplayer channels and wait for their notedones to go HIGH, which couldn't happen until the fourth "note" activate was loaded in so that the notes counters would start. We added a 4th channel meant only for the activate signal and that resolved that issue.

Since we sometimes had to combine 3 notes 16-bit samples to output a chord, we increased our final sample size to 18 instead of scaling down the resultant. We didn't realize until the very end that the 18-bit signal was conflicting with the codec and its setup and was causing intense audio distortion because the issue wasn't visible via the testbench. We partially resolved the issue by scaling down the note's final samples and fitting it within 16-bits.

There is still some distortion that occurs at seemingly random intervals. There was a timing violation which was resolved by pipelining. We added another flip-flop into sine_reader and a flip flop for each channel inside of create_harmonics. There is no longer a timing violation, but some slight distortion still remains.

Harmonics & Interactive Instrument Editing/Multiple Voices blend - 4 pts

Feature Description

Every note can be played as a harmony instead. The user is able to set the "weight" of the harmonics by pressing the weight button aka bttn3.

- 0 for base note only and no harmonics (default)
- 1 for base note scaled to 5/8ths amplitude + sample scaled to 3/8ths amplitude
- 2 for base note scaled to 5/8ths amplitude + sample scaled to 1/4th amplitude + another sample scaled to 1/8th amplitude

There is a little bar graph on the bottom left of the display which indicates what weight is currently set to (refer to image on p. 4).

Implementation Description

*refer to table in Module IO Appendix and figure in Diagram Appendix

Weight is set using the iie module. It takes in weight_button and sets weight to the next value every time it's pressed. When weight_button is pressed while weight is 2, it loops around and sets weight to 0.

The harmonies are created by create_harmonic. Each note_player channel utilizes freq_rom to receive the proper step size for its note, and then passes it create_harmonic which has three sine_reader channels to create 3 samples. According to the set weight, the samples are then scaled and combined.

Testing

Testing iie was easy as all we had to do was set weight_button to HIGH and ensure that it properly goes from 0 to 1 to 2, and back to 0.

The harmonics was a bit more difficult as we ran into timing violations as we have lots of addition of large numbers occurring. We looked into the worst slack and where the thread was and traced it back to carry addition. This was solved by pipelining! We added a flip-flop within note_player so that after it received the output from create_harmonic (where 16-bit addition was occurring), it held it for a clock cycle before passing it to harm_chord_player where more 16-bit addition was occurring. Harm_chord_player was difficult to test as a lot of numbers had to be hand-calculated so it was tested with music_player's test bench instead. With information being exchanged between song_reader and the codec it made sense to test it with music_player in order to utilize song_rom.

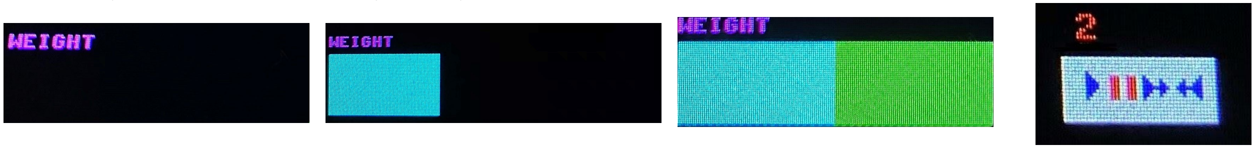
The rest of the testing focused on the ability to use weight to create the right harmony, and that our output from create_harmonics was as expected. We spent lots of time looking at the individual sample waveforms and made sure it lined up with what the resultant waveform looked like.

Context-Sensitive Icons - 2 pt

Feature Description

There are three different sets of context-sensitive icons.

- Weight bar graph: a bar graph that indicates the current number weight is set to.
- Current song: displays which song number currently on.
- Play controls: a box that has icons for play, pause, fast-forward and rewind. If the music synthesizer is currently in any of those states then it will color the icon red.



^weight graphical display (first 3 pictures). Last picture is of the control box. Shows that it is currently paused on song 2.

Implementation Description

*refer to table in Module IO Appendix and figure in Diagram Appendix
This feature is self-contained inside of wave_display. We added additional logic which determines if the current x and y coordinate is within the range of certain constants (defined to be the starting and ending xy positions for each icon), then to set color and addr for tcgrom for that icon. We edited tcgrom to include icons for play, pause, fast-forward, and rewind.

Testing

As this feature is purely display components, most of the testing took place by generating bit-stream and looking at the actual display. We started off by coding/testing this feature within Lab5's project by hard-coding values for things like the fast-forward and rewind switch. We wanted to ensure nothing we did within wave_display affected the depiction of our sine wave, without being concerned that our code for the final was affecting the sine wave.
In terms of simulations, we checked several different x and y positions that correlated to be before, in, and after our icons so that the correct rgb values and addr are received. The rest was spent viewing it on the actual display.

After being satisfied with how it all looked on Lab5, we moved it into the Final project. The most difficult aspect of this was doing the correct math to display the icons in the correct positions and having to wait for generate bitstream in order to look at it. We are especially proud of the playback button control box!

Playback Controls - 1 pt

Feature Description

While switch0 is HIGH, the board will enter fast-forward mode and play any song at double speed. While switch1 is HIGH, the board will enter rewind mode and play the song both backwards and at double speed. We haven't implemented playing the song in reverse.

Implementation Description

*refer to table in Module IO Appendix and figure in Diagram Appendix

For the most part song_reader upholds its original implementation from Lab4&5, except for the addition of fast-forward and rewind. If either ff_switch0 or r_switch1 are HIGH, then the duration will be divided by 2 before being sent to harm_chord_player.

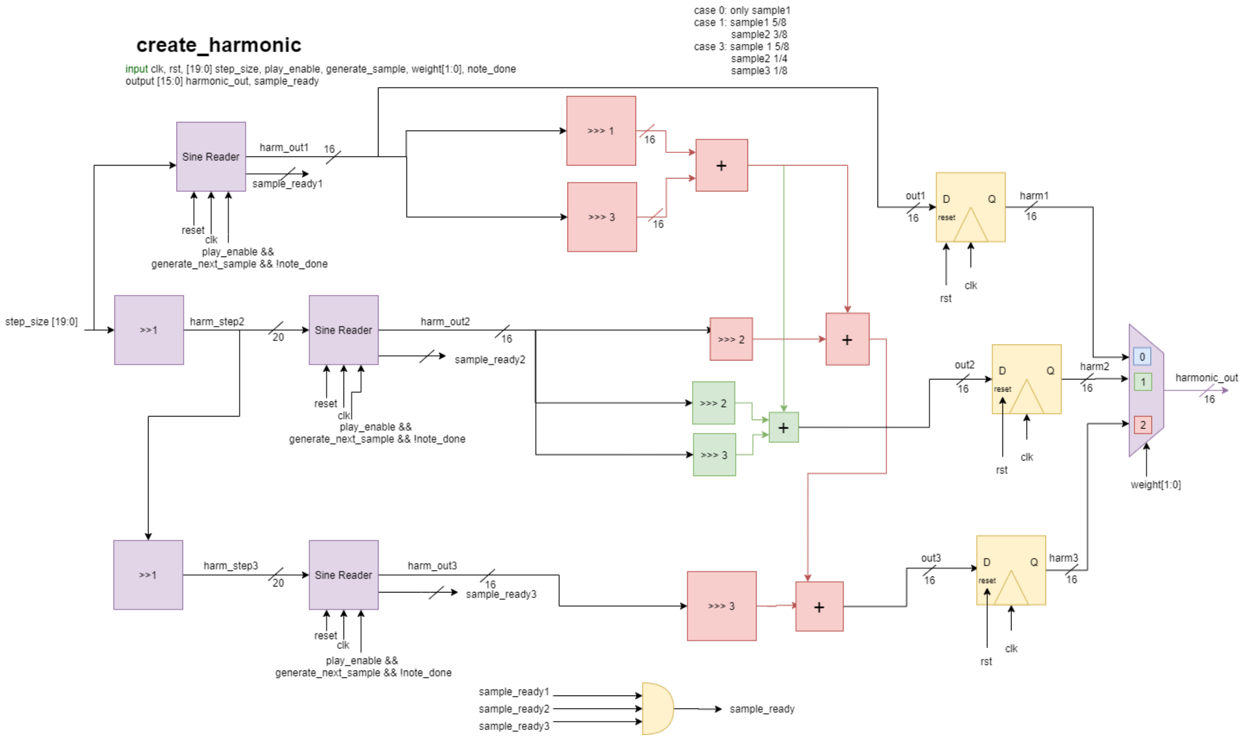
Testing

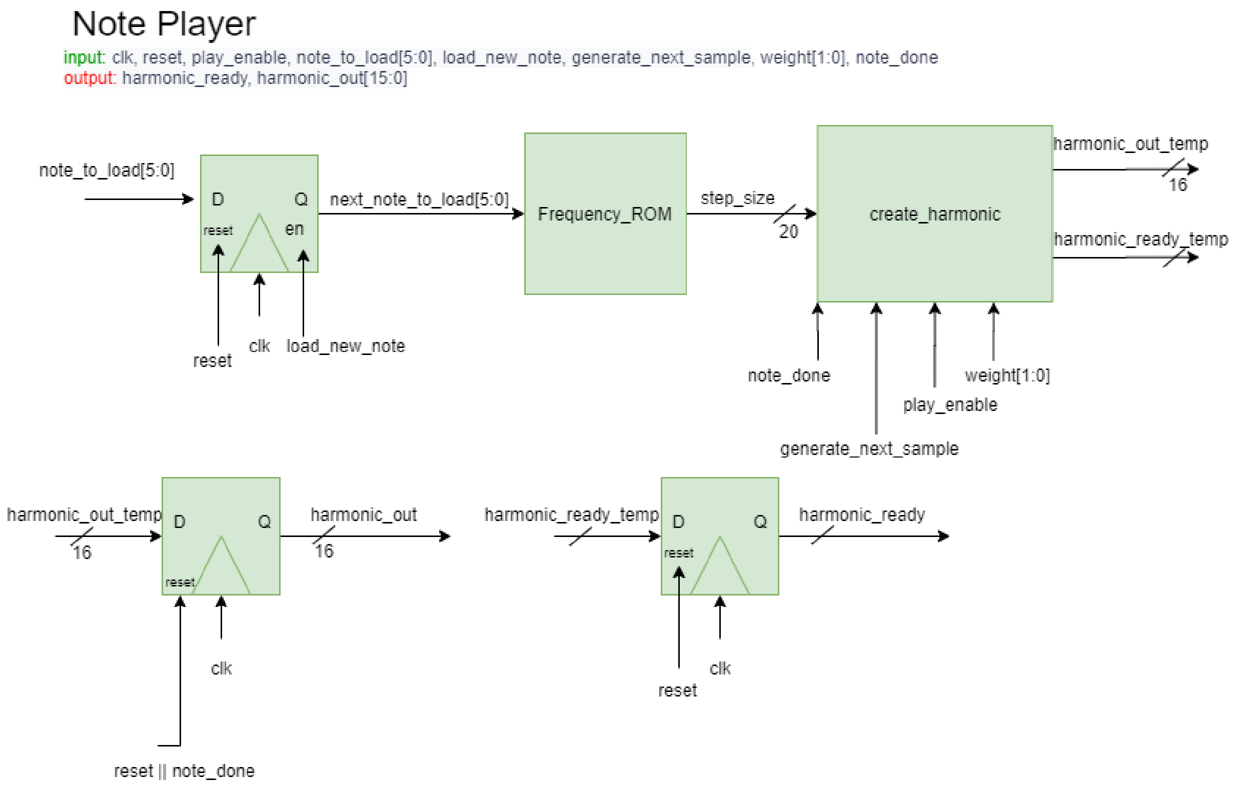
For both fast-forward and rewind, we looked in the testbench to ensure that when either switch is HIGH that the duration was cut in half. We also wanted to make sure that the addition of either feature did not mess with any other aspect of the code. Once we generated bit-stream, we made sure that the audio sounded correct, the sine wave displayed correctly, and that adjusting either switch gave us the results we expected.

We ran into a problem with rewind not decrementing properly, but were able to solve that by modifying the ternary statements within the case statements to include the case where rewind is HIGH and at curr_note_num. We kept running into a new bug where rewind would cause the board to either get stuck on the first note of the song or would complete a song and cause the board to skip a song. At this point we had to prioritize wrapping up the project over debugging (especially since one of our boards lost a jumper and stopped working), so we took out the decrementing functionality of rewind.

Appendix

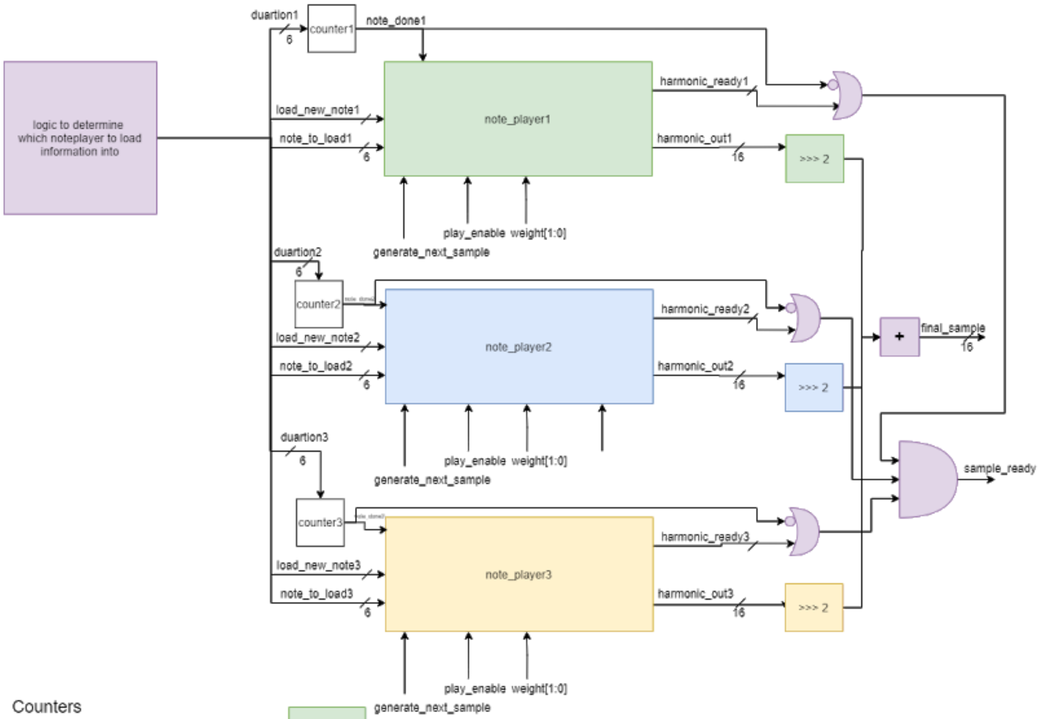
FSMs and Block Diagrams



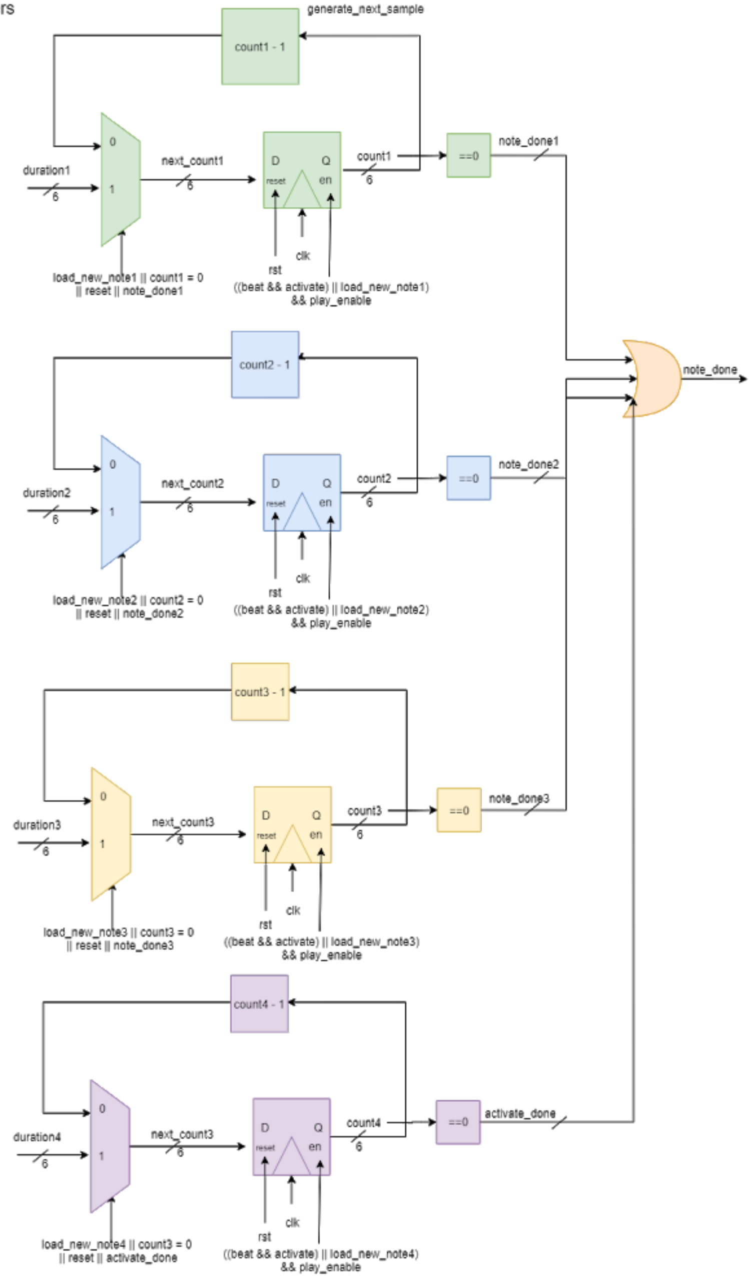


harm_chord_player

input clk, rst, play_enable, [5:0] note_to_load, [5:0] duration, load_new_note, activate, beat, generate_next_sample, [1:0] weight
output note_done, sample_ready, [15:0] final_sample, activate_done

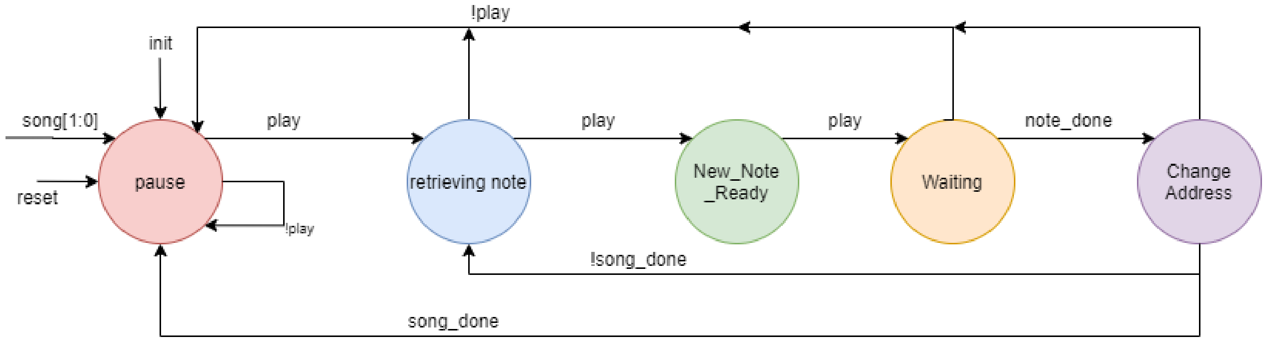


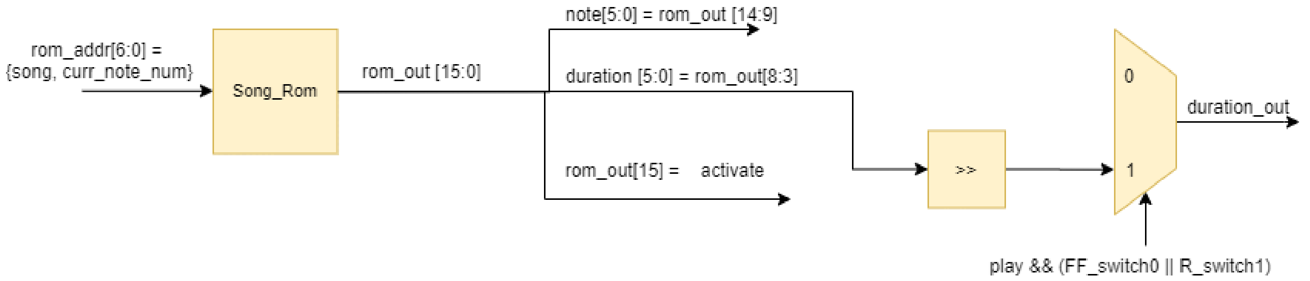
Counters

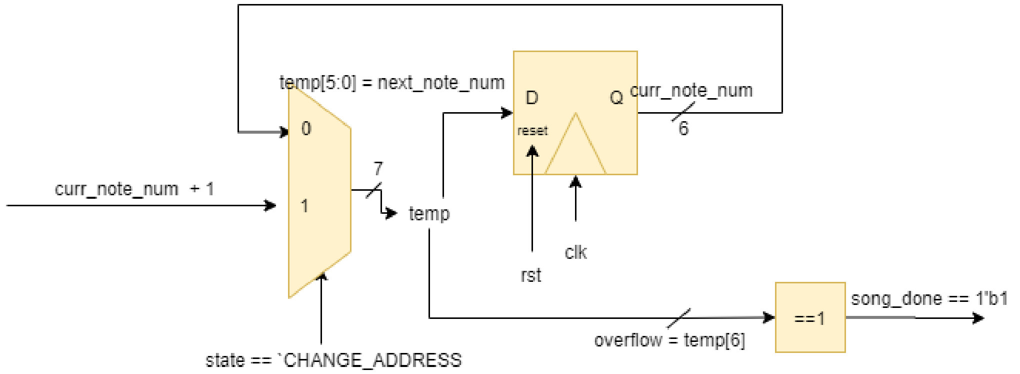

count1 - 1
0
1
duration1
6
next_count1
6
D
Q
reset
en
count1
6
==0
note_done1
clk
rst
load_new_note1 || count1 = 0 || reset || note_done1
((beat && activate) || load_new_note1) && play_enable
count2 - 1
0
1
duration2
6
next_count2
6
D
Q
reset
en
count2
6
==0
note_done2
clk
rst
load_new_note2 || count2 = 0 || reset || note_done2
((beat && activate) || load_new_note2) && play_enable
note_done
count3 - 1
0
1
duration3
6
next_count3
6
D
Q
reset
en
count3
6
==0
note_done3
clk
rst
load_new_note3 || count3 = 0 || reset || note_done3
((beat && activate) || load_new_note3) && play_enable
count4 - 1
0
1
duration4
6
next_count3
6
D
Q
reset
en
count4
6
==0
activate_done
clk
rst
load_new_note4 || count3 = 0 || reset || activate_done
((beat && activate) || load_new_note4) && play_enable

song_reader

input: clk, reset, play, song[1:0], note_done, activate_done, ff_switch0, r_switch1
output: song_done, note[5:0], duration[5:0], new_note, activate

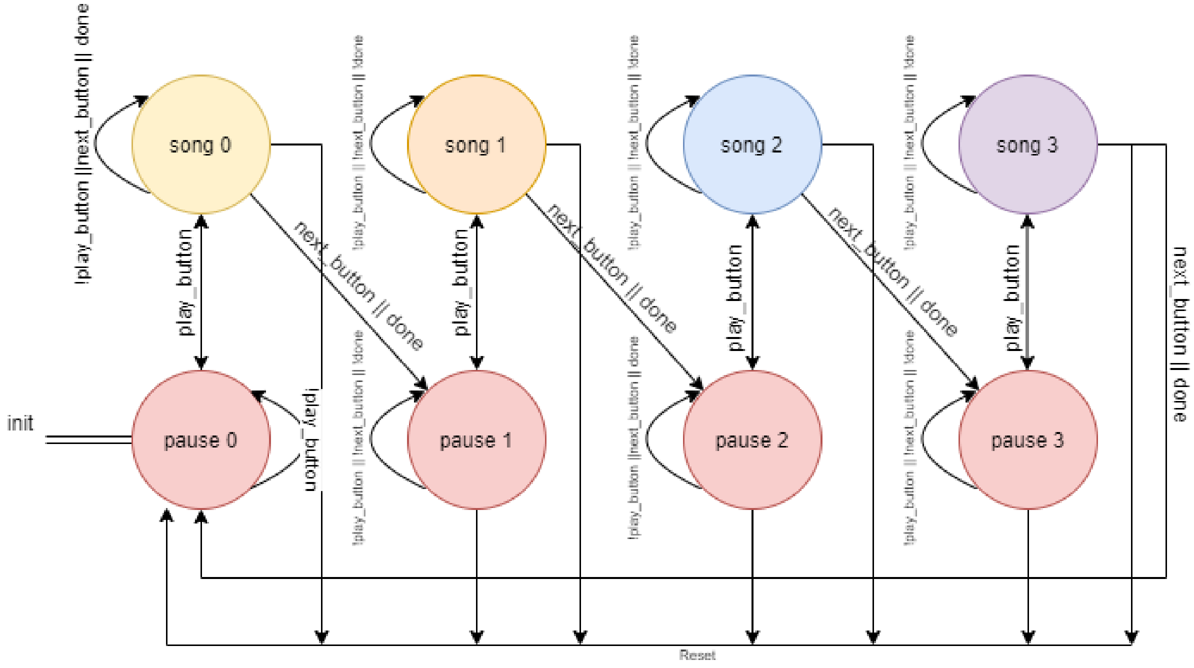



note[5:0] = rom_out [14:9]
rom_addr[6:0] = {song, curr_note_num}
Song_Rom
rom_out [15:0]
duration [5:0] = rom_out[8:3]
0
duration_out
rom_out[15] = activate
>>
1
play && (FF_switch0 || R_switch1)



MCU

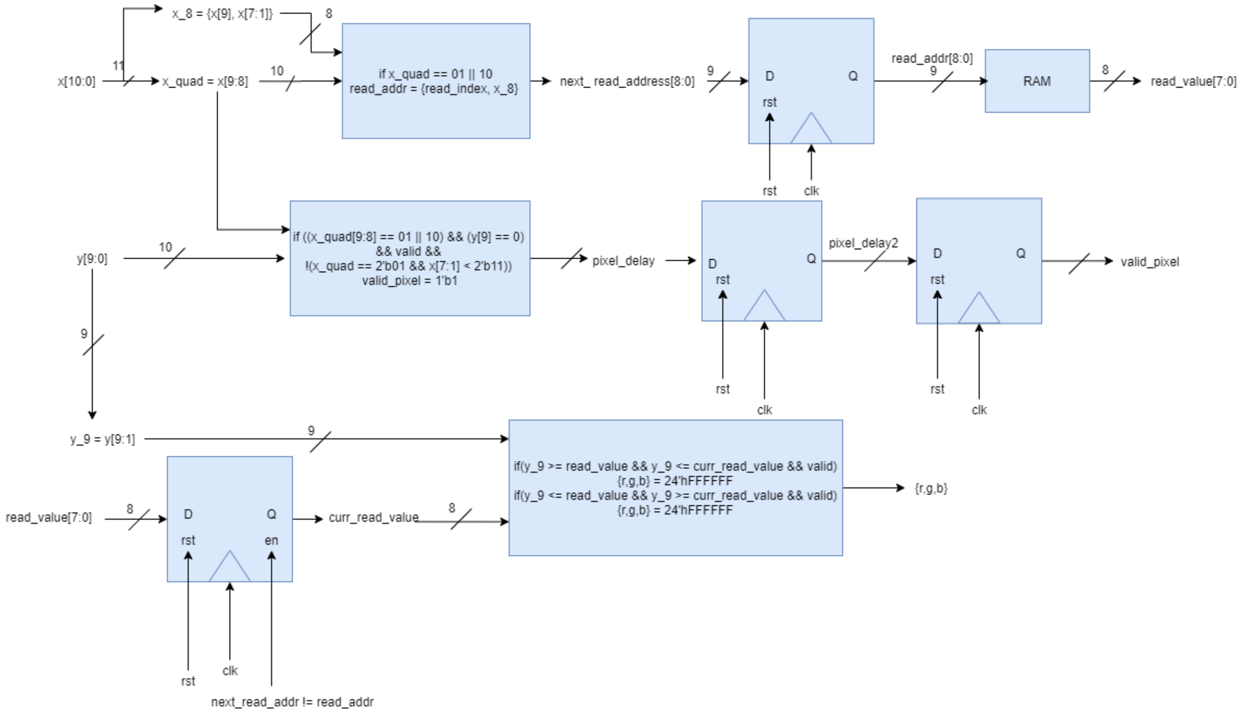
inputs clk, reset, play_button, next_button, song_done
output play, reset_player, song[1:0]


!play_button ||next_button || done
song 0
song 1
song 2
song 3
!play_button || !next_button || !done
!play_button || !next_button || !done
!play_button || !next_button || !done
play_button
play_button
play_button
play_button
next_button || done
next_button || done
next_button || done
next_button || done
init
pause 0
pause 1
pause 2
pause 3
!play_button
!play_button || !next_button || !done
!play_button ||!next_button || !done
!play_button || !next_button || !done
Reset

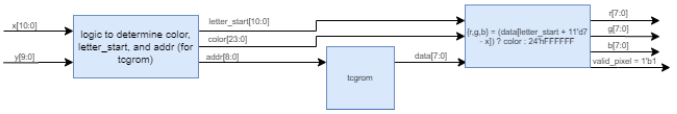
wave_display

input: clk, reset, x[10:0], y[9:0], valid, [7:0] read_value, read_index, [1:0] weight, ff_switch0, r_switch1, [1:0] song_num, play
output: [8:0] read_address, valid_pixel, [7:0] r, [7:0] g, [7:0] b

logic to display sine wave

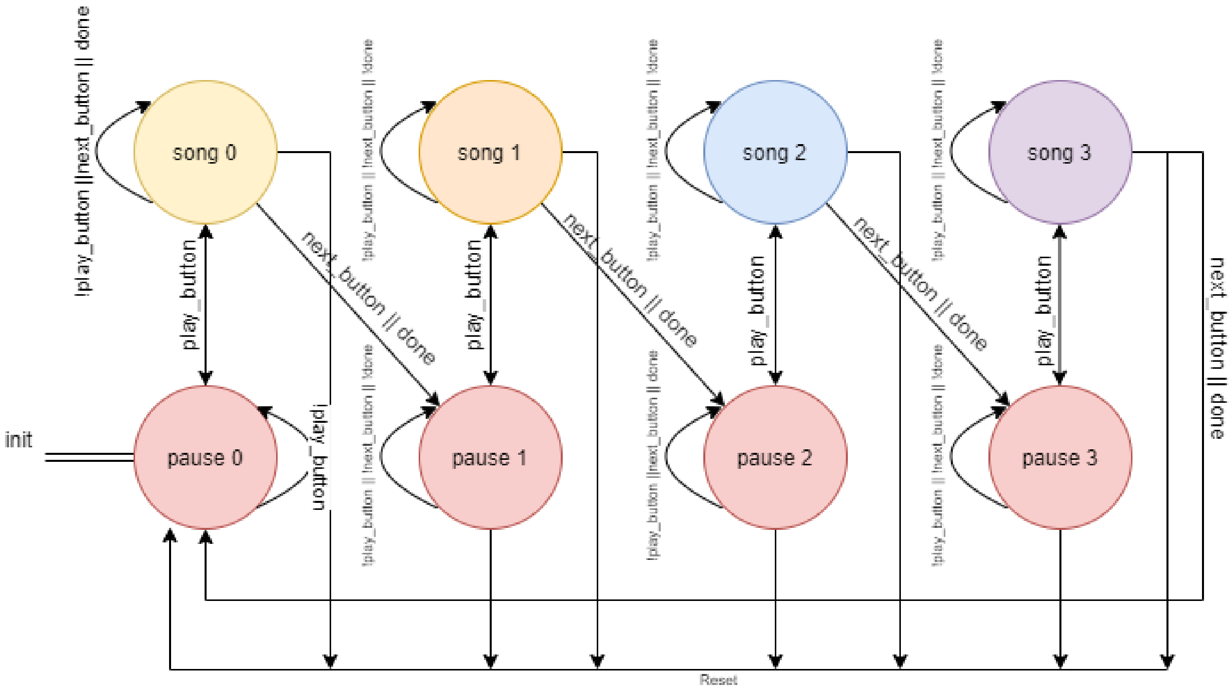


logic to display weight, icons, and song num


x[10:0]
y[9:0]
logic to determine color, letter_start, and addr (for tcgrom)
letter_start[10:0]
color[23:0]
addr[8:0]
tcgrom
data[7:0]
{r,g,b} = (data[letter_start + 11'd7 - x]) ? color : 24'hFFFFFF
r[7:0]
g[7:0]
b[7:0]
valid_pixel = 1'b1

MCU

inputs clk, reset, play_button, next_button, song_done
output play, reset_player, song[1:0]

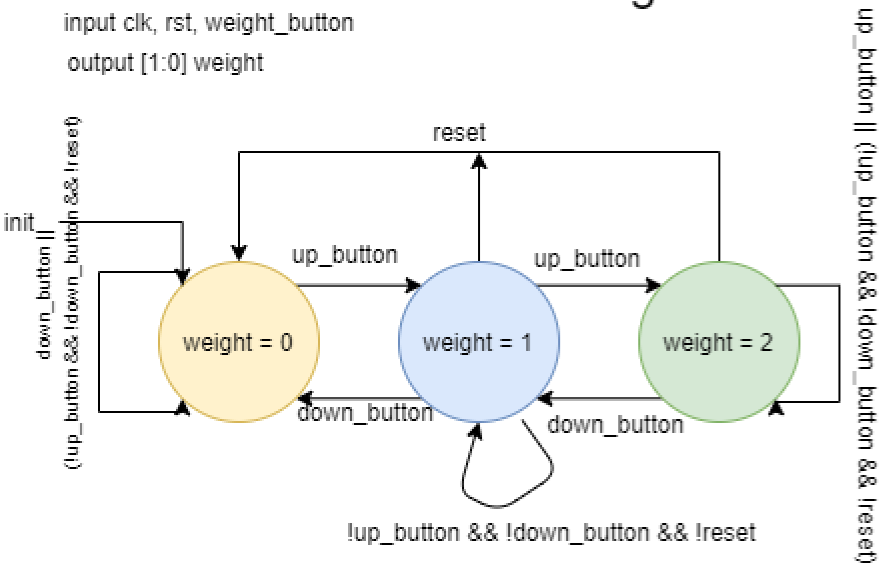


!play_button ||next_button || done
song 0
song 1
song 2
song 3
!play_button || !next_button || !done
!play_button || !next_button || !done
!play_button || !next_button || !done
play_button
play_button
play_button
play_button
next_button || done
next_button || done
next_button || done
next_button || done
init
pause 0
pause 1
pause 2
pause 3
!play_button
!play_button || !next_button || !done
!play_button ||next_button || done
!play_button || !next_button || !done
Reset

interactive instrument editing

input clk, rst, weight_button

output [1:0] weight



(!up_button && !down_button && !reset)
down_button ||
init
reset
up_button
up_button
weight = 0
weight = 1
weight = 2
down_button
down_button
!up_button && !down_button && !reset
up_button || (!up_button && !down_button && !reset)

Module IO Tables

Music Player

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
play_button	input	A one-cycle pulse indicating the play_button has been pressed
next_button	input	A one-cycle pulse indicating the next_button has been pressed
new_frame	input	The raw new_frame signal from the ac97_if codec
weight[1:0]	input	Output from iie. Indicates how
ff_switch0	input	Correlates to the state of switch 0 on the board. HIGH if the song should be played fast-forward. LOW to play at normal speed.
r_switch1	input	Correlates to the state of switch 1 on the board. HIGH if the song should be played in reverse. LOW to play normally.
new_sample_generated	output	This output must go high for one cycle when a new sample is generated.
sample_out[15:0]	output	Our final output sample to the codec. This needs to be synced to new_frame
current_song[1:0]	output	Indicates which song the music_player is on. Input for wave_display_top to correctly display icons.
play	output	HIGH when notes are being played aka the play state, LOW when in pause state. Input for wave_display_top to correctly display icons.

Master Control Unit

No changes made from Lab 4&5 implementation

Song Reader

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
play	input	True if the song reader should be playing
song[1:0]	input	The song to play
note_done	input	From the note_player to indicate that the note has finished and that it is ready for the next note. It is created by ORing the note_done associated with each of the 3 note_player channels
ff_switch0	input	Correlates to the state of switch 0 on the board. HIGH if the song should be played fast-forward. LOW to play at normal speed. Duration will be cut in half.
r_switch1	input	Correlates to the state of switch 1 on the board. HIGH if the song should be played in reverse. LOW to play normally. Duration will be cut in half.
activate_done	input	Input from harm_chord_player which indicates if the activate duration has completed so that a new activate duration can be properly loaded in.
note[5:0]	output	The note from the song_rom to play now.
duration[5:0]	output	The duration for the note from the song_rom to play now
new_note	output	One cycle pulse that tells the note_player to latch in the values on note and duration and start playing that note.
song_done	output	True if the song has finished
activate	output	From the beginning of a note's address - indicates whether or not time is passing yet and the notes should actually be played. True means there is passage of time.

Harmonic Chord Player

The purpose of this module is to create three different channels in order to make it possible to play chords of up to 3 notes. The module will determine which channel is empty when it has a load_new_note is true and set its respective note_player's duration, note_to_load and load_new_note. It will then instantiate three different create_harmonic modules in order to create a harmonic for each channel and then return final_sample which is made up of all the samples combined together.

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
play_enable	input	True if the song reader should be playing
note_to load[5:0]	input	The note to load, passed from song_reader to one of the three note_player channels
duration[5:0]	input	The duration for the note from the song_rom to play now
load_new_note	input	One cycle pulse that tells the note_player to latch in the values on note and duration and start playing that note.
activate	input	From song_reader, indicates whether or not time is passing yet and therefore whether or not the note should be played yet
beat	input	Goes high for one cycle at 48Hz
generate_next_sample	input	From the codec_conditioner telling us to generate and output the next sample
weight[1:0]	input	Input for note_player to pass to create_harmonic in order to add the correct number of harmonies as indicated by the user.
note_done	output	Goes high when we have finished playing our note. It is created by ORing the note_dones for each of our 3 note_player channels.
sample_ready	output	Tells the codec_conditioner that we have a new sample ready for it. It is created by ANDing all the sample_readys from our three create_harmonic channels.

activate_done	output	Input for song_reader in order to ensure proper loading of the next active duration not happening too soon. HIGH when activate duration gets to 0. LOW when still counting.
final_sample[15:0]	output	The 16-bit audio sample output for our note(s). It adds up the samples for each note we currently have in our note_player channels and their harmonies.

Note Player

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
play_enable	input	True if the song reader should be playing
note_to load[5:0]	input	The note to load, passed from song_reader to one of the three note_player channels
load_new_note	input	One cycle pulse that tells the note_player to latch in the values on note and duration and start playing that note.
weight[1:0]	input	Input for note_player to pass to create_harmonic in order to add the correct number of harmonies as indicated by the user.
generate_next_sample	input	From the codec_conditioner telling us to generate and output the next sample
note_done	input	Goes high when we have finished playing our note. Passed into create_harmonic in order to generate samples only when note_done is LOW.
harmonic_ready	output	Tells the harm_chord_player that we have a new sample ready for it.
harmonic_out[15:0]	output	The 16-bit audio sample output for our note including our harmonies.

Create Harmonic

The purpose of this module is to return a sample that represents a harmony for the passed in sample_in. Weight, which is edited by interactive instrument editing, will determine how many samples are used to generate a harmony.

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
play_enable	input	True if the song reader should be playing
generate_next_sample	input	From the codec_conditioner telling us to generate and output the next sample.
weight[1:0]	input	The user can change the weight of the harmonics, i.e. the number of samples that will be generated for the note, by pressing the right button. The options are none, one or two. None means that it will be the base note only.
note_done	input	Passed from harm chord player to note player to create harmonic in order to ensure samples are only generated when note_done is LOW.
step_size[19:0]	input	Passed in from note_player, used to generate the appropriate samples.
harmonic_out[15:0]	output	The 16-bit audio sample output for our note, is a combination of all our generated samples to create a harmony for our note.
sample_ready	output	Tells the codec_conditioner that we have a new sample ready for it.

Sine Reader

No changes made from Lab 4&5 implementation except added another flip-flop.

Wave Display Top

No changes made from Lab 5 implementation except passing in weight, play, ff_swtich0, r_switch1 and current song to wave_display in order to render the appropriate icons.

Wave Capture

No changes made from Lab 5 implementation

Interactive Instrument Editing

The purpose of this module is to edit weight, the variable which indicates how many samples will be used in create_harmonic to generate a harmony. Weight can only range from 0 (no harmony, only the base note) to 2 (base note at ⅝, sample1 at ¼ and sample2 at ⅛).

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
weight_button	input	If true then we want to increase weight by 1. If weight is 2 then it will wrap around to 0.
weight[1:0]	output	By default it is 0. Outputs how many samples will make up the harmony for a note in create_harmonic.

Wave Display

Signal	**Direction**	**Description**
clk	input	Clock signal
reset	input	Reset signal
x[10:0]	input	The current X position of the VGA display
y[9:0]	input	The current Y position of the VGA display
valid	input	Whether or not the VGA coordinates are valid for displaying data.
read_index	input	Bit indicating which part of RAM to read from
read_value[7:0]	input	The data you read back from the RAM
weight[1:0]	input	Input from iie (interactive instrument editing module). Indicates how many boxes should display to demonstrate what the weight is set to.

song_num[1:0]	input	Input from music player. Used to display what song number we are currently on.
ff_swtich0	input	If the switch is HIGH then the fast-forward icon is highlighted, if the switch is LOW then the fast-forward icon is not highlighted.
r_swtich1	input	If the switch is HIGH then the rewind icon is highlighted, if the switch is LOW then the rewind icon is not highlighted.
play	input	Input from Music Player. If play is HIGH then play icon is highlighted, if play is LOW then pause icon is highlighted.
read_address[8:0]	output	The address in the RAM to read. Remember: it takes one cycle to get the data back!
valid_pixel	output	True if the pixel specified by x and y should be turned on.
r[7:0] g[7:0] b[7:0]	output	The color you want the wave to be. This can be assigned a constant value.