progress notes

25 March 2023 2:00 PM

work on the project not completed by me, to my understanding:
- importing title sprite to project
- creating and importing user ID sprite to project
- use of 7-segment display on board
- explanation presentation video

10/4/23
created experimental accelerometer project using resources found online, created by david marion.

https://github.com/FPGADude/Digital-Design

15/04/23
Spent some time trying to parameterise and sort out the code to make it more readable and understandable

added banner, adjusted border, rebuilt drawcon logic to allow multiple items to be drawn

tried to sort out sprites but mario is doing some trippy stuff

16/04/23
found the required dimensions for. it scrolls when moving, which suggests that there may be something faulty with the draw controller overall, or the vga logic.
With everything i've learnt so far about verilog and the nexys7, i'm going to try rebuilding the controllers in accordance with vga standards and in systemverilog since the company that's hired me seems to use it.

https://www.verilogpro.com/systemverilog-always_comb-always_ff/

https://www.verilogpro.com/verilog-reg-verilog-wire-systemverilog-logic/

i've found myself frustrated with verilog's limitations on data types and oop. it's like using C after being somewhat familiar with C++; rather frustrating stuff.

https://www.sutherland-hdl.com/papers/2006-SNUG-Boston_standard_gotchas_paper.pdf - very useful paper on tricky bits of the language

17/04/23
rewrite is going to be too long a time sink, even if the extra functionality makes writing easier.
sticking with verilog old.
I emailed Eduardo last night for some help on the scrolling issue, and I think I've got it now. I've given the sprite a reference point at the very start of it's draw to reset the address, so that the sprite is always drawn relative to the reference point and now doesn't skip lines (thus scroll) when horizontal lines are redrawn. I still see some skipping occur, so it looks like the bottom of the sprite 'blips', however this gets overridden fairly quickly with the correct data, so I think I can ignore it for now. It's 'good enough' essentially, and this only appears when moving the sprite quickly.

I'll now be working on SPI to get accelerometer data. I don't entirely understand it all yet, but it looks quite challenging.

accel module implemented by adapting an existing project. code linked:

https://github.com/FPGADude/Digital-Design/tree/main/FPGA%20Projects/Nexys%20A7%203-Axis%20Accelerometer%20SPI

drew a bounding box for the player controller. lines map across screen. changed drawing logic to draw a large box then a smaller one inside. reordered drawing logic so that the largest box is drawn after the small box after the player. gives the impression of a player border now. parameterised all variables so that they can be updated when i get the correct sizing. Parameterised bounding box so player goes right up to edge. Required changing movement speed to 1px.

attempted to move player-movement to accel. accel returns 5 bits of data for x, y, z. z not necessary.

flat position in x and y (either 0 or 31 due to non-flat bottom) returns 0 acceleration for player.
as acl value increases, the value is directly added to the current position value and updated. this works for values 0...15, however 16...31 need to be subtracted from 16 to return negative values (i.e. move backwards). x values appear inverted, and therefore are negated. discovered here that verilog case statements do not support ranges, so wrote this in if-else instead.

to bound player movement, current updated position is checked against parameterised bounds. if greater, position is set the bound, else position is set to update as normal. this ordering stops the player block vibrating out the bounds, as checking the bound after update causes this.

next step for tomorrow is to move the player box to the centre of the drawn screen, and provide a colour gradient for the player to move on. the player block needs to be replaced with a sprite with a pointer. creating 'transparency' will be problematic.

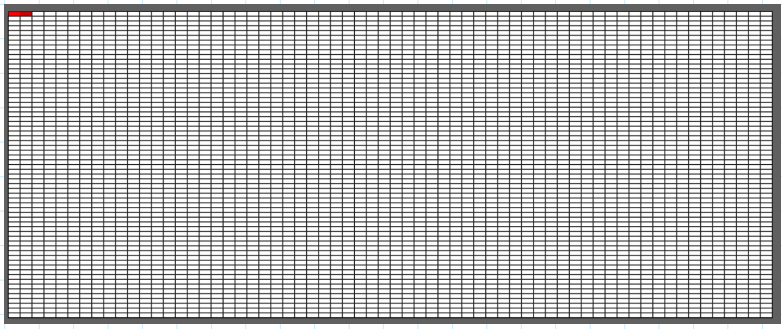
20/04/23
in further communications with Eduardo regarding the acl, he linked the digilent github page for the acl, written in vhdl. I tried implementing this briefly, overwriting David's one. It didn't immediately work, and it's likely because a supervisor file that I didn't include for interpreting the data is needed. With some work, I could get it all to fit together, however with a working acl module already, this would be a waste of time.

I am drawing out dimensions for the gui elements. this will help me formalise sizing, and move the player box into the center after creating the divisions. i'll aim to keep things parameterised so i can include difficulty levels for the game if i get time. it also helps

with keeping the code managed in any case.

sizing the elements correctly was proving difficult, so programmed in a 16x16 grid. took some fiddling to get the parameters to sit correctly. there's still one pixel dropped on the left side for some reason.

considering ways to create the colour selection area. started making a coloured grid to use as a sprite, however manually colouring would be tedious. i got two colours in...
approaching it from a programmatic angle instead.



i'm currently facing the rolling colour problem like i did with the sprite. i'm trying to adjust the width to get static vertical bands across the gradient, but no such luck.
I'll also need to consider creating a module so the correct value can always be calculated and returned, regardless of whether it's being drawn over by the player sprite (which causes additional wrapping issues at the moment).

no value for the width change seemed to work...

21/04/23
i'm thinking that creating a module for the colour picker will allow for a bit of separation of intents. I can use the colour output in the two places i need - the colour picker itself, and 'transparency' of the sprite. this also helps clean up the logic of drawcon a bit so it isn't a massive mess.
I'm thinking of having counters like with the vga controller so that it always outputs the right colour given coordinate position, regardless of whether it's overwritten by the sprite.

moving the logic to a separate module now means the player sprite does not interfere with the draw, however the colours are still scrolling... good news is that there are distinct columns at least, however it's also got a triangular appearance.

stripping out the vertical logic, now the columns are appearing fine, although they seem to repeat, which suggests my maths is wrong. I seem to get almost 4 columns of almost repeating colour. they also still appear to have banding between them.
I think it might be best to stick this in a testbench and figure out where the numbers are going wrong.

I doubled the cell width and I got 2 repeated columns. it only looks like 18 gradients of colour, though. I predict the vertical banding is similarly due to cell size issues.

the similuation doesn't want to launch for some reason. trying with hardcoding some spacing values to make sure the dimensions are correct.

[Labtools 27-1832] create_wave_config not a supported tcl command in labtools hardware mode. Default wave configurations are automatically created when triggering an ILA core.

the location is now severely misplaced, but I think I understand the colouring issue.
on the horizontal, the extra red bands are a result of the colour chang overflowing past magenta. why it repeats back again after that is still unknown. the bands horizontal lines will be a result of something going wrong with how the y change is handled, since the colour runs constant instead of changing. this suggests skipping.

the horizontal banding is due to how colour values tick up.
if the colour starts at 000, and each x cell adds 1, at 00F, the next value is 010, which appears black, producing the 'band'.
Slowly per band, the green levels increment and become more visible. now, though, there's some weird unevenness going on and i'm not sure why. plus it appears like it's scrolling again.

it may be better to approach this with a bitmap of colour values for it to read off. this will help with the random colour generation logic later, as there's only a set of colours it can read from.

the horizontal banding might be due to the code resetting colour to 'start_colour' taking precedence to the gradienting code across x. we'll see if switching it fixes it.

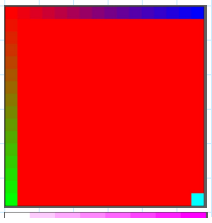
i forgot the existence of for loops since i was treating the overall hardware loop as where i needed to aim the logic. I'm now attempting to get the drawing logic to draw in a for loop.

i don't think a for loop will work, since this doesn't work on the same scale as a higher level programming language. i think the loop only works for simultaneous assignments, and not something that runs over multiple clock cycles.

bitmapping currently results in some interesting colours patterns, but it's not yet stable.

this is proving to be perhaps the most difficult part. It's taking way longer than expected. I'm trying a lot of things and I feel like I'm inching towards a solution but it's still out of grasp.

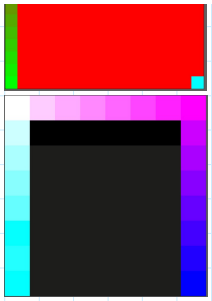
23/04/23
i've tried a lot of programmatic methods but i honestly cannot seem to get anything to work. I think I'll just hardcode it and go from there. I think I'll reduce the colour map to 16x16 and draw it as a square sprite.



in the end, it'll look something like this:
i think i can program a python script to produce this output as a .coe

technically i made a script however it goes over bounds with the logic. it turns out the colour space transformations i'm aiming for

in the end, it'll look something like this:
i think i can program a python script to produce this output as a .coe



technically i made a script however it goes over bounds with the logic. it turns out the colour space transformations i'm aiming for (red to blue to green) cannot be created in the way that I want.
I can go rgb to cmy, with white in the bottom right.
I'll need to rethink how the colours transform in any case.

I created a new 8x8 colour space, which I'll now try to program a script for. i've got a bitmap and a .coe to work with now.

apparently 256x256 pixels is out of ram size. I'll have to go for the encoded version in which case. Making it 128x128 worked, however. implementation of the blockmem fails due to unconnected pin 'ena'. not sure what's happening here. back to the drawing board it looks like...

i'll build it up from basic hardcoding - i'll bound the drawing box with absolute numbers like I have for other features, and then consider improving it programmatically.
Worst case, I'll get python to spit me out a hardcoded version.

i've gotten gradients along y working, although there's a weird dip in the line for some reason. I believe the dip is before the hardcoded draw start is.

i now have a gradient running vertically downwards. now i need to figure out the logic to break each row into its colours. i'll start by creating a start_pointer to point to the 'row colour', and increment the colour pointer of there. to test it works, i'll modify the logic to include this and hopefully the cyan gradient should show instead of the magenta one.

looking at the hardcoded logic, the most sensible method of encoding this would be a for loop.
while y is in a band of 16, check the x band and assign the appropriate colour. however, i struggle to get for loops to work so i'm not sure i'll even try this method

hardcoding worked! it looks disgusting and i'm horrified as a programmer to have written this (even if i scripted the writing) but it's the most success i've had yet. it seems to have a border around it, although i think i can remove that by squeezing the sizing values a little.

gradient is working, and so is colour pointing in a janky manner. I'm going to draw and load in a sprite for the pointing reticle now.

i now have selected colour logic working. i need to integrate this with the sprite now to give 'transparency'

while generating the new sprite, i realised the issue with the colour gradient sprite was that i didn't have the Enable Port Type to Always, which is why it wasn't finding the pin it expected.

sprite currently draws as a solid square for some reason. turns out i was using the wrong mem block.

now it's generated, it's cut off the side a bit. feels like dealing with mario all over again. this is proving to be annoying again...

after fiddling around with the sprite layout itself and some of the code, it's finally looking centralised. It's still broken, but it'll do for now. I'll focus on other things.

the transparency is doing something extremely strange, but that may be to do witht he case statements somehow?? I'll switch it back and see ifit fixes itself.
...which it did. rather annoying but the order of a single line seems to have made the difference.

transparency works although the sprite is still jank so i may need to drop it :(

now to work on game logic proper. i'll need to look into randomness probably. let's fill the grid with colours first. the easiest method may very well be to just hardcode it again...

attempting to launch a simulation to see if i can generate random numbers. it doesn't seem tohave failed like before - it seems i've fixed whatever io problems were killing the last sims.

looks like the random number gen is good to go. that was a lot easier to implement than i expected. I know of another group using the temp sensor as a random number gen seed and i'm glad i don't need to go to such extents.

to simplify things, i'm going to move the colourspace out of the colour_picker and into drawcon, so colour_grid can use it also without having multiple copies. now i only need to output the pointer.

in the sim, the pointer is XX for a long period of time. i think this is due to the random number array not filling out quite right.
when hardcoding in a 'random' array, the code now works fine. the problem is clearly with the array. i'll need to go over random generation again.

for now, i'll work with the hardcoded grid and get the game logic sorted before moving onto making it gamelike.

i think the way to approach this is with a 2d array that records whether a cells is connected. i'm having to rebuild how colour

assignment works a bit to work with this new 2d array system. so far no errors.

an error regarding combinatorial loops and race conditions came up during bitstream generation.

can't seem to get the grid generation code to really stick. will review tomorrow.

25/04/23
Thomas has been in contact. Will arrange a meeting to see what he can contribute.

In the meantime, i'm hardcoding in the colour changing and observing the effect on completely disconnected cells. This will let me determine whether colour changing is possible with the current logic of the 2D array. I just realised, but it'll also help me determine that the grid is even working as expected. It's possible I've referenced the cells incorrectly.

I've just realised a major error - initial begin blocks are not synthesisable and haven't been running this entire time! pretty dumb error, but hopefully progress will be better now.
in further research, it appears that xilinx boards *will* synthesize the initial block, so it was never a problem. It also explains why it was working when it supposedly should not have been.

The problem now is that the pointer_array that stores the colour the block should be does not appear to work. 'current_pointer' appears to assign only the maximum value (63) to the cell and does not appear to dynamically update. this may be due to verilog not handling 2d arrays well. I'll try changing the filetype to system verilog, which may make a difference.

I added a temporary register to hold the selected value from the picker, in drawcon. this is then passed to the grid gen. Now colour selection works. the systemverilog change wasn't necessary.

I've gotten one cell to follow the selected colour by hardcoding in the references. I've added code to check the surrounding cells and hopefully update them to follow if they match. If this works, I can then spread the logic to work programmatically with counter variables. The connecting logic is a bit janky and doesn't seem to work as expected. This will take more time to figure out.

26/04/23
performed a code handover with detailed explanations of every element of the game to thomas so that he is able to take over the project from hereon.

27/04/23
provided thomas a document on xilinx's lfsr model composer block, which will likely be required for the random grid generation.

created a logo for the game

04/05/23
going to work on improving the codebase, primarily by switching hardcoded logic into for loops.
currently it just produced a black screen in my attempt to recode the colourspace into a synthesisable loop. this may be because it has to be in a sequential block and the initial begin statement won't suffice.
also moved the title sprite.
fixed the adjacency logic so the game plays as intended now, although it's added some weird artefacts across the cells and since the compilations are taking an absurd amount of time i'm not sure how much i can try to debug that issue.

05/05/23
can't seem to remove the flickering coloured lines.
now working on implementing for loops for the other hardcode logic.

added parameterisation to cell generation

- thomas is still working on implementing the timer, and will be merging the codebase of the two projects

09/05/23
made a video to accompany thomas'.
dug through to make some bug fixes and vastly improved the graphical quality, which until now had been flickering severely.