

1 Introduction to VGA

VGA (Video Graphics Array) is a visual graphics standard introduced in 1987. It was developed to work with CRT (Cathode-Ray Tube) technology, which works by drawing each pixel in horizontal lines vertically down the display to create a *frame*.

In order to produce the effect of a static image given only a single pixel controlled per instant, the entire frame would be redrawn (or ‘refreshed’) quicker than the eye can perceive. A refresh rate of 60 Hz would be typical.

Due to the physical deflection of an electron beam to produce pixels, extra time for the beam to adjust to a new line is included in the standard, called the *blanking interval*. The start of a new line is controlled by the horizontal sync (hsync) signal, with a new frame in turn controlled by a vertical sync (vsync) signal. The blanking intervals before and after these signals are respectively called the *front porch* and *back porch*. In modern standards like HDMI, this extra time has been retained and instead carries extra data like accompanying audio.

1.1 Driving the display

There are four requirements for driving a VGA display:

1. Clock signal driving the display signal counters.
2. Horizontal and vertical signals, defining pixels.
3. Drawing graphical elements to the screen.
4. Physical video output to a display.

1.1.1 Clock

The required display frequency clock can be calculated by:

$$f_{\text{clock}} = L_h \times L_v \times f_r$$

where: L_h = horizontal lines, 1903

L_v = vertical lines, 931

f_r = refresh frequency, 60 Hz

The results in a required clock frequency of 106.30158 MHz. The easiest method of producing clock signals in Verilog is to divide down the main processor clock to the required subdivision. In the case of the Nexys A7, the onboard clock is 100 MHz and thus cannot be divided down to a value that is greater. However, by using the Clocking Wizard IP provided by Xilinx, this can be produced. The caveat is that it will not be necessarily exact. In this case, the actual output clock is 106.296 MHz, which is within 0.005 %.

1.1.2 Signals

In this implementation, the horizontal and vertical signals are controlled by counters operating at the overall display frequency, determining which pixel is being drawn. The counter values then determine the widths of each signal parameter. The parameters are outlined in table 1, which informs the display dimensions as shown in fig. 1.

	Sync signal	Back porch	Display	Front porch
Horizontal	151	384	1824	1903
Vertical	2	30	930	931

Table 1: VGA signal parameters

The visible portion produces a display resolution of 1440x900.

1.1.3 Drawing graphics

Graphics drawing is delegated to its own module¹ separate from the VGA controller. To provide an overview, the graphics drawing module (draw controller) takes as input the current pixel by coordinates in x and y with reference to the top left corner of the visible display. It outputs the colour for this pixel as a 12-bit hexadecimal number, which can then be passed to the VGA port.

¹Section 2.8.

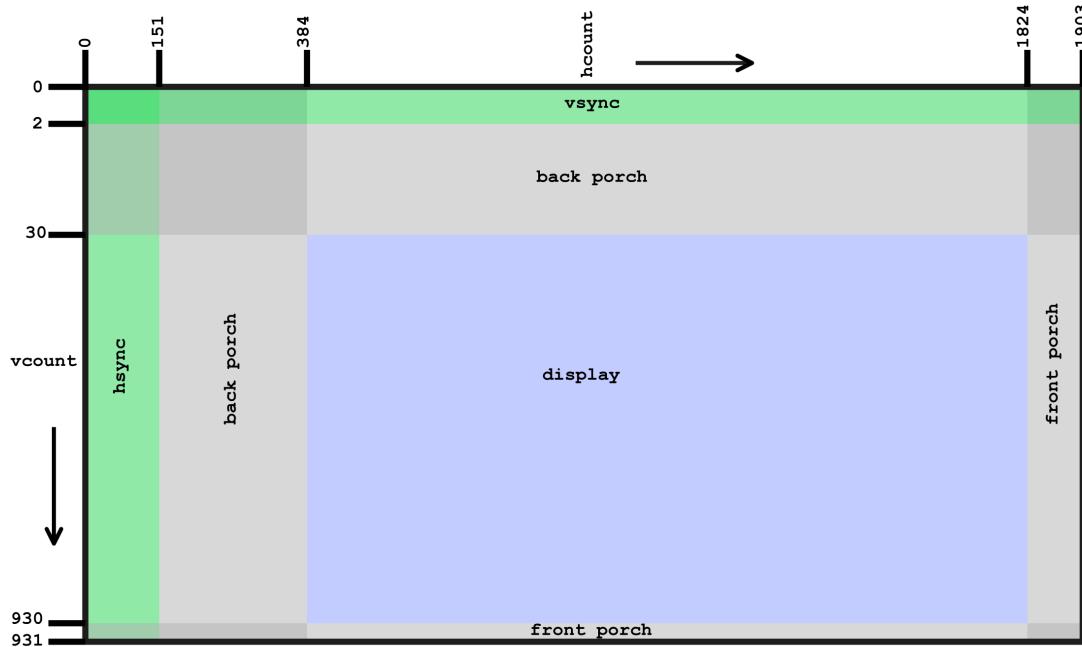


Figure 1: Display parameters

1.1.4 Video output

The specifics of the output (i.e. required voltages, pin connections etc. to the D-sub connector) is abstracted away in the constraints file for the board. In this file, the pins of the Nexys A7 with regard to the VGA port are provided and simply need to be included in the compilation to be used.

The colour channels in the port are different physical connections, taking up six pins of the connector - one each for red, green, and blue, in addition to each colour's ground reference. In the implementation for this project, the 4-bit output colour variables (`VGA_R`, `VGA_G`, `VGA_B`) have been merged into a single 12-bit variable (`colour`).

The mapping occurs as follows:

Original	<code>VGA_R[3:0]</code>	<code>VGA_G[3:0]</code>	<code>VGA_B[3:0]</code>
Implementation	<code>colour[11:8]</code>	<code>colour[7:4]</code>	<code>colour[3:0]</code>

The advantage of this is that colour assignment to a pixel can now be encoded as a single hexadecimal number with three digits - the first encoding the red value, the second green, and the third blue. This is particularly useful as the encoding now follows web-safe hexadecimal colour space², reducing the guesswork in developing a colour on screen.

1.2 Limitations

With VGA as the only display adapter available on the board, the colours available is limited to 4-bits for the red, green, and blue channels each. This results in 4096 total colours. Most modern monitor displays, on the other hand, permit over 16 million colours due to the use of 24-bit colour space.

1.3 Implementation

The implementation of the VGA controller can be found in listing 6.

2 The game

2.1 Gameplay

The player controls a reticle in a bounded box with a colour gradient, which is placed on a grid of colours. The player selects a colour within the gradient by tilting the board until the reticle is positioned above the colour to be selected. The aim is to select colours in order to clear the grid as quickly as possible. When a cell in the grid is selected ('matched'), it follows the reticle in its colour selection. As more colours are selected, a

²A palette reference available at [RapidTables](#).

greater portion of the screen is matched until the background is a solid colour. To begin with, only the cells surrounding the central selector can be matched, however colours can then be matched to any successive cells that have also been matched, ‘flooding’ from the central region.

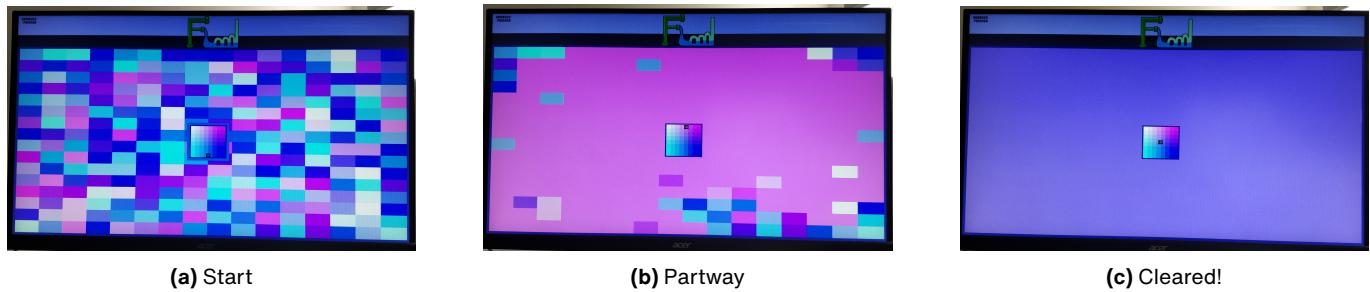


Figure 2: Gameplay states

2.2 Module overview

The structure of the program follows a hierarchical design, with module `game_top` as the parent module. The relationships and data connections between modules is demonstrated in fig. 3. This module specifies many parameters that are carried over to child modules.

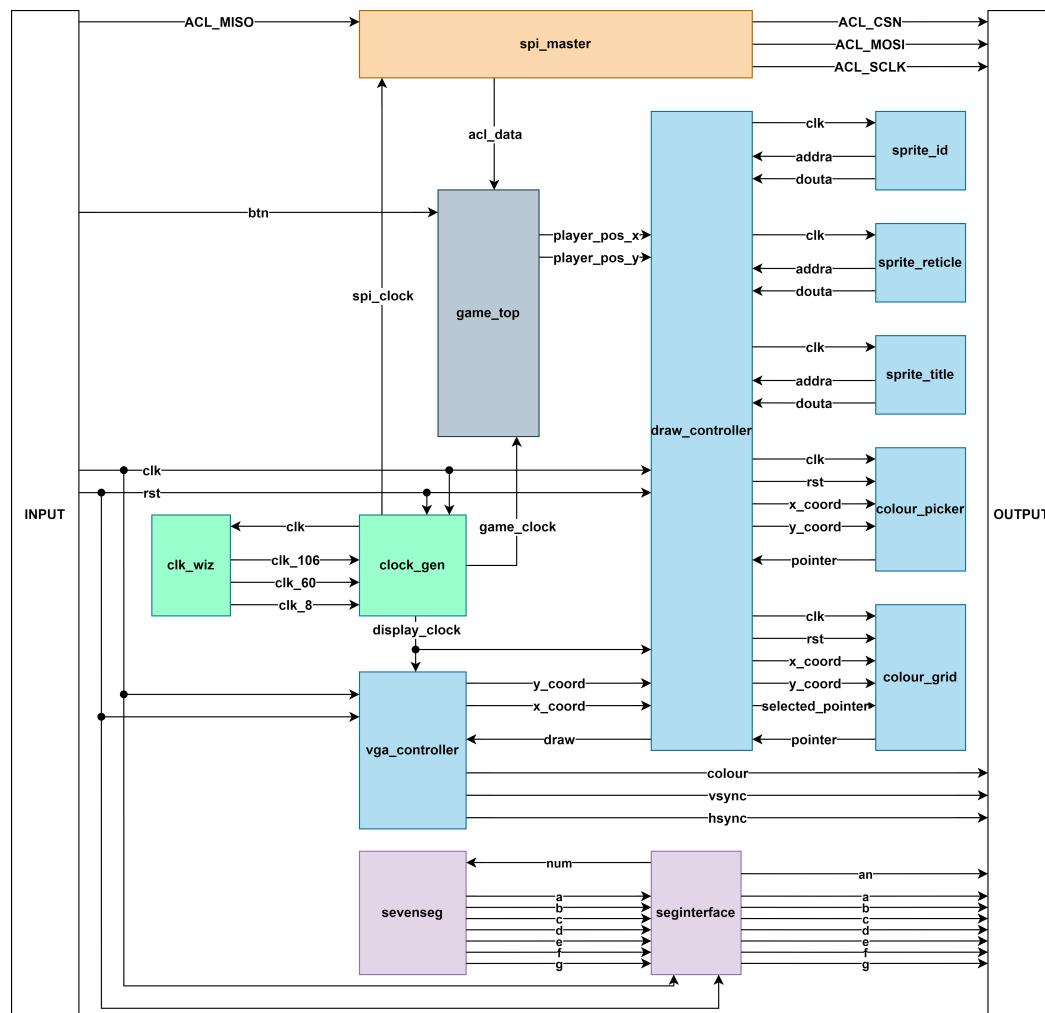


Figure 3: Relationship diagram

The following sections discuss the constituent modules in detail.

2.3 `game_top`

Code available in listing 1.

This module is the topmost module, and primarily handles connections both to the board's peripherals from modules, and between modules themselves. It additionally controls movement logic for the player sprite³.

The accelerometer controller⁴ outputs a single 15-bit string of data, `acl_data`, encoding five bits of x , y , and z orientation data each. For the purposes of this project, only the x and y data are needed, with the tilt directions indicated in fig. 4. The relevant data can be accessed with a bit-slice `acl_data[9:5]` for x and `acl_data[14:10]` for y .

The 5-bits of returned data indicates the level of tilt, with 0 meaning level, 15 meaning a 90° rotation anticlockwise/left in x and clockwise/forward in y , and 31 meaning an almost complete rotation. In order to adapt this to control movement, these values need to be split and adjusted. Regarding horizontal movement of the player sprite, a value of 0-15 can be added to `player_pos_x` to update position, producing a translation to the right. A negative value produces a translation left. Tilting the board left returns 0-15, so this simply needs to be negated from the player position to produce the desired translation. Tilting the board right returns 31-16, so first a subtraction from 31 is performed, and then the value is added to position⁵.

In order to bound the player movement, so it does not move out of the colour selection area, two changes are needed. First, the bounding parameters, i.e. the maximum and minimum coordinates the player can move to, can be specified in parameters. Then, each time the player position is to update, the current position in addition to the updated position is compared to the bounding parameter. If greater, the current position is set to the bounding parameter⁶. This maintains a smooth collision with the bounding walls.

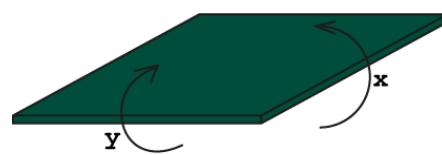


Figure 4: Accelerometer tilt influence

2.4 spi_master

This module is used to communicate with the ADXL362 [1] accelerometer, which is exposed on the SPI interface. The code in this module is written by David Marion as part of his experimentation with FPGAs and is available on his GitHub [2].

The module collects data from the accelerometer and returns a 15-bit number encoding x , y , and z acceleration data, which has been adapted in `game_top`⁷ to produce movement of the player sprite.

2.5 clock_gen

Code available in listing 2.

Different parts of the game have different clock requirements. The game clock, for example, runs at 60 Hz. The following clocks are produced by the `clock_gen` module:

Primary clock	100 MHz
Display clock	106.3 MHz
Game clock	60 Hz
SPI clock	4 MHz

Many of these clocks are produced by the clocking wizard IP, and then divided down - by 1M in the case of Game clock; by 2 in the case of SPI clock. The reason for the division is due to the minimum clock producible by the IP is 4.687 MHz. The display clock, on the other hand, is passed on directly.

2.5.1 clk_wiz

This IP is provided for Xilinx boards and can be used to produce different clocks between 800.0 MHz and 4.687 MHz. It is used to produce three clocks used in the program, however there is a small amount of discrepancy between what is requested and what can be produced.

Name	Requested	Actual	Discrepancy
clk_106	106.30158 MHz	106.296 MHz	0.00525 %
clk_60	60 Hz	59.792 MHz	0.347 %
clk_8	8 MHz	7.972 MHz	0.350 %

Table 2: Clocking wizard clocks

³Section 2.8.1.

⁴Section 2.4.

⁵Lines 55-70.

⁶Lines 72-88.

⁷Section 2.3.

2.6 vga_controller

Code available in listing 6.

This has been explained in some detail in the introductory section 1, with some explanation of the approach for implementation in section 1.1. The specifics of the implementation involve using parameterised values to set the bounds for the display and line counters.

Broadly, the counters `hcount` and `vcount` are used to count the horizontal and vertical lines (and thereby the specific pixel) and set its colour (e.g. `0x00F` for blue) to the input, draw, within the `display_region`, or `COL_BLACK` (`0x000`) outside it. Between 0 and `HSYNC_START`, the `hsync` signal is sent to the VGA port. (The same follows for `vsync`).

2.7 seginterface

Code available in listing 7.

Strobes the LED interface and sets the digit to each display.

2.7.1 sevenseg

Code available in listing 8.

Defines the digits displayable on the interface by a binary sequence representing which LED segment is lit.

2.8 draw_controller

Code available in listing 3.

As the most complex module written for this project, this module controls all graphical elements on the screen. It takes as input coordinates for the current drawing pixel and player, and outputs the colour of the pixel to be drawn.

Subsequent sections (2.8.4 and 2.8.5) rely on a specific set of colours defined as a colour space, which is a 64-index array of 12-bit colour values. Following the progression that can be seen in fig. 5, the top left corner starts with a colour value of `0xFFFF` (white) and reduces by 2 in green horizontally (to magenta, `0xF0F`) and by 2 in red vertically (to cyan, `0x0FF`). This leaves the bottom right most value as all blue (`0x00F`). This is encoded sequentially, with a snippet of the corner values of fig. 5 shown in table 3.

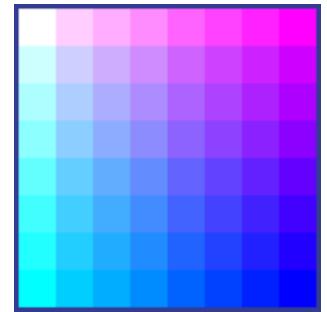


Figure 5: Gradient model

Array index	Hexadecimal value	Colour	
0	0xFFFF	White	█
7	0xF0F	Magenta	█
56	0xF0F	Cyan	█
63	0xF0F	Blue	█

Table 3: colourspace encoding

Generation of the colour space occurs in a FOR loop, using the subtraction of `0x020` from `0xFFFF`⁸.

Subsequently, the elements to appear on-screen are drawn in hierarchical order. Order is determined by a conditional tree that evaluates when the coordinates of the pixel to be drawn are within the space defined for the element. The following snippet illustrates this idea, where the banner colour is drawn at the top of the screen, otherwise the background colour is drawn.

```

1   if (y_coord < BANNER_HEIGHT) begin
2       pixel <= COL_BANNER;
3   end else begin
4       pixel <= COL_BACKGROUND;
5   end

```

The following elements are drawn in order of hierarchy:

1. ID sprite
2. Banner
3. Title sprite

⁸Lines 66-73 of `draw_controller`. Details of implementation can be found in section 3.

4. Screen border
5. Player reticle sprite
6. Gradient colour picker
7. Picker selection area border
8. Picker border
9. Grid background

2.8.1 sprite_reticle

The ‘player’ sprite appears as a controllable reticle intended to point at the colour selected. The sprite was drawn in Microsoft Paint and converted from its base format (.bmp) to space delimited memory encoding (.coe) that can be used by Xilinx’s block memory generator IP to write the sprite into ROM.

Due to what is most likely issues with reading the sprite from memory, the image has been adjusted from fig. 6a to fig. 6b to account for ‘wrapping’ that appears to occur.

As a controllable object, additional complications are introduced as the sprite moves or generally interacts with the drawing logic. A pixel to be written from memory is skipped, leading to the appearance of ‘scrolling’. This is fixed by ensuring the first address from ROM is to be written after either the last address has been reached, or the current drawing coordinates (`x_coord` and `y_coord`) coincide with the current position of the sprite (`player_pos_x` and `player_pos_y`, indicating the top left corner)⁹.

A final feature of the sprite is the red (0xFF00) pixels in the centre is used to determine the colour ‘selected’. When drawing the sprite, if the pixel being drawn is a red pixel, the pointer returned by `gradient_pointer` from `colour_picker`¹⁰ is recorded into `selected_pointer` and the colour value held in `col_selected` in order to change the control area border colour. A similar technique is used to provide ‘transparency’ for the white portions of the sprite, simply by overriding the pixel colour with `colourspace[gradient_pointer]`¹¹.

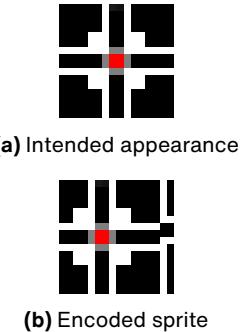


Figure 6: Player sprites

2.8.2 sprite_title

A sprite for the title was created in Adobe Photoshop owing to its increased complexity in detail. The background colours were selected to fit seamlessly with the top banner of the game.

It was rendered into the game from a ROM similar to the player sprite.



Figure 7: Title sprite

2.8.3 sprite_title

Produces the ID values 2008503 and 1922268 in the top left corner.

2.8.4 colour_picker

Code available in listing 4.

This module creates the central colour selection region for the player, outputting a pointer for current pixel to be drawn.

The initial idea was to create the selection colour space as a sprite, leading to the creation of fig. 5 in Adobe Illustrator to be imported as a ROM. The sprite approach was changed as it consumes a large amount of memory¹². A programmatic approach, using a FOR loop is used instead¹³. In this case, the 8x8 grid is generated from a nested FOR loop; the outer loop specifies the height of the grid cells and the inner loop the width of the cell. The colour is then selected by a sequential counter pointing to `colourspace`.

2.8.5 colour_grid

Code available in listing 5.

This module works similar to `colour_picker` in producing a grid from a nested FOR loop, however it has to deal with the extra complexity of dynamically updating the colour of its cells. This leads to some artefacts

⁹Lines 110-112 of `draw_controller`.

¹⁰Section 2.8.4.

¹¹Lines 114-121 of `draw_controller`.

¹²The first version with 256 16x16 pixel cells required more memory than was available on the FPGA.

¹³Lines 35-41.

appearing as this logic interacts with the display adapter. The general logic of the game in terms of ‘flooding’ is handled in this module. It receives the current colour pointer as input and outputs a pointer for the colour of the current background pixel.

Two 16×16 2D arrays are used in this module¹⁴:

- connected_array – Holds a value to describe the state of the cell.
 - 2** The cell is ‘connected’ and changes colour with the selection from colour_picker.
 - 1** The cell is adjacent to a ‘connected’ cell, and needs to be checked if it matches the selection.
 - 0** The cell is in its default state.
- pointer_array – Holds the pointer to the colour for the cell. Starts ‘randomised’¹⁵.

Every clock cycle, a single cell is analysed, with the entire grid checked over a million times in a single game frame. The incremental order is horizontally, then vertically. If the cell is adjacent (value 1), the colour is checked. If matched, the colour of the cell is updated to the currently selected colour (via pointer_array), its connected_array value updated to 2, and each cell adjacent is updated to have a value of 1 if currently 0. Otherwise, if the value is 2, the colour is simply updated¹⁶.

2.9 Unimplemented features

2.9.1 FOR loops

In some instances, logic that had been hardcoded was successfully converted into a much easier to read and maintainable FOR loop. However, this is not the case in every instance. While the creation of the colour space in draw_controller¹⁷ was successfully converted, the same cannot be said for the value assignments to connected_array in colour_grid. The result, in the instances adjacency worked, appeared to match cells somewhat randomly or led to some cells failing to change colour.

2.9.2 Difficulty modes

Different levels of gameplay difficulty, where the user selects an option that dictates the grid size (i.e. 8^2 rather than 16^2 - the current default) would have been implemented leveraging the dynamic nature of the grid generation if handled in a loop. In that case, the loop would be built into a conditional, where the selected grid size would inform the variables used to define each cell’s width and height. Even the number of colours could be adjusted similarly to lower the difficulty in discerning colours.

2.9.3 Randomisation

The grid was intended to be generated randomly for each game. Generating a string of random numbers is possible with a Linear-Feedback Shift Register (LFSR), which produces a deterministic string of random numbers. The sequence is dependent on the initial input (seed), which itself would require true random generation in some manner. A possibility for this is to take a reading from the board temperature sensor.

2.9.4 Timer and completion

The idea of ‘filling the grid as quickly as possible’ falters at the point where the game lacks a timer for the user to see how quickly they have completed the board. clock_gen would need to output a clock for seconds, incrementing a counter on every positive edge. Then, the program can read from a numeral sprite map to render digits on screen that show the current decimal value of the counter.

Completion can be determined by summing connected_array. A cell is matched when its value is 2. For a 256 cell array, if the sum equals 512, the game is finished. Extending this idea further allows showing the current percentage of completion.

3 Testing and verification process

Testing designs before compiling them becomes ever more important as complexity scales. In the later stages of the project, a single compilation could take as much as ten minutes, highly restricting iterative testing methodologies like divide and conquer¹⁸. Therefore, the importance of testing via simulations is emphasised.

¹⁴The file type was changed to SystemVerilog as multidimensional and unpacked arrays are not supported by Verilog.

¹⁵Section 2.9.3.

¹⁶Lines 548-576.

¹⁷Section 2.8.

¹⁸A methodology where debugging statements are printed to console to narrow down the source of a problem. Highly reliant on running many test suites in quick succession.

To illustrate this process, an example will be outlined. Following an agile-like development methodology for this project prioritised producing functional code as quickly as possible. For static but repeatable elements, this means a large amount of hardcoded software, making the code unwieldy and increasing the difficulty maintaining the codebase. To improve this, the logic can be synthesised from a FOR loop instead.

3.1 The hardcore

```

1 initial begin
2     colourspace[0]  <= 12'hfff;
3     colourspace[1]  <= 12'hfdf;
4     colourspace[2]  <= 12'hfbf;
5     ...
6     colourspace[7]  <= 12'f1f;
7     colourspace[8]  <= 12'dff;
8     ...
9     colourspace[63] <= 12'h11f
10 end

```

This was generated from the following Python script (modified slightly for clarity):

```
1 r = 15
2 g = 15
3 b = 15
4 for count in range(0,64):
5     print(f"colorspace[{count}] <= 12'h{r:x}{g:x}{b:x};")
6     g = g - 2
7     if(g < 0):
8         r = r - 2
9         g = 15
```

3.2 Rewrite in Verilog

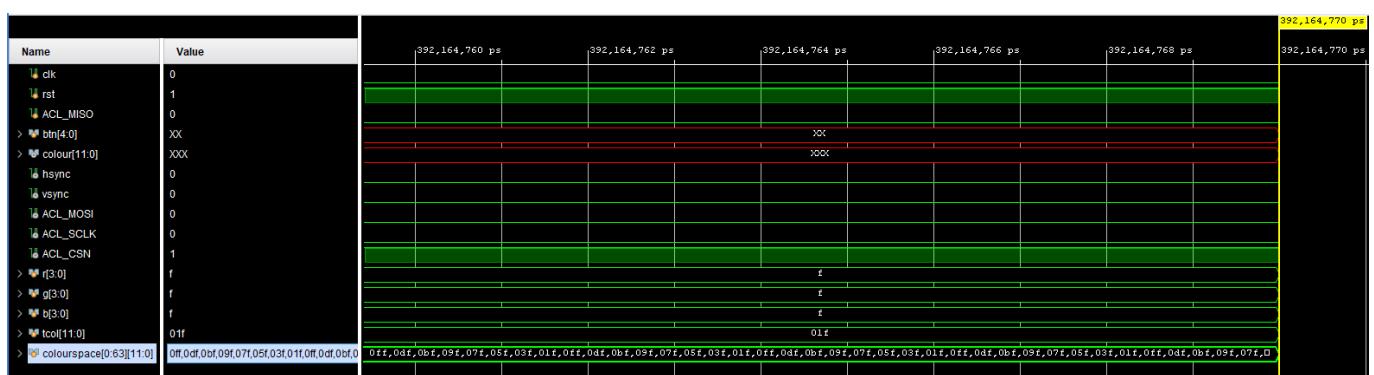
The hardcoded logic can be synthesised from a FOR loop, written in Verilog directly in this case. This is almost a direct translation from the Python script to Verilog, however there are a number of differences. Since the code does not need to be printed to be copied, that stage is omitted entirely. Instead, the assignment of the colour to colourspace is made directly through an intermediary variable `tcol`, which is made of the `r`, `g`, and `b` bitshifted to their respective places.

```

1   for(i=0;i<64;i=i+1)begin
2       tcol = r<<8;
3       tcol = tcol + g<<4;
4       tcol = tcol + b;
5       colourspace[i] = tco;
6       g = g - 2;
7       if(g < 1) begin
8           r = r - 2;
9           g = 4'hf;
10      end
11  end

```

This can then be run in a test bench to see the resultant values of tcol, r, g, b, and colourspace.



The resulting colour values are not quite correct, with the red value stuck at 0.

```

1 for(i=0;i<64;i=i+1)begin
2     colourspace[i] = r<<8 + g<<4 + b;
3     g = g - 4'h2;
4     if(g < 4'h1) begin
5         r = r - 4'h2;
6         g = 4'hf;
7     end
8 end

```

> colourspace[0:63][11:0]	000,000,000,000,000,000,000,000,000,000,
> r[3:0]	f
> g[3:0]	f
> b[3:0]	f

This attempts to perform the bit-shift in a single line, however the end result is all 0. The intermediary variable is necessary.

```

1 for(i=0;i<64;i=i+1)begin
2     tcol = tcol + g<<4;
3     tcol = tcol + b;
4     colourspace[i] = tcol
5     g = g - 4'h2;
6     if(g < 4'h1) begin
7         r = r - 4'h2;
8         g = 4'hf;
9     end
10 end

```

> colourspace[0:63][11:0]	1ef,fcf,daf,b8f,96f,74f,52f,30f,1ef,fcf,daf,b
> r[3:0]	f
> g[3:0]	f
> b[3:0]	f

This indicates the red bit-shift is always returning 0 and non-functional. The gradient produced is also not what is intended. It may be better to move to subtracting a constant value.

```
1 for(i=0;i<64;i=i+1)begin  
2     colourspace[i] = tcol;  
3     tcol = tcol - 12'h020;  
4     if(tcol[7:4] < 4'h1) begin  
5         tcol = tcol - 12'h200;  
6     end  
7 end
```

>  colourspace[0:63][11:0] fff,fd,fb,f9,f,f7,f,f5,f,f3,f,f1,f,eff,edf,ebf,e9f,e7f

This subtractive method appears better, however the aim is to drop the red value by two, whereas here it only drops by one. The loop in an initial block does not appear to be synthesised correctly for actual implementation, resulting in a black screen.

```
1 for(i=0;i<64;i=i+1)begin  
2     colourspace[i] = tcol;  
3     tcol = tcol - 12'h020;  
4     if(tcol[7:4] < 4'h1) begin  
5         tcol = tcol - 12'h500;  
6     end  
7 end
```

colourspace[0:63][11:0]	fff,fdf,fbf,f9f,f7f,f5f,f3f,f1f,eff,edf,ebf,e9f,e7f
> [0][11:0]	fff
> [1][11:0]	fdf
> [2][11:0]	fbf
> [3][11:0]	f9f
> [4][11:0]	f7f

The code was moved to an always block to allow it to be synthesised to the display correctly. Furthermore, the reduction was increased. The test bench shows the value stay the same - the IF statement is not evaluating true since `tcol[7:4]` will move from 1 to F.

```

1 for(i=0;i<64;i=i+1)begin
2     colourspace[i] = tcol;
3     tcol = tcol - 12'h020;
4     if(tcol[7:4] == 4'hf) begin
5         tcol = tcol - 12'h100;
6     end
7 end

```

colourspace[0:63][11:0]	
> [0][11:0]	fff
> [1][11:0]	fdf
> [2][11:0]	fbf
> [3][11:0]	f9f
> [4][11:0]	f7f

The values are now reducing in the correct progression.

4 Conclusion

4.1 Evaluation

The designed game will be evaluated against the criteria.

User control The user takes control of a reticle to select a colour, which is controllable by tilting the board in the desired movement direction. The sprite is bounded within the selection area.

Object interactions The game progresses by clearing the screen by progressively selecting colours. The grid items need to effectively interact with each other in order to determine adjacency and when a grid has been matched to the selected colour. However, some visual artefacts may occur. The player sprite also interacts with the selection grid underneath to select a colour and assumes the characteristics of transparency.

Info bar An info bar is drawn from the graphics controller as a banner at the topmost hierarchical layer, and includes sprites for the game title and IDs.

Extra hardware The game makes use of the in-built accelerometer, available on the SPI interface, in order to control player movement. It also displays the game title on the 7-segment display.

Sprites The game makes use of 3 sprites, one for the player, title, and IDs. There are some issues with the player sprite appearance, where the sprite is shifted somewhat. However, these have been mitigated by adjusting the sprite ROM to negate this effect.

4.2 Reflection

Colour blindness support The game in its current form is not supportive of most forms of colour vision deficiency, with the likely exception of tritanopia and tritanomaly. Support can be increased by adjusting the colour space used, i.e. using a Cividis or Viridis colour space [3], or allowing different colour selections by the player.

Support for achromatopsia may prove difficult while retaining game flow and simplicity. One possible solution is to use symbolic elements on cells, and improve the animation between cells to indicate matching. This would, however, require a significant redesign of the game logic.

4.2.1 Player comments

During player testing, a notable complaint that cropped up was the difficulty of matching a colour, with the controls feeling ‘slippery’ and ‘imprecise’. This issue can be broken down into two parts - difficulty in identifying the colour to match, and moving to that colour.

Identification With the 64 colour selection, differentiating tones in the same quadrant can prove difficult. Colour definition can be improved with a greater display resolution, perhaps by moving to a more up-to-date display standard, although this would require the requisite adapter to be available on the board.

Movement Movement is altogether both too responsive and not responsive enough. This makes fine adjustments difficult and overshoots frequent. Overshoots can be improved with the current accelerometer implementation by compressing the responsive range to cover half the rotation it does currently. In other words, rather than a complete half turn returning 16 (the greatest value), only a quarter turn is needed, with every second value skipped. The downside is that fine adjustments will be made more difficult. A complete reimplementation of the accelerometer will be needed to return data in finer intervals to combat this.

4.2.2 General comments

The development process for the game was not smooth. Debugging, especially for graphical glitches or player interactions, proved particularly difficult to manage. This is due to how they cannot be observed easily within test benches. Some issues also do not appear to be problematic in test benches, whereas fail to render in actual implementation. The final implementation also suffers some logical glitches, where a cell remains unmatched. Overall, the game functions fairly well otherwise.

References

- [1] Analog Devices. *Micropower, 3-Axis, $\pm 2\text{ g}/\pm 4\text{ g}/\pm 8\text{ g}$ Digital Output MEMS Accelerometer. ADXL362*. One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., 2019. URL: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL362.pdf>.
- [2] David Marion. *Digital-Design/FPGA Projects/Nexys A7 3-Axis Accelerometer SPI/spi_master.v*. Commit: 79cf099bee21f10e80c8b62b6bbacb0392313f0d. 23rd July 2022. URL: https://github.com/FPGADude/Digital-Design/blob/main/FPGA%20Projects/Nexys%20A7%203-Axis%20Accelerometer%20SPI/spi_master.v.
- [3] Jamie R Nuñez, Christopher R Anderton and Ryan S Renslow. ‘Optimizing colormaps with consideration for color vision deficiency to enable accurate interpretation of scientific data’ en. In: *PLoS One* 13.7 (Aug. 2018), e0199239.
- [4] Digilent. *Nexys A7™ FPGA Board Reference Manual*. 1300 Henley Court, Pullman, WA99163, 10th July 2019.

A Code

A.1 Main files

Listing 1: game_top.v

```

1 module game_top(
2     input clk, rst,
3     input [4:0] btn,
4     output [11:0] colour,
5     output hsync, vsync,
6     input ACL_MISO,
7     output ACL_MOSI, ACL_SCLK, ACL_CSN,
8     output a, b, c, d, e, f, g,
9     output [7:0] an
10 );
11
12     wire [11:0] draw;
13     wire [10:0] x_coord, y_coord;
14
15     wire game_clock;
16     wire display_clock;
17     wire spi_clock;
18     wire [14:0] acl_data;
19
20     // screen size params
21     parameter WIDTH = 11'd1426;      // total display width available
22     parameter HEIGHT = 11'd898;       // ... height
23     parameter BORDER = 11'd8;         // border on 'gameplay' edges
24     parameter BANNER_HEIGHT = 11'd130; // size of info banner at top of screen
25     // gameplay area has 1408x752 remaining available space
26
27     // player control area
28     parameter CONTROL_X = 11'd625 + BORDER;    // top corner reference coordinates
29     parameter CONTROL_Y = 11'd297 + BANNER_HEIGHT + BORDER;
30     parameter CONTROL_COLOUR_BORDER = 11'd10;    // size of colour selection border
31     parameter CONTROL_BORDER = 11'd5;           // size of border of control area
32     parameter CONTROL_WIDTH = 11'd157;          // overall width
33     parameter CONTROL_HEIGHT = 11'd157;         // ... height
34
35     // player sprite
36     parameter SPRITE_X = 11'd16;
37     parameter SPRITE_Y = 11'd16;
38
39     parameter START_X = CONTROL_X + ((CONTROL_WIDTH)>>1) -11'd8;      // player start position
40     parameter START_Y = CONTROL_Y + ((CONTROL_HEIGHT)>>1) -11'd8;
41     parameter RIGHT_BOUND_X = CONTROL_X + CONTROL_WIDTH - CONTROL_BORDER - SPRITE_X -11'd9;
42     parameter LEFT_BOUND_X = CONTROL_X + CONTROL_BORDER + 11'd10;
43     parameter BOTTOM_BOUND_Y = CONTROL_Y + CONTROL_HEIGHT - CONTROL_BORDER - SPRITE_Y -11'd9;
44     parameter TOP_BOUND_Y = CONTROL_Y + CONTROL_BORDER + 11'd10;
45
46     reg [10:0] player_pos_x = START_X, player_pos_y = START_Y;
47     reg [10:0] player_pos_update_x, player_pos_update_y;
48
49 //     player movement
50     always@(posedge game_clock) begin
51         if(btn[0] || !rst) begin
52             player_pos_x <= START_X;
53             player_pos_y <= START_Y;
54         end else begin
55             // x movement
56             if (acl_data[9:5] <= 5'd15) begin
57                 // 0...15, update left
58                 player_pos_update_x <= -(acl_data[9:5]);
59             end else begin
60                 // 16...31, update right
61                 player_pos_update_x <= 5'd31 - acl_data[9:5];
62             end
63             // y movement
64             if (acl_data[14:10] <= 5'd15) begin
65                 // 0...15, update 0...15, up
66                 player_pos_update_y <= acl_data[14:10];
67             end else begin
68                 // 16...31, update -16...0, down
69                 player_pos_update_y <= -(5'd31 - acl_data[14:10]);
70             end
71
72             // bound y movement so any overspill is reset to bound
73             if(player_pos_y + player_pos_update_y < TOP_BOUND_Y) begin

```

```
74     player_pos_y <= TOP_BOUND_Y;
75     end else if (player_pos_y + player_pos_update_y > BOTTOM_BOUND_Y) begin
76         player_pos_y <= BOTTOM_BOUND_Y;
77     end else begin
78         // update position
79         player_pos_y <= player_pos_y + player_pos_update_y;
80     end
81     // bound x movement
82     if(player_pos_x + player_pos_update_x < LEFT_BOUND_X) begin
83         player_pos_x <= LEFT_BOUND_X;
84     end else if (player_pos_x + player_pos_update_x > RIGHT_BOUND_X) begin
85         player_pos_x <= RIGHT_BOUND_X;
86     end else begin
87         player_pos_x <= player_pos_x + player_pos_update_x;
88     end
89 end
90
91
92 //      accelerometer interface
93 spi_master master(
94     .iclk(spi_clock),
95     .miso(ACL_MISO),
96     .sclk(ACL_SCLK),
97     .mosi(ACL_MOSI),
98     .cs(ACL_CS),
99     .acl_data(acl_data)
100 );
101 // generate clock signals
102 clock_gen clock_gen_inst(
103     .clk(clk), .rst(rst),
104     .game_clock(game_clock), .display_clock(display_clock), .spi_clock(spi_clock)
105 );
106 // draw graphics
107 draw_controller drawcon_inst(
108     .clk(display_clock), .rst(rst),
109     .draw(draw),
110     .x_coord(x_coord), .y_coord(y_coord),
111     .player_pos_x(player_pos_x), .player_pos_y(player_pos_y)
112 );
113 // display to screen
114 vga_controller vga_inst(
115     .clk(display_clock),
116     .rst(rst),
117     .draw(draw),
118     .colour(colour),
119     .hsync(hsync), .vsync(vsync),
120     .x_coord(x_coord), .y_coord(y_coord)
121 );
122 // 7-seg display
123 seginterface seg_inst(
124     .a(a),
125     .b(b),
126     .c(c),
127     .d(d),
128     .e(e),
129     .f(f),
130     .g(g),
131     .an(an),
132     .clk(clk),
133     .rst(rst)
134 );
135 endmodule
```

Listing 2: clock_gen.v

```
1 module clock_gen(
2     input clk, rst,
3     output game_clock, display_clock, spi_clock
4 );
5
6 // clock wires
7 wire clk_60;
8 wire clk_8;
9
10 // game clock
11 reg gameclk;
12 reg [19:0] gclk_div;
13
14 // spi clock
15 reg spiclk;
16 reg spiclk_div;
17
18 clk_wiz_0 clk_wiz_instance
19 (
20 // Clock out ports
21 .clk_106(display_clock),
22 .clk_60(clk_60), //60MHz
23 .clk_8(clk_8), // 8MHz
24 // Clock in ports
25 .clk_in1(clk)
26 );
27
28 // produce spi clock of 4MHz
29 always@ (posedge clk_8) begin
30     if (!rst) begin
31         spiclk_div <= 0;
32         spiclk <= 0;
33     end else begin
34         if (spiclk_div) begin
35             spiclk_div <= 0;
36             spiclk <= ~spiclk;
37         end else begin
38             spiclk_div <= 1;
39         end
40     end
41 end
42
43 // produce game clock of 60Hz
44 always@ (posedge clk_60) begin
45     if (!rst) begin
46         gclk_div <= 0;
47         gameclk <= 0;
48     end else begin
49         if (gclk_div >= 20'd1000000) begin
50             gclk_div <= 0;
51             gameclk <= ~gameclk;
52         end else begin
53             gclk_div <= gclk_div + 1;
54         end
55     end
56 end
57
58 assign game_clock = gameclk;
59 assign spi_clock = spiclk;
60 endmodule
```

A.2 Graphics

Listing 3: drawcon.v

```

1 module draw_controller(
2     input clk, rst,
3     output [11:0] draw,
4     input [10:0] x_coord, y_coord, player_pos_x, player_pos_y
5 );
6
7     // colour space generation
8     reg [11:0] colourspace[0:63];
9     integer i;
10    reg[11:0] tcol = 12'hfff;
11
12    // pointer wires
13    wire [6:0] gradient_pointer;
14    wire [6:0] grid_pointer;
15    reg [6:0] selected_pointer;
16
17    // screen size params
18    parameter WIDTH = 11'd1426;      // total display width available
19    parameter HEIGHT = 11'd898;       // ... height
20    parameter BORDER = 11'd8;        // border on 'gameplay' edges
21    parameter BANNER_BG_HEIGHT = 11'd81; // size of top part of info banner at top of screen
22    parameter BANNER_HEIGHT = 11'd130; // size of info banner at top of screen
23    // gameplay area has 1408x752 remaining available space
24
25    // player control area
26    parameter CONTROL_X = 11'd625 + BORDER;    // top corner reference coordinates
27    parameter CONTROL_Y = 11'd297 + BANNER_HEIGHT + BORDER;
28    parameter CONTROL_COLOUR_BORDER = 11'd10;    // size of colour selection border
29    parameter CONTROL_BORDER = 11'd5;           // size of border of control area
30    parameter CONTROL_WIDTH = 11'd157;          // overall width
31    parameter CONTROL_HEIGHT = 11'd157;         // ... height
32
33    // colour selector parameters
34    parameter GRADIENT_START_X = CONTROL_X + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
35    parameter GRADIENT_START_Y = CONTROL_Y + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
36    parameter GRADIENT_LENGTH = 128;
37
38    // colour params
39    parameter COL_BORDER = 12'h33A;
40    parameter COL_BANNER_BG = 12'hAAF;
41    parameter COL_BANNER_FLOOR = 12'h336;
42
43    // Title sprite
44    parameter TITLE_SPRITE_X = 200, TITLE_SPRITE_Y = 130;
45    parameter TITLE_POS_X = 720;
46    reg [14:0] title_addr;
47    wire [11:0] col_title;
48
49    // player sprite
50    parameter RETICLE_SPRITE_LENGTH = 16;
51    reg [7:0] reticle_addr;
52    wire [11:0] col_reticle;
53
54    // drawing pixel
55    reg [11:0] pixel;
56    reg [11:0] col_selected; // selected colour
57
58    // ID sprite
59    parameter ID_SPRITE_X = 61;
60    parameter ID_SPRITE_Y = 31;
61    parameter ID_SPRITE_OFFSET = 5;
62    reg [10:0] id_addr;
63    wire [11:0] col_id;
64
65 always@(posedge clk) begin
66     // create colourspace
67     for(i=0;i<64;i=i+1)begin
68         colourspace[i] = tcol;
69         tcol = tcol - 12'h020;
70         if(tcol[7:4] == 4'hf) begin
71             tcol = tcol - 12'h100;
72         end
73     end
74
75     // banner
76     if (y_coord < BANNER_HEIGHT) begin

```

```

77     if ( (x_coord < TITLE_POS_X - TITLE_SPRITE_X/2 ) || (x_coord >= TITLE_POS_X + TITLE_SPRITE_X/2 )
78     || (y_coord >= TITLE_SPRITE_Y ) ) begin
79         if ((x_coord >= ID_SPRITE_OFFSET) && (x_coord < ID_SPRITE_OFFSET + ID_SPRITE_X) && (y_coord >=
80             ID_SPRITE_OFFSET) && (y_coord < ID_SPRITE_OFFSET + ID_SPRITE_Y)) begin
81             pixel <= col_id;
82             if ((id_addr == (ID_SPRITE_X * ID_SPRITE_Y) -1)|| ( (x_coord == ID_SPRITE_OFFSET) && (
83                 y_coord == ID_SPRITE_OFFSET) )) begin
84                 id_addr <= 0;
85             end else begin
86                 id_addr <= id_addr + 1;
87             end
88             // banner colours
89         end else if (y_coord < BANNER_BG_HEIGHT) begin
90             pixel <= COL_BANNER_BG;
91         end else begin
92             pixel <= COL_BANNER_FLOOR;
93         end
94         // title sprite
95     end else begin
96         pixel <= col_title;
97         if ((title_addr == (TITLE_SPRITE_X * TITLE_SPRITE_Y) -1)|| ( (x_coord == TITLE_POS_X -
98             TITLE_SPRITE_X/2) && (y_coord == 0) )) begin
99             title_addr <= 0;
100            end else begin
101                title_addr <= title_addr + 1;
102            end
103        end
104    end else begin
105        // below banner
106        //frame border
107        if ((x_coord < BORDER) || (x_coord > (WIDTH - BORDER)) ||
108            (y_coord < BANNER_HEIGHT + BORDER) || (y_coord > (HEIGHT - BORDER))) begin
109            pixel <= COL_BORDER;
110        // player reticle sprite
111        end else if ( (x_coord >= player_pos_x ) && (x_coord < player_pos_x + RETICLE_SPRITE_LENGTH ) &&
112            (y_coord >= player_pos_y ) && (y_coord < player_pos_y + RETICLE_SPRITE_LENGTH ) )
113        begin
114            // reset addr
115            if ( (reticle_addr == (RETICLE_SPRITE_LENGTH**2)-1) ||           // at end of sprite draw
116                ( (x_coord == player_pos_x ) && (y_coord == player_pos_y) ) ) begin // at beginning of
117                sprite position
118                reticle_addr <= 0;
119            end else begin
120                // sprite transparency
121                if (col_reticle == 12'hfff) begin
122                    pixel <= colourspace[gradient_pointer];
123                // colour selection
124                end else if (col_reticle == 12'hf00) begin
125                    pixel <= colourspace[gradient_pointer];
126                    col_selected <= colourspace[gradient_pointer];
127                    selected_pointer <= gradient_pointer;
128                end else begin
129                    pixel <= col_reticle;
130                end
131                reticle_addr <= reticle_addr + 1;
132            end
133        // colour selector gradient area
134        end else if (
135            (x_coord >= GRADIENT_START_X) && (x_coord < (GRADIENT_START_X+GRADIENT_LENGTH)) &&
136            (y_coord >= GRADIENT_START_Y) && (y_coord < (GRADIENT_START_Y+GRADIENT_LENGTH))
137        )
138        begin
139            pixel <= colourspace[gradient_pointer];
140
141            // colour selection area border
142        end else if (
143            (x_coord > (CONTROL_X+CONTROL_COLOUR_BORDER)) && (x_coord < (CONTROL_X + CONTROL_WIDTH -
144                CONTROL_COLOUR_BORDER) ) &&
145                (y_coord > CONTROL_Y+CONTROL_COLOUR_BORDER) && (y_coord < (CONTROL_Y + CONTROL_HEIGHT -
146                    CONTROL_COLOUR_BORDER) )
147            ) begin
148                pixel <= COL_BORDER;
149
150            // selected colour picker border
151        end else if (
152            (x_coord > CONTROL_X) && (x_coord < (CONTROL_X + CONTROL_WIDTH) ) &&
153            (y_coord > CONTROL_Y) && (y_coord < (CONTROL_Y + CONTROL_HEIGHT) )
154        ) begin

```

```
148     pixel <= col_selected;
149
150     // grid background
151     end else begin
152         pixel <= colourspace[grid_pointer];
153     end
154 end
155
156 assign draw = pixel;
157
158 blk_mem_gen_2 sprite_reticle(
159     .clka(clk),
160     .addr(reticle_addr),
161     .douta(col_reticle)
162 );
163
164 blk_mem_gen_3 sprite_title(
165     .clka(clk),
166     .addr(title_addr),
167     .douta(col_title)
168 );
169
170 blk_mem_gen_0 sprite_id(
171     .clka(clk),
172     .addr(id_addr),
173     .douta(col_id)
174 );
175
176 colour_picker colour_picker_inst(
177     .clk(clk), .rst(rst),
178     .x_coord(x_coord), .y_coord(y_coord),
179     .pointer(gradient_pointer)
180 );
181
182 colour_grid colour_grid_inst(
183     .clk(clk), .rst(rst),
184     .x_coord(x_coord), .y_coord(y_coord),
185     .selected_pointer(selected_pointer),
186     .pointer(grid_pointer)
187 );
188
189
190 endmodule
```

Listing 4: colour_picker.v

```

1 module colour_picker(
2     input clk, rst,
3     input [10:0] x_coord, y_coord,
4     output [6:0] pointer
5 );
6
7 // screen size params
8 parameter BORDER = 8;           // border on 'gameplay' edges
9 parameter BANNER_HEIGHT = 130;   // size of info banner at top of screen
10
11 // player control area
12 parameter CONTROL_X = 625 + BORDER;    // top corner reference coordinates
13 parameter CONTROL_Y = 297 + BANNER_HEIGHT + BORDER;
14 parameter CONTROL_COLOUR_BORDER = 10;    // size of colour selection border
15 parameter CONTROL_BORDER = 5;           // size of border of control area
16
17 // colour selector parameters
18 parameter GRADIENT_START_X = CONTROL_X + CONTROL_COLOUR_BORDER + CONTROL_BORDER -1;
19 parameter GRADIENT_START_Y = CONTROL_Y + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
20 parameter GRADIENT_LENGTH = 128;
21
22 reg [8:0] col_pointer = 0;
23 reg [8:0] start_pointer = 0;
24 reg yflag = 1;
25
26 integer i,j;
27 reg [6:0] p = 0;
28
29 always@(posedge clk) begin
30     // 128x128
31     if ((x_coord >= GRADIENT_START_X) && (x_coord < GRADIENT_START_X + GRADIENT_LENGTH ) &&
32     (y_coord >= GRADIENT_START_Y) && (y_coord < GRADIENT_START_Y + GRADIENT_LENGTH))
33     begin
34         // create colour picker
35         for(j=0;j<8;j=j+1)begin
36             for(i=0;i<8;i=i+1)begin
37                 if ((x_coord >= GRADIENT_START_X + (i*16)) && (x_coord < GRADIENT_START_X + (i*16) +
16) && (y_coord >= GRADIENT_START_Y + (j*16)) && (y_coord < GRADIENT_START_Y + j*16+16))begin
38                     col_pointer <= p; end
39                     p = p+1;
40                     if(p == 64)begin p = 0; end
41                 end
42             end
43         end
44
45         assign pointer = col_pointer;
46
47 endmodule

```

Listing 5: colour_grid_i.sv

```

1 module colour_grid(
2     input clk, rst,
3     input [10:0] x_coord, y_coord,
4     input [6:0] selected_pointer,
5     output [6:0] pointer
6 );
7
8 // screen size params
9 parameter BORDER = 11'd8;           // border on 'gameplay' edges
10 parameter BANNER_HEIGHT = 11'd130; // size of info banner at top of screen
11
12 // 16 cells sizes
13 parameter CELL_WIDTH = 88;
14 parameter CELL_HEIGHT = 47;
15
16 // adjacency
17 reg [1:0] connected_array [0:15][0:15]; // 16x16 cell reference array, x/y
18 reg [6:0] pointer_array [0:15][0:15]; // current pointer array
19 reg [5:0] xcount, ycount;
20
21 reg [6:0] col_pointer;
22
23 integer i,j,k,h;
24
25 // initialise arrays
26 initial begin
27     // adjacency array
28     begin
29         connected_array[0][0] <= 0;
30         connected_array[1][0] <= 0;
31         connected_array[2][0] <= 0;
32         connected_array[3][0] <= 0;
33         connected_array[4][0] <= 0;
34         connected_array[5][0] <= 0;
35         connected_array[6][0] <= 0;
36         connected_array[7][0] <= 0;
37         connected_array[8][0] <= 0;
38         connected_array[9][0] <= 0;
39         connected_array[10][0] <= 0;
40         connected_array[11][0] <= 0;
41         connected_array[12][0] <= 0;
42         connected_array[13][0] <= 0;
43         connected_array[14][0] <= 0;
44         connected_array[15][0] <= 0;
45         connected_array[0][1] <= 0;
46         connected_array[1][1] <= 0;
47         connected_array[2][1] <= 0;
48         connected_array[3][1] <= 0;
49         connected_array[4][1] <= 0;
50         connected_array[5][1] <= 0;
51         connected_array[6][1] <= 0;
52         connected_array[7][1] <= 0;
53         connected_array[8][1] <= 0;
54         connected_array[9][1] <= 0;
55         connected_array[10][1] <= 0;
56         connected_array[11][1] <= 0;
57         connected_array[12][1] <= 0;
58         connected_array[13][1] <= 0;
59         connected_array[14][1] <= 0;
60         connected_array[15][1] <= 0;
61         connected_array[0][2] <= 0;
62         connected_array[1][2] <= 0;
63         connected_array[2][2] <= 0;
64         connected_array[3][2] <= 0;
65         connected_array[4][2] <= 0;
66         connected_array[5][2] <= 0;
67         connected_array[6][2] <= 0;
68         connected_array[7][2] <= 0;
69         connected_array[8][2] <= 0;
70         connected_array[9][2] <= 0;
71         connected_array[10][2] <= 0;
72         connected_array[11][2] <= 0;
73         connected_array[12][2] <= 0;
74         connected_array[13][2] <= 0;
75         connected_array[14][2] <= 0;
76         connected_array[15][2] <= 0;
77         connected_array[0][3] <= 0;
78         connected_array[1][3] <= 0;

```

```
79     connected_array[2][3] <= 0;
80     connected_array[3][3] <= 0;
81     connected_array[4][3] <= 0;
82     connected_array[5][3] <= 0;
83     connected_array[6][3] <= 0;
84     connected_array[7][3] <= 0;
85     connected_array[8][3] <= 0;
86     connected_array[9][3] <= 0;
87     connected_array[10][3] <= 0;
88     connected_array[11][3] <= 0;
89     connected_array[12][3] <= 0;
90     connected_array[13][3] <= 0;
91     connected_array[14][3] <= 0;
92     connected_array[15][3] <= 0;
93     connected_array[0][4] <= 0;
94     connected_array[1][4] <= 0;
95     connected_array[2][4] <= 0;
96     connected_array[3][4] <= 0;
97     connected_array[4][4] <= 0;
98     connected_array[5][4] <= 0;
99     connected_array[6][4] <= 0;
100    connected_array[7][4] <= 0;
101    connected_array[8][4] <= 0;
102    connected_array[9][4] <= 0;
103    connected_array[10][4] <= 0;
104    connected_array[11][4] <= 0;
105    connected_array[12][4] <= 0;
106    connected_array[13][4] <= 0;
107    connected_array[14][4] <= 0;
108    connected_array[15][4] <= 0;
109    connected_array[0][5] <= 0;
110    connected_array[1][5] <= 0;
111    connected_array[2][5] <= 0;
112    connected_array[3][5] <= 0;
113    connected_array[4][5] <= 0;
114    connected_array[5][5] <= 0;
115    connected_array[6][5] <= 0;
116    connected_array[7][5] <= 0;
117    connected_array[8][5] <= 0;
118    connected_array[9][5] <= 0;
119    connected_array[10][5] <= 0;
120    connected_array[11][5] <= 0;
121    connected_array[12][5] <= 0;
122    connected_array[13][5] <= 0;
123    connected_array[14][5] <= 0;
124    connected_array[15][5] <= 0;
125    connected_array[0][6] <= 0;
126    connected_array[1][6] <= 0;
127    connected_array[2][6] <= 0;
128    connected_array[3][6] <= 0;
129    connected_array[4][6] <= 0;
130    connected_array[5][6] <= 0;
131    connected_array[6][6] <= 0;
132    connected_array[7][6] <= 1;
133    connected_array[8][6] <= 1;
134    connected_array[9][6] <= 0;
135    connected_array[10][6] <= 0;
136    connected_array[11][6] <= 0;
137    connected_array[12][6] <= 0;
138    connected_array[13][6] <= 0;
139    connected_array[14][6] <= 0;
140    connected_array[15][6] <= 0;
141    connected_array[0][7] <= 0;
142    connected_array[1][7] <= 0;
143    connected_array[2][7] <= 0;
144    connected_array[3][7] <= 0;
145    connected_array[4][7] <= 0;
146    connected_array[5][7] <= 0;
147    connected_array[6][7] <= 0;
148    connected_array[7][7] <= 1;
149    connected_array[8][7] <= 1;
150    connected_array[9][7] <= 0;
151    connected_array[10][7] <= 0;
152    connected_array[11][7] <= 0;
153    connected_array[12][7] <= 0;
154    connected_array[13][7] <= 0;
155    connected_array[14][7] <= 0;
156    connected_array[15][7] <= 0;
157    connected_array[0][8] <= 0;
```

```
158     connected_array[1][8] <= 0;
159     connected_array[2][8] <= 0;
160     connected_array[3][8] <= 0;
161     connected_array[4][8] <= 0;
162     connected_array[5][8] <= 0;
163     connected_array[6][8] <= 0;
164     connected_array[7][8] <= 1;
165     connected_array[8][8] <= 1;
166     connected_array[9][8] <= 0;
167     connected_array[10][8] <= 0;
168     connected_array[11][8] <= 0;
169     connected_array[12][8] <= 0;
170     connected_array[13][8] <= 0;
171     connected_array[14][8] <= 0;
172     connected_array[15][8] <= 0;
173     connected_array[0][9] <= 0;
174     connected_array[1][9] <= 0;
175     connected_array[2][9] <= 0;
176     connected_array[3][9] <= 0;
177     connected_array[4][9] <= 0;
178     connected_array[5][9] <= 0;
179     connected_array[6][9] <= 0;
180     connected_array[7][9] <= 1;
181     connected_array[8][9] <= 1;
182     connected_array[9][9] <= 0;
183     connected_array[10][9] <= 0;
184     connected_array[11][9] <= 0;
185     connected_array[12][9] <= 0;
186     connected_array[13][9] <= 0;
187     connected_array[14][9] <= 0;
188     connected_array[15][9] <= 0;
189     connected_array[0][10] <= 0;
190     connected_array[1][10] <= 0;
191     connected_array[2][10] <= 0;
192     connected_array[3][10] <= 0;
193     connected_array[4][10] <= 0;
194     connected_array[5][10] <= 0;
195     connected_array[6][10] <= 0;
196     connected_array[7][10] <= 0;
197     connected_array[8][10] <= 0;
198     connected_array[9][10] <= 0;
199     connected_array[10][10] <= 0;
200     connected_array[11][10] <= 0;
201     connected_array[12][10] <= 0;
202     connected_array[13][10] <= 0;
203     connected_array[14][10] <= 0;
204     connected_array[15][10] <= 0;
205     connected_array[0][11] <= 0;
206     connected_array[1][11] <= 0;
207     connected_array[2][11] <= 0;
208     connected_array[3][11] <= 0;
209     connected_array[4][11] <= 0;
210     connected_array[5][11] <= 0;
211     connected_array[6][11] <= 0;
212     connected_array[7][11] <= 0;
213     connected_array[8][11] <= 0;
214     connected_array[9][11] <= 0;
215     connected_array[10][11] <= 0;
216     connected_array[11][11] <= 0;
217     connected_array[12][11] <= 0;
218     connected_array[13][11] <= 0;
219     connected_array[14][11] <= 0;
220     connected_array[15][11] <= 0;
221     connected_array[0][12] <= 0;
222     connected_array[1][12] <= 0;
223     connected_array[2][12] <= 0;
224     connected_array[3][12] <= 0;
225     connected_array[4][12] <= 0;
226     connected_array[5][12] <= 0;
227     connected_array[6][12] <= 0;
228     connected_array[7][12] <= 0;
229     connected_array[8][12] <= 0;
230     connected_array[9][12] <= 0;
231     connected_array[10][12] <= 0;
232     connected_array[11][12] <= 0;
233     connected_array[12][12] <= 0;
234     connected_array[13][12] <= 0;
235     connected_array[14][12] <= 0;
236     connected_array[15][12] <= 0;
```

```
237     connected_array[0][13] <= 0;
238     connected_array[1][13] <= 0;
239     connected_array[2][13] <= 0;
240     connected_array[3][13] <= 0;
241     connected_array[4][13] <= 0;
242     connected_array[5][13] <= 0;
243     connected_array[6][13] <= 0;
244     connected_array[7][13] <= 0;
245     connected_array[8][13] <= 0;
246     connected_array[9][13] <= 0;
247     connected_array[10][13] <= 0;
248     connected_array[11][13] <= 0;
249     connected_array[12][13] <= 0;
250     connected_array[13][13] <= 0;
251     connected_array[14][13] <= 0;
252     connected_array[15][13] <= 0;
253     connected_array[0][14] <= 0;
254     connected_array[1][14] <= 0;
255     connected_array[2][14] <= 0;
256     connected_array[3][14] <= 0;
257     connected_array[4][14] <= 0;
258     connected_array[5][14] <= 0;
259     connected_array[6][14] <= 0;
260     connected_array[7][14] <= 0;
261     connected_array[8][14] <= 0;
262     connected_array[9][14] <= 0;
263     connected_array[10][14] <= 0;
264     connected_array[11][14] <= 0;
265     connected_array[12][14] <= 0;
266     connected_array[13][14] <= 0;
267     connected_array[14][14] <= 0;
268     connected_array[15][14] <= 0;
269     connected_array[0][15] <= 0;
270     connected_array[1][15] <= 0;
271     connected_array[2][15] <= 0;
272     connected_array[3][15] <= 0;
273     connected_array[4][15] <= 0;
274     connected_array[5][15] <= 0;
275     connected_array[6][15] <= 0;
276     connected_array[7][15] <= 0;
277     connected_array[8][15] <= 0;
278     connected_array[9][15] <= 0;
279     connected_array[10][15] <= 0;
280     connected_array[11][15] <= 0;
281     connected_array[12][15] <= 0;
282     connected_array[13][15] <= 0;
283     connected_array[14][15] <= 0;
284     connected_array[15][15] <= 0;
285 end
286 // 'random' grid colours
287 begin
288     pointer_array[0][0] <= 34;
289     pointer_array[1][0] <= 40;
290     pointer_array[2][0] <= 32;
291     pointer_array[3][0] <= 55;
292     pointer_array[4][0] <= 50;
293     pointer_array[5][0] <= 41;
294     pointer_array[6][0] <= 39;
295     pointer_array[7][0] <= 45;
296     pointer_array[8][0] <= 63;
297     pointer_array[9][0] <= 31;
298     pointer_array[10][0] <= 19;
299     pointer_array[11][0] <= 52;
300     pointer_array[12][0] <= 17;
301     pointer_array[13][0] <= 0;
302     pointer_array[14][0] <= 28;
303     pointer_array[15][0] <= 46;
304     pointer_array[0][1] <= 31;
305     pointer_array[1][1] <= 53;
306     pointer_array[2][1] <= 30;
307     pointer_array[3][1] <= 60;
308     pointer_array[4][1] <= 56;
309     pointer_array[5][1] <= 18;
310     pointer_array[6][1] <= 26;
311     pointer_array[7][1] <= 25;
312     pointer_array[8][1] <= 1;
313     pointer_array[9][1] <= 5;
314     pointer_array[10][1] <= 23;
315     pointer_array[11][1] <= 30;
```

```
316     pointer_array[12][1] <= 43;
317     pointer_array[13][1] <= 8;
318     pointer_array[14][1] <= 13;
319     pointer_array[15][1] <= 28;
320     pointer_array[0][2] <= 51;
321     pointer_array[1][2] <= 17;
322     pointer_array[2][2] <= 43;
323     pointer_array[3][2] <= 30;
324     pointer_array[4][2] <= 4;
325     pointer_array[5][2] <= 12;
326     pointer_array[6][2] <= 27;
327     pointer_array[7][2] <= 25;
328     pointer_array[8][2] <= 19;
329     pointer_array[9][2] <= 8;
330     pointer_array[10][2] <= 53;
331     pointer_array[11][2] <= 57;
332     pointer_array[12][2] <= 30;
333     pointer_array[13][2] <= 38;
334     pointer_array[14][2] <= 38;
335     pointer_array[15][2] <= 60;
336     pointer_array[0][3] <= 54;
337     pointer_array[1][3] <= 22;
338     pointer_array[2][3] <= 37;
339     pointer_array[3][3] <= 22;
340     pointer_array[4][3] <= 19;
341     pointer_array[5][3] <= 10;
342     pointer_array[6][3] <= 21;
343     pointer_array[7][3] <= 31;
344     pointer_array[8][3] <= 40;
345     pointer_array[9][3] <= 55;
346     pointer_array[10][3] <= 20;
347     pointer_array[11][3] <= 57;
348     pointer_array[12][3] <= 23;
349     pointer_array[13][3] <= 25;
350     pointer_array[14][3] <= 9;
351     pointer_array[15][3] <= 18;
352     pointer_array[0][4] <= 42;
353     pointer_array[1][4] <= 17;
354     pointer_array[2][4] <= 26;
355     pointer_array[3][4] <= 31;
356     pointer_array[4][4] <= 12;
357     pointer_array[5][4] <= 17;
358     pointer_array[6][4] <= 38;
359     pointer_array[7][4] <= 34;
360     pointer_array[8][4] <= 11;
361     pointer_array[9][4] <= 4;
362     pointer_array[10][4] <= 51;
363     pointer_array[11][4] <= 18;
364     pointer_array[12][4] <= 32;
365     pointer_array[13][4] <= 38;
366     pointer_array[14][4] <= 40;
367     pointer_array[15][4] <= 4;
368     pointer_array[0][5] <= 45;
369     pointer_array[1][5] <= 18;
370     pointer_array[2][5] <= 12;
371     pointer_array[3][5] <= 41;
372     pointer_array[4][5] <= 36;
373     pointer_array[5][5] <= 44;
374     pointer_array[6][5] <= 0;
375     pointer_array[7][5] <= 53;
376     pointer_array[8][5] <= 9;
377     pointer_array[9][5] <= 16;
378     pointer_array[10][5] <= 2;
379     pointer_array[11][5] <= 7;
380     pointer_array[12][5] <= 0;
381     pointer_array[13][5] <= 47;
382     pointer_array[14][5] <= 41;
383     pointer_array[15][5] <= 33;
384     pointer_array[0][6] <= 21;
385     pointer_array[1][6] <= 32;
386     pointer_array[2][6] <= 4;
387     pointer_array[3][6] <= 38;
388     pointer_array[4][6] <= 56;
389     pointer_array[5][6] <= 61;
390     pointer_array[6][6] <= 1;
391     pointer_array[7][6] <= 43;
392     pointer_array[8][6] <= 31;
393     pointer_array[9][6] <= 5;
394     pointer_array[10][6] <= 14;
```

```
395     pointer_array[11][6] <= 40;
396     pointer_array[12][6] <= 1;
397     pointer_array[13][6] <= 16;
398     pointer_array[14][6] <= 54;
399     pointer_array[15][6] <= 3;
400     pointer_array[0][7] <= 55;
401     pointer_array[1][7] <= 38;
402     pointer_array[2][7] <= 28;
403     pointer_array[3][7] <= 46;
404     pointer_array[4][7] <= 5;
405     pointer_array[5][7] <= 47;
406     pointer_array[6][7] <= 51;
407     pointer_array[7][7] <= 60;
408     pointer_array[8][7] <= 52;
409     pointer_array[9][7] <= 52;
410     pointer_array[10][7] <= 33;
411     pointer_array[11][7] <= 18;
412     pointer_array[12][7] <= 42;
413     pointer_array[13][7] <= 45;
414     pointer_array[14][7] <= 41;
415     pointer_array[15][7] <= 26;
416     pointer_array[0][8] <= 26;
417     pointer_array[1][8] <= 20;
418     pointer_array[2][8] <= 58;
419     pointer_array[3][8] <= 38;
420     pointer_array[4][8] <= 55;
421     pointer_array[5][8] <= 41;
422     pointer_array[6][8] <= 48;
423     pointer_array[7][8] <= 27;
424     pointer_array[8][8] <= 5;
425     pointer_array[9][8] <= 45;
426     pointer_array[10][8] <= 32;
427     pointer_array[11][8] <= 54;
428     pointer_array[12][8] <= 56;
429     pointer_array[13][8] <= 11;
430     pointer_array[14][8] <= 11;
431     pointer_array[15][8] <= 55;
432     pointer_array[0][9] <= 15;
433     pointer_array[1][9] <= 58;
434     pointer_array[2][9] <= 4;
435     pointer_array[3][9] <= 55;
436     pointer_array[4][9] <= 33;
437     pointer_array[5][9] <= 62;
438     pointer_array[6][9] <= 56;
439     pointer_array[7][9] <= 50;
440     pointer_array[8][9] <= 39;
441     pointer_array[9][9] <= 9;
442     pointer_array[10][9] <= 23;
443     pointer_array[11][9] <= 38;
444     pointer_array[12][9] <= 16;
445     pointer_array[13][9] <= 37;
446     pointer_array[14][9] <= 54;
447     pointer_array[15][9] <= 51;
448     pointer_array[0][10] <= 19;
449     pointer_array[1][10] <= 61;
450     pointer_array[2][10] <= 41;
451     pointer_array[3][10] <= 25;
452     pointer_array[4][10] <= 25;
453     pointer_array[5][10] <= 46;
454     pointer_array[6][10] <= 9;
455     pointer_array[7][10] <= 31;
456     pointer_array[8][10] <= 3;
457     pointer_array[9][10] <= 27;
458     pointer_array[10][10] <= 63;
459     pointer_array[11][10] <= 42;
460     pointer_array[12][10] <= 47;
461     pointer_array[13][10] <= 18;
462     pointer_array[14][10] <= 0;
463     pointer_array[15][10] <= 63;
464     pointer_array[0][11] <= 28;
465     pointer_array[1][11] <= 4;
466     pointer_array[2][11] <= 45;
467     pointer_array[3][11] <= 31;
468     pointer_array[4][11] <= 55;
469     pointer_array[5][11] <= 30;
470     pointer_array[6][11] <= 22;
471     pointer_array[7][11] <= 13;
472     pointer_array[8][11] <= 32;
473     pointer_array[9][11] <= 16;
```

```

474     pointer_array[10][11] <= 40;
475     pointer_array[11][11] <= 2;
476     pointer_array[12][11] <= 15;
477     pointer_array[13][11] <= 60;
478     pointer_array[14][11] <= 40;
479     pointer_array[15][11] <= 39;
480     pointer_array[0][12] <= 6;
481     pointer_array[1][12] <= 22;
482     pointer_array[2][12] <= 23;
483     pointer_array[3][12] <= 8;
484     pointer_array[4][12] <= 1;
485     pointer_array[5][12] <= 30;
486     pointer_array[6][12] <= 44;
487     pointer_array[7][12] <= 12;
488     pointer_array[8][12] <= 56;
489     pointer_array[9][12] <= 3;
490     pointer_array[10][12] <= 5;
491     pointer_array[11][12] <= 41;
492     pointer_array[12][12] <= 18;
493     pointer_array[13][12] <= 28;
494     pointer_array[14][12] <= 8;
495     pointer_array[15][12] <= 44;
496     pointer_array[0][13] <= 11;
497     pointer_array[1][13] <= 29;
498     pointer_array[2][13] <= 2;
499     pointer_array[3][13] <= 24;
500     pointer_array[4][13] <= 44;
501     pointer_array[5][13] <= 37;
502     pointer_array[6][13] <= 40;
503     pointer_array[7][13] <= 26;
504     pointer_array[8][13] <= 61;
505     pointer_array[9][13] <= 26;
506     pointer_array[10][13] <= 52;
507     pointer_array[11][13] <= 20;
508     pointer_array[12][13] <= 12;
509     pointer_array[13][13] <= 27;
510     pointer_array[14][13] <= 0;
511     pointer_array[15][13] <= 55;
512     pointer_array[0][14] <= 46;
513     pointer_array[1][14] <= 10;
514     pointer_array[2][14] <= 2;
515     pointer_array[3][14] <= 36;
516     pointer_array[4][14] <= 43;
517     pointer_array[5][14] <= 33;
518     pointer_array[6][14] <= 60;
519     pointer_array[7][14] <= 5;
520     pointer_array[8][14] <= 33;
521     pointer_array[9][14] <= 44;
522     pointer_array[10][14] <= 5;
523     pointer_array[11][14] <= 26;
524     pointer_array[12][14] <= 23;
525     pointer_array[13][14] <= 8;
526     pointer_array[14][14] <= 17;
527     pointer_array[15][14] <= 49;
528     pointer_array[0][15] <= 16;
529     pointer_array[1][15] <= 35;
530     pointer_array[2][15] <= 4;
531     pointer_array[3][15] <= 62;
532     pointer_array[4][15] <= 11;
533     pointer_array[5][15] <= 20;
534     pointer_array[6][15] <= 13;
535     pointer_array[7][15] <= 51;
536     pointer_array[8][15] <= 53;
537     pointer_array[9][15] <= 34;
538     pointer_array[10][15] <= 39;
539     pointer_array[11][15] <= 41;
540     pointer_array[12][15] <= 31;
541     pointer_array[13][15] <= 61;
542     pointer_array[14][15] <= 26;
543     pointer_array[15][15] <= 24;
544   end
545 end
546
547 always@ (posedge clk) begin
548   // check currently connected cells
549   if (connected_array[xcount][ycount] == 1) begin
550     //if matched
551     if (pointer_array[xcount][ycount] == selected_pointer) begin
552       // check adjacent connected cells

```

```
553         if (connected_array[xcount+1][ycount] == 0) begin
554             connected_array[xcount+1][ycount] <= 1;
555         end
556
557         if (connected_array[xcount-1][ycount] == 0) begin
558             connected_array[xcount-1][ycount] <= 1;
559         end
560
561         if (connected_array[xcount][ycount+1] == 0) begin
562             connected_array[xcount][ycount+1] <= 1;
563         end
564
565         if (connected_array[xcount][ycount-1] == 0) begin
566             connected_array[xcount][ycount-1] <= 1;
567         end
568
569         //set matched, change colour
570         connected_array[xcount][ycount] <= 2;
571         pointer_array[xcount][ycount] <= selected_pointer;
572     end
573 //update already matched cell
574 end else if (connected_array[xcount][ycount] == 2) begin
575     pointer_array[xcount][ycount] <= selected_pointer;
576 end
577
578 // create background cell grid
579 for(k=0;k<16;k=k+1)begin
580     for(h=0;h<16;h=h+1)begin
581         if ((x_coord >= BORDER + h*CELL_WIDTH -1) && (x_coord < BORDER + h*CELL_WIDTH + CELL_WIDTH
582 +1) &&
583             (y_coord >= BORDER + BANNER_HEIGHT + k*CELL_HEIGHT) && (y_coord < BORDER +
584 BANNER_HEIGHT + k*CELL_HEIGHT + CELL_HEIGHT+1)
585             )begin col_pointer <= pointer_array[h][k]; end
586         end
587     end
588
589     // cell position counters
590     if(ycount > 15) begin
591         ycount <= 0;
592         xcount <= 0;
593     end else if(xcount > 15) begin
594         xcount <= 0;
595         ycount <= ycount + 1;
596     end else begin
597         xcount <= xcount + 1;
598     end
599     assign pointer = col_pointer;
600 endmodule
```

A.3 Display

Listing 6: vga_out.v

```

1 module vga_controller(
2     input clk, rst,
3     output [11:0] colour,
4     input [11:0] draw,
5     output hsync, vsync,
6     output [10:0] x_coord, y_coord
7 );
8
9     //display parameters
10    // horizontal parameters
11    parameter DISPLAYH_MAX = 11'd1824;
12    parameter DISPLAYH_MIN = 11'd384;
13    parameter HSYNC_START = 11'd151;
14    parameter HCOUNT_MAX = 11'd1903;
15
16    // vertical parameters
17    parameter DISPLAYV_MAX = 10'd931;
18    parameter DISPLAYV_MIN = 10'd31;
19    parameter VSYNC_START = 10'd2;
20    parameter VCOUNT_MAX = 10'd931;
21
22    // background colour
23    parameter COL_BLACK = 12'h000;
24
25    //internal signals
26    reg [10:0] hcount;
27    reg [9:0] vcount;
28    reg [10:0] x_reg;
29    reg [10:0] y_reg;
30
31    wire display_region;
32    wire v_region;
33    wire h_end = (hcount == HCOUNT_MAX);
34    wire v_end = (vcount == VCOUNT_MAX);
35
36    wire [3:0]draw_r;
37    wire [3:0]draw_g;
38    wire [3:0]draw_b;
39
40    //hsync vsync combinational
41    assign hsync = ((hcount >= 11'd0) && (hcount <= HSYNC_START));
42    assign vsync = ((vcount >= 10'd0) && (vcount <= VSYNC_START));
43
44    assign display_region = ((hcount >= DISPLAYH_MIN) && (hcount <= DISPLAYH_MAX) && (vcount >=
45    DISPLAYV_MIN) && (vcount <= DISPLAYV_MAX));
46
47    assign colour = (display_region) ? draw : COL_BLACK;
48
49    always@(posedge clk) begin
50        if (!rst) begin
51            hcount <= 11'd0;
52            vcount <= 10'd0;
53            x_reg <= 11'd0;
54            y_reg <= 11'd0;
55        end
56        else begin
57            // hcount counter
58            if (h_end) begin
59                hcount <= 11'd0;
60            end else begin
61                hcount <= hcount + 11'd1;
62            end
63            // vcount counter
64            if (v_end) begin
65                vcount <= 10'd0;
66            end else if (h_end) begin
67                vcount <= vcount + 10'd1;
68            end
69            //x coordinate counter
70            if((hcount >= DISPLAYH_MIN) && (hcount <= DISPLAYH_MAX)) begin
71                x_reg <= x_reg + 11'd1;
72            end else begin
73                x_reg <= 11'd0;
74            end
75            //y coordinate counter
76            if (h_end) begin

```

```
76         if((vcount >= DISPLAYV_MIN) && (vcount <= DISPLAYV_MAX)) begin
77             y_reg <= y_reg + 11'd1;
78         end else begin
79             y_reg <= 11'd0;
80         end
81     end
82
83     end
84 end
85
86 assign x_coord = x_reg;
87 assign y_coord = y_reg;
88
89 endmodule
```

A.4 Seven-segment display

Listing 7: seginterface.v

```

1 module seginterface(
2     input clk, rst,
3     output div_clk,
4     output a, b, c, d, e, f, g,
5     output [7:0] an
6 );
7
8     wire led_clk;
9     reg [3:0] dig_sel;
10
11    reg [28:0] clk_count = 11'd0;
12
13    always @(posedge clk)
14        clk_count <= clk_count + 1'b1;
15
16    assign led_clk = clk_count[16];
17    assign div_clk = clk_count[25];
18
19    reg [7:0] led_strobe = 8'b11111110;
20    always @(posedge led_clk)
21        led_strobe <= {led_strobe[6:0],led_strobe[7]};
22    assign an = led_strobe;
23
24    reg [2:0] led_index = 3'd0;
25    always @(posedge led_clk)
26        led_index <= led_index + 1'b1;
27
28    always@*
29        case (led_index)
30            3'd0: dig_sel = 4'hd;
31            3'd1: dig_sel = 4'h0;
32            3'd2: dig_sel = 4'h0;
33            3'd3: dig_sel = 4'h0;
34            3'd4: dig_sel = 4'h0;
35            3'd5: dig_sel = 4'h0;
36            3'd6: dig_sel = 4'h1;
37            3'd7: dig_sel = 4'hf;
38        endcase
39
40    sevenseg M1(.num(dig_sel), .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g));
41
42 endmodule

```

Listing 8: sevenseg.v

```
1 `timescale 1ns / 1ps
2
3 module sevenseg(
4     input [3:0] num,
5     output a,
6     output b,
7     output c,
8     output d,
9     output e,
10    output f,
11    output g
12 );
13
14 reg [6:0] intseg;
15 assign {a,b,c,d,e,f,g} = ~intseg;
16
17 always@*
18 begin
19     case(num)
20         4'h0: intseg = 7'b0011101;
21         4'h1: intseg = 7'b0110000;
22         4'h2: intseg = 7'b1101101;
23         4'h3: intseg = 7'b1111001;
24         4'h4: intseg = 7'b0110011;
25         4'h5: intseg = 7'b1011011;
26         4'h6: intseg = 7'b1011111;
27         4'h7: intseg = 7'b1110000;
28         4'h8: intseg = 7'b1111111;
29         4'h9: intseg = 7'b1111011;
30         4'ha: intseg = 7'b1110111;
31         4'hb: intseg = 7'b0011111;
32         4'hc: intseg = 7'b1001110;
33         4'hd: intseg = 7'b0111101;
34         4'he: intseg = 7'b1001111;
35         4'hf: intseg = 7'b1000111;
36     endcase
37 end
38
39
40 endmodule
```