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 106.301 58 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 module[1] 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.

[1] Section 2.8.

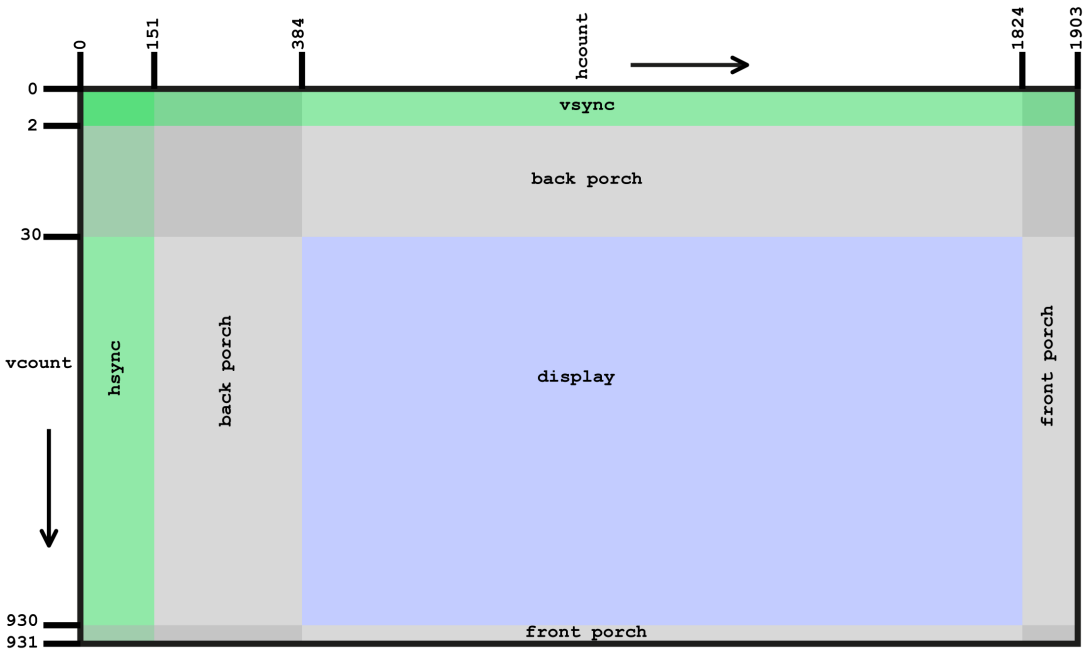


Figure 1: Display parameters

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	`VGA_R[3:0]`	`VGA_G[3:0]`	`VGA_B[3:0]`
Implementation	`colour[11:8]`	`colour[7:4]`	`colour[3:0]`

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 space[2], 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.

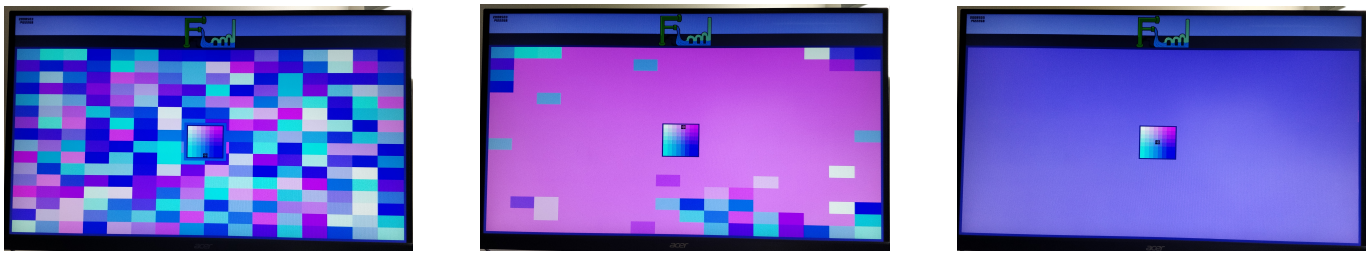
2 The game

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

[2] A 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.



(a) Start **(b)** Partway **(c)** Cleared!

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.

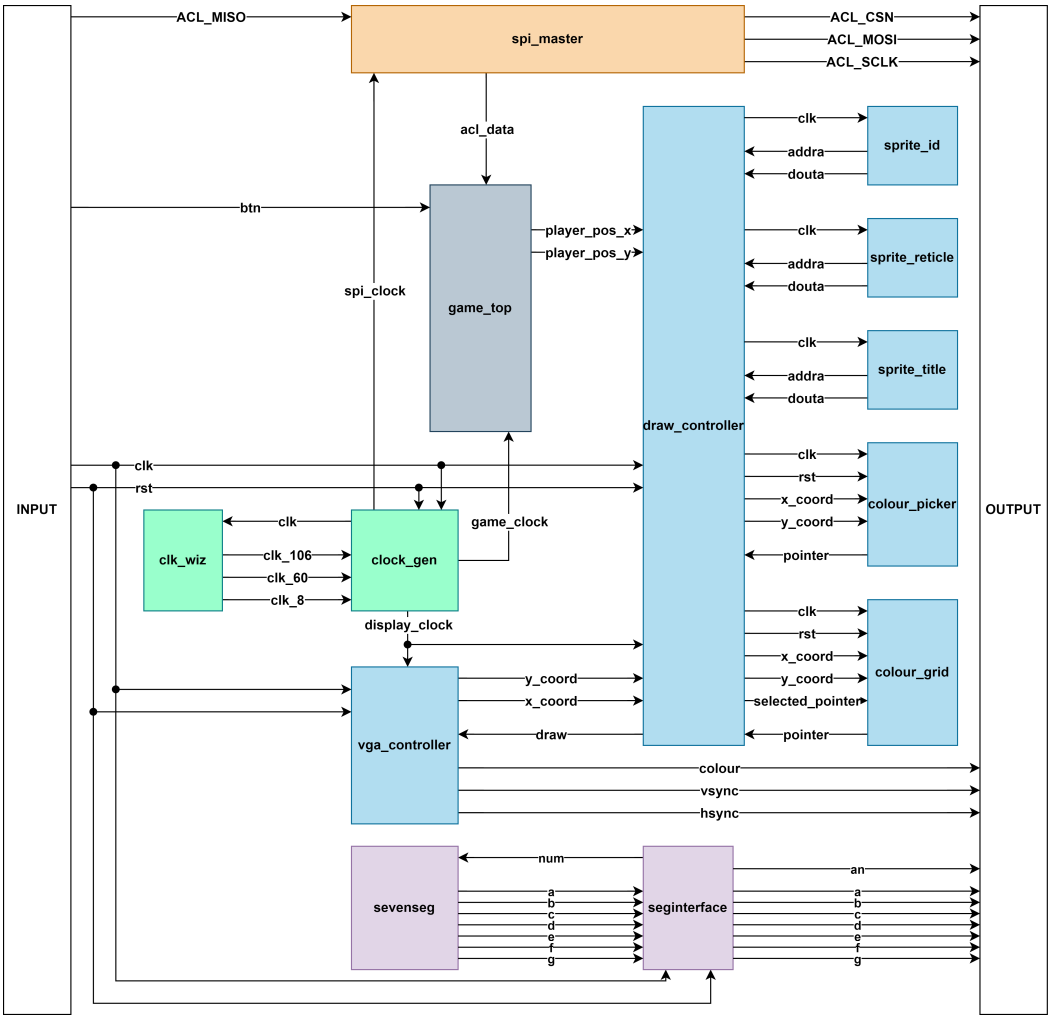


Figure 3: Relationship diagram

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 sprite[3].

The accelerometer controller[4] 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*.

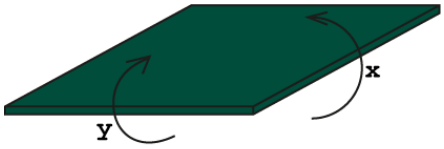


Figure 4: Accelerometer tilt influence

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 position[5].

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 parameter[6]. This maintains a smooth collision with the bounding walls.

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 1 M 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`	106.301 58 MHz	106.296 MHz	0.005 25 %
`clk_60`	60 Hz	59.792 MHz	0.347 %
`clk_8`	8 MHz	7.972 MHz	0.350 %

Table 2: Clocking wizard clocks

[3] Section 2.8.1.
[4] Section 2.4.
[5] Lines 55-70.
[6] Lines 72-88.
[7] Section 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 `0xFFF` (white) and reduces by 2 in green horizontally (to magenta, `0xF0F`) and by 2 in red vertically (to cyan, `0x0FF`). The 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.

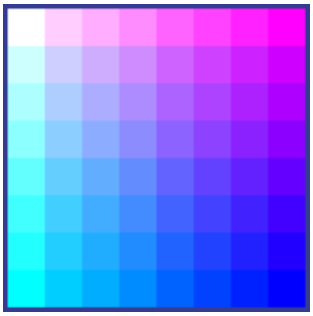


Figure 5: Gradient model

Array index	Hexadecimal value	Colour
0	0xFFF	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 `0xFFF`[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

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

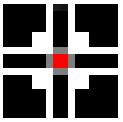
4. Screen border
5. Player reticle sprite
6. Gradient colour picker
7. Picker selection area border
8. Picker border
9. Grid background

2.8.1 sprite_reticle

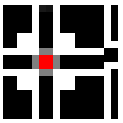
The 'player' sprite appears as a controllable reticle intended to point at the colour selected. The sprite was drawn in Microsoft Paint and converted from its base format (.bmp) to space delimited memory encoding (.coe) that can be used by Xilinx's block memory generator IP to write the sprite into ROM.

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) Intended appearance



(b) Encoded sprite

Figure 6: Player sprites

A final feature of the sprite is the red (`0xF00`) 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].

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.

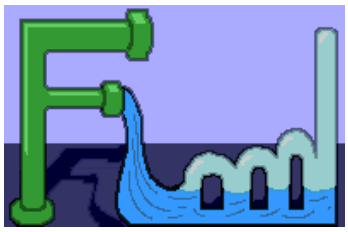


Figure 7: Title sprite

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 memory[12]. A programmatic approach, using a FOR loop is used instead[13]. In this case, the 8×8 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

[9] Lines 110-112 of `draw_controller`.
[10] Section 2.8.4.
[11] Lines 114-121 of `draw_controller`.
[12] The first version with 256 16×16 pixel cells required more memory than was available on the FPGA.
[13] Lines 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 in input and outputs a pointer for the colour of the current background pixel.

Two 16×16 2D arrays are used in this module[14]:

- `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 updated[16].

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_controller`[17] 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 conquer[18]. Therefore, the importance of testing via simulations is emphasised.

[14] The file type was changed to SystemVerilog as multidimensional and unpacked arrays are not supported by Verilog.
[15] Section 2.9.3.
[16] Lines 548-576.
[17] Section 2.8.
[18] A 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 hardcode

```
initial begin
    colourspace[0] <= 12'hfff;
    colourspace[1] <= 12'hfdf;
    colourspace[2] <= 12'hfbf;
    ...
    colourspace[7] <= 12'f1f;
    colourspace[8] <= 12'dff;
    ...
    colourspace[63] <= 12'h11f;
end
```

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

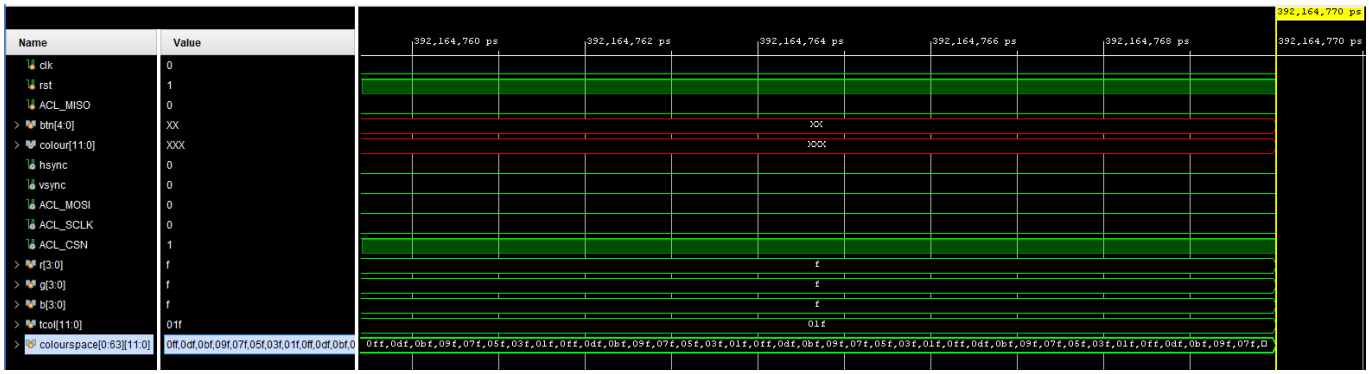
```
    r = 15
    g = 15
    b = 15
    for count in range(0,64):
        print(f"colorspace[{count}] <= 12'h{r:x}{g:x}{b:x};")
        g = g - 2
        if(g < 0):
            r = r - 2
            g = 15
```

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.

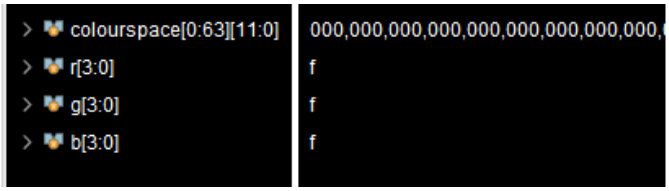
```
for(i=0;i<64;i=i+1)begin
    tcol = r<<8;
    tcol = tcol + g<<4;
    tcol = tcol + b;
    colourspace[i] = tcol;
    g = g - 2;
    if(g < 1) begin
        r = r - 2;
        g = 4'hf;
    end
end
```

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



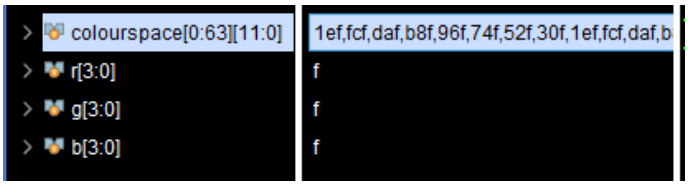
The resulting colour values are not quite correct, with the red value stuck at 0.

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



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

```
for(i=0;i<64;i=i+1)begin
    tcol = tcol + g<<4;
    tcol = tcol + b;
    colourspace[i] = tcol;
    g = g - 4'h2;
    if(g < 4'h1) begin
        r = r - 4'h2;
        g = 4'hf;
    end
end
```

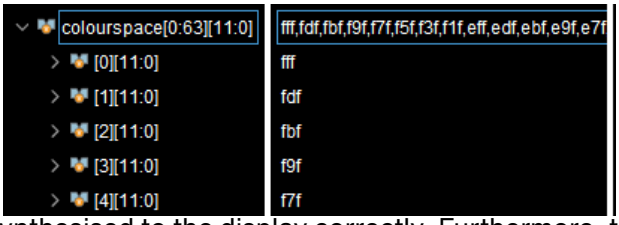


This indicates the red bit-shift is always returning 0 and non-functional. The gradient produced is also not what is intended. It may be better to move to subtracting a constant value.

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

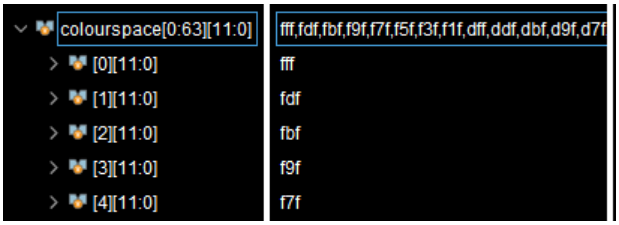
This subtractive method appears better, however the aim is to drop the red value by two, whereas here it only drops by one. The loop in an `initial` block does not appear to be synthesised correctly for actual implementation, resulting in a black screen.

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



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.

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



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 tritanomoly. 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, ±2 g/±4 g/±8 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

```
module game_top(
    input clk, rst,
    input [4:0] btn,
    output [11:0] colour,
    output hsync, vsync,
    input ACL_MISO,
    output ACL_MOSI, ACL_SCLK, ACL_CSN,
    output a, b, c, d, e, f, g,
    output [7:0] an
    );

    wire [11:0] draw;
    wire [10:0] x_coord, y_coord;

    wire game_clock;
    wire display_clock;
    wire spi_clock;
    wire [14:0] acl_data;

    // screen size params
    parameter WIDTH = 11'd1426;     // total display width available
    parameter HEIGHT = 11'd898;     // ... height
    parameter BORDER = 11'd8;     // border on 'gameplay' edges
    parameter BANNER_HEIGHT = 11'd130;  // size of info banner at top of screen
    // gameplay area has 1408x752 remaining available space

    // player control area
    parameter CONTROL_X = 11'd625 + BORDER;   // top corner reference coordinates
    parameter CONTROL_Y = 11'd297 + BANNER_HEIGHT + BORDER;
    parameter CONTROL_COLOUR_BORDER = 11'd10;   // size of colour selection border
    parameter CONTROL_BORDER = 11'd5;   // size of border of control area
    parameter CONTROL_WIDTH = 11'd157;  // overall width
    parameter CONTROL_HEIGHT = 11'd157; // ... height

    // player sprite
    parameter SPRITE_X = 11'd16;
    parameter SPRITE_Y = 11'd16;

    parameter START_X = CONTROL_X + ((CONTROL_WIDTH)>>1) -11'd8;     // player start position
    parameter START_Y = CONTROL_Y + ((CONTROL_HEIGHT)>>1) -11'd8;
    parameter RIGHT_BOUND_X = CONTROL_X + CONTROL_WIDTH - CONTROL_BORDER - SPRITE_X -11'd9;
    parameter LEFT_BOUND_X = CONTROL_X + CONTROL_BORDER + 11'd10;
    parameter BOTTOM_BOUND_Y = CONTROL_Y + CONTROL_HEIGHT - CONTROL_BORDER - SPRITE_Y -11'd9;
    parameter TOP_BOUND_Y = CONTROL_Y + CONTROL_BORDER + 11'd10;

    reg [10:0] player_pos_x = START_X, player_pos_y = START_Y;
    reg [10:0] player_pos_update_x, player_pos_update_y;

//   player movement
     always@(posedge game_clock) begin
        if(btn[0] || !rst) begin
            player_pos_x <= START_X;
            player_pos_y <= START_Y;
        end else begin
            // x movement
            if (acl_data[9:5] <= 5'd15) begin
            // 0...15, update left
                player_pos_update_x <= -(acl_data[9:5]);
            end else begin
            // 16...31, update right
                player_pos_update_x <= 5'd31 - acl_data[9:5];
            end
            // y movement
            if (acl_data[14:10] <= 5'd15) begin
            // 0...15, update 0...15, up
                player_pos_update_y <= acl_data[14:10];
            end else begin
            // 16...31, update -16...0, down
                player_pos_update_y <= -(5'd31 - acl_data[14:10]);
            end

            // bound y movement so any overspill is reset to bound
            if(player_pos_y + player_pos_update_y < TOP_BOUND_Y) begin
```

```
                player_pos_y <= TOP_BOUND_Y;
            end else if (player_pos_y + player_pos_update_y > BOTTOM_BOUND_Y) begin
                player_pos_y <= BOTTOM_BOUND_Y;
            end else begin
            // update position
                player_pos_y <= player_pos_y + player_pos_update_y;
            end
            // bound x movement
            if(player_pos_x + player_pos_update_x < LEFT_BOUND_X) begin
                player_pos_x <= LEFT_BOUND_X;
            end else if (player_pos_x + player_pos_update_x > RIGHT_BOUND_X) begin
                player_pos_x <= RIGHT_BOUND_X;
            end else begin
                player_pos_x <= player_pos_x + player_pos_update_x;
            end
        end
    end

//      accelerometer interface
     spi_master master(
        .iclk(spi_clock),
        .miso(ACL_MISO),
        .sclk(ACL_SCLK),
        .mosi(ACL_MOSI),
        .cs(ACL_CSN),
        .acl_data(acl_data)
    );
    // generate clock signals
    clock_gen clock_gen_inst(
        .clk(clk), .rst(rst),
        .game_clock(game_clock), .display_clock(display_clock), .spi_clock(spi_clock)
    );
    // draw graphics
    draw_controller drawcon_inst(
        .clk(display_clock), .rst(rst),
        .draw(draw),
        .x_coord(x_coord), .y_coord(y_coord),
        .player_pos_x(player_pos_x), .player_pos_y(player_pos_y)
    );
    // display to screen
    vga_controller vga_inst(
        .clk(display_clock),
        .rst(rst),
        .draw(draw),
        .colour(colour),
        .hsync(hsync), .vsync(vsync),
        .x_coord(x_coord), .y_coord(y_coord)
    );
    // 7-seg display
    seginterface seg_inst(
        .a(a),
        .b(b),
        .c(c),
        .d(d),
        .e(e),
        .f(f),
        .g(g),
        .an(an),
        .clk(clk),
        .rst(rst)
    );
endmodule
```

Listing 2: clock_gen.v

```
module clock_gen(
    input clk, rst,
    output game_clock, display_clock, spi_clock
    );

    // clock wires
    wire clk_60;
    wire clk_8;

    // game clock
    reg gameclk;
    reg [19:0] gclk_div;

    // spi clock
    reg spiclk;
    reg spiclk_div;

    clk_wiz_0 clk_wiz_instance
    (
    // Clock out ports
    .clk_106(display_clock),
    .clk_60(clk_60),  //60MHz
    .clk_8(clk_8),    // 8MHz
    // Clock in ports
    .clk_in1(clk)
    );

    // produce spi clock of 4MHz
    always@(posedge clk_8) begin
        if(!rst) begin
            spiclk_div <= 0;
            spiclk <= 0;
        end else begin
            if (spiclk_div) begin
                spiclk_div <= 0;
                spiclk <= ~spiclk;
            end else begin
                spiclk_div <= 1;
            end
        end
    end

    // produce game clock of 60Hz
    always@(posedge clk_60) begin
        if(!rst) begin
            gclk_div <= 0;
            gameclk <= 0;
        end else begin
            if (gclk_div >= 20'd1000000) begin
                gclk_div <= 0;
                gameclk <= ~gameclk;
            end else begin
                gclk_div <= gclk_div + 1;
            end
         end
    end

    assign game_clock = gameclk;
    assign spi_clock = spiclk;
endmodule
```

A.2 Graphics

Listing 3: drawcon.v

```
module draw_controller(
    input clk, rst,
    output [11:0] draw,
    input [10:0] x_coord, y_coord, player_pos_x, player_pos_y
    );

    // colour space generation
    reg [11:0] colourspace[0:63];
    integer i;
    reg[11:0] tcol = 12'hfff;

    // pointer wires
    wire [6:0] gradient_pointer;
    wire [6:0] grid_pointer;
    reg [6:0] selected_pointer;

    // screen size params
    parameter WIDTH = 11'd1426;     // total display width available
    parameter HEIGHT = 11'd898;     // ... height
    parameter BORDER = 11'd8;     // border on 'gameplay' edges
    parameter BANNER_BG_HEIGHT = 11'd81;  // size of top part of info banner at top of screen
    parameter BANNER_HEIGHT = 11'd130;  // size of info banner at top of screen
    // gameplay area has 1408x752 remaining available space

    // player control area
    parameter CONTROL_X = 11'd625 + BORDER;   // top corner reference coordinates
    parameter CONTROL_Y = 11'd297 + BANNER_HEIGHT + BORDER;
    parameter CONTROL_COLOUR_BORDER = 11'd10;   // size of colour selection border
    parameter CONTROL_BORDER = 11'd5;   // size of border of control area
    parameter CONTROL_WIDTH = 11'd157;  // overall width
    parameter CONTROL_HEIGHT = 11'd157; // ... height

    // colour selector parameters
    parameter GRADIENT_START_X = CONTROL_X + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
    parameter GRADIENT_START_Y = CONTROL_Y + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
    parameter GRADIENT_LENGTH = 128;

    // colour params
    parameter COL_BORDER = 12'h33A;
    parameter COL_BANNER_BG = 12'hAAF;
    parameter COL_BANNER_FLOOR = 12'h336;

    // Title sprite
    parameter TITLE_SPRITE_X = 200, TITLE_SPRITE_Y = 130;
    parameter TITLE_POS_X = 720;
    reg [14:0] title_addr;
    wire [11:0] col_title;

    // player sprite
    parameter RETICLE_SPRITE_LENGTH = 16;
    reg [7:0] reticle_addr;
    wire [11:0] col_reticle;

    // drawing pixel
    reg [11:0] pixel;
    reg [11:0] col_selected; // selected colour

    // ID sprite
    parameter ID_SPRITE_X = 61;
    parameter ID_SPRITE_Y = 31;
    parameter ID_SPRITE_OFFSET = 5;
    reg [10:0] id_addr;
    wire [11:0] col_id;

always@(posedge clk) begin
    // create colourspace
    for(i=0;i<64;i=i+1)begin
        colourspace[i] = tcol;
        tcol = tcol - 12'h020;
        if(tcol[7:4] == 4'hf) begin
            tcol = tcol - 12'h100;
        end
    end

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

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

Listing 4: colour_picker.v

```
module colour_picker(
    input clk, rst,
    input [10:0] x_coord, y_coord,
    output [6:0] pointer
    );

    // screen size params
    parameter BORDER = 8;     // border on 'gameplay' edges
    parameter BANNER_HEIGHT = 130;  // size of info banner at top of screen

    // player control area
    parameter CONTROL_X = 625 + BORDER;   // top corner reference coordinates
    parameter CONTROL_Y = 297 + BANNER_HEIGHT + BORDER;
    parameter CONTROL_COLOUR_BORDER = 10;   // size of colour selection border
    parameter CONTROL_BORDER = 5;   // size of border of control area

    // colour selector parameters
    parameter GRADIENT_START_X = CONTROL_X + CONTROL_COLOUR_BORDER + CONTROL_BORDER -1;
    parameter GRADIENT_START_Y = CONTROL_Y + CONTROL_COLOUR_BORDER + CONTROL_BORDER;
    parameter GRADIENT_LENGTH = 128;

    reg [8:0] col_pointer = 0;
    reg [8:0] start_pointer = 0;
    reg yflag = 1;

    integer i,j;
    reg [6:0] p = 0;

    always@(posedge clk) begin
        // 128x128
        if ((x_coord >= GRADIENT_START_X) && (x_coord < GRADIENT_START_X + GRADIENT_LENGTH ) &&
        (y_coord >= GRADIENT_START_Y) && (y_coord < GRADIENT_START_Y + GRADIENT_LENGTH)
        ) begin
            // create colour picker
            for(j=0;j<8;j=j+1)begin
                for(i=0;i<8;i=i+1)begin
                    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
    col_pointer <= p; end
                    p = p+1;
                    if(p == 64)begin p = 0; end
                end
            end
        end
    end

    assign pointer = col_pointer;

endmodule
```

Listing 5: colour_grid_i.sv

```
module colour_grid(
    input clk, rst,
    input [10:0] x_coord, y_coord,
    input [6:0] selected_pointer,
    output [6:0] pointer
    );

    // screen size params
    parameter BORDER = 11'd8;           // border on 'gameplay' edges
    parameter BANNER_HEIGHT = 11'd130;  // size of info banner at top of screen

    // 16 cells sizes
    parameter CELL_WIDTH = 88;
    parameter CELL_HEIGHT = 47;

    // adjacency
    reg [1:0] connected_array [0:15][0:15];   // 16x16 cell reference array, x/y
    reg [6:0] pointer_array [0:15][0:15];     // current pointer array
    reg [5:0] xcount, ycount;

    reg [6:0] col_pointer;

    integer i,j,k,h;

    // initialise arrays
    initial begin
        // adjacency array
        begin
            connected_array[0][0] <= 0;
            connected_array[1][0] <= 0;
            connected_array[2][0] <= 0;
            connected_array[3][0] <= 0;
            connected_array[4][0] <= 0;
            connected_array[5][0] <= 0;
            connected_array[6][0] <= 0;
            connected_array[7][0] <= 0;
            connected_array[8][0] <= 0;
            connected_array[9][0] <= 0;
            connected_array[10][0] <= 0;
            connected_array[11][0] <= 0;
            connected_array[12][0] <= 0;
            connected_array[13][0] <= 0;
            connected_array[14][0] <= 0;
            connected_array[15][0] <= 0;
            connected_array[0][1] <= 0;
            connected_array[1][1] <= 0;
            connected_array[2][1] <= 0;
            connected_array[3][1] <= 0;
            connected_array[4][1] <= 0;
            connected_array[5][1] <= 0;
            connected_array[6][1] <= 0;
            connected_array[7][1] <= 0;
            connected_array[8][1] <= 0;
            connected_array[9][1] <= 0;
            connected_array[10][1] <= 0;
            connected_array[11][1] <= 0;
            connected_array[12][1] <= 0;
            connected_array[13][1] <= 0;
            connected_array[14][1] <= 0;
            connected_array[15][1] <= 0;
            connected_array[0][2] <= 0;
            connected_array[1][2] <= 0;
            connected_array[2][2] <= 0;
            connected_array[3][2] <= 0;
            connected_array[4][2] <= 0;
            connected_array[5][2] <= 0;
            connected_array[6][2] <= 0;
            connected_array[7][2] <= 0;
            connected_array[8][2] <= 0;
            connected_array[9][2] <= 0;
            connected_array[10][2] <= 0;
            connected_array[11][2] <= 0;
            connected_array[12][2] <= 0;
            connected_array[13][2] <= 0;
            connected_array[14][2] <= 0;
            connected_array[15][2] <= 0;
            connected_array[0][3] <= 0;
            connected_array[1][3] <= 0;
```

```
            connected_array[2][3] <= 0;
            connected_array[3][3] <= 0;
            connected_array[4][3] <= 0;
            connected_array[5][3] <= 0;
            connected_array[6][3] <= 0;
            connected_array[7][3] <= 0;
            connected_array[8][3] <= 0;
            connected_array[9][3] <= 0;
            connected_array[10][3] <= 0;
            connected_array[11][3] <= 0;
            connected_array[12][3] <= 0;
            connected_array[13][3] <= 0;
            connected_array[14][3] <= 0;
            connected_array[15][3] <= 0;
            connected_array[0][4] <= 0;
            connected_array[1][4] <= 0;
            connected_array[2][4] <= 0;
            connected_array[3][4] <= 0;
            connected_array[4][4] <= 0;
            connected_array[5][4] <= 0;
            connected_array[6][4] <= 0;
            connected_array[7][4] <= 0;
            connected_array[8][4] <= 0;
            connected_array[9][4] <= 0;
            connected_array[10][4] <= 0;
            connected_array[11][4] <= 0;
            connected_array[12][4] <= 0;
            connected_array[13][4] <= 0;
            connected_array[14][4] <= 0;
            connected_array[15][4] <= 0;
            connected_array[0][5] <= 0;
            connected_array[1][5] <= 0;
            connected_array[2][5] <= 0;
            connected_array[3][5] <= 0;
            connected_array[4][5] <= 0;
            connected_array[5][5] <= 0;
            connected_array[6][5] <= 0;
            connected_array[7][5] <= 0;
            connected_array[8][5] <= 0;
            connected_array[9][5] <= 0;
            connected_array[10][5] <= 0;
            connected_array[11][5] <= 0;
            connected_array[12][5] <= 0;
            connected_array[13][5] <= 0;
            connected_array[14][5] <= 0;
            connected_array[15][5] <= 0;
            connected_array[0][6] <= 0;
            connected_array[1][6] <= 0;
            connected_array[2][6] <= 0;
            connected_array[3][6] <= 0;
            connected_array[4][6] <= 0;
            connected_array[5][6] <= 0;
            connected_array[6][6] <= 0;
            connected_array[7][6] <= 1;
            connected_array[8][6] <= 1;
            connected_array[9][6] <= 0;
            connected_array[10][6] <= 0;
            connected_array[11][6] <= 0;
            connected_array[12][6] <= 0;
            connected_array[13][6] <= 0;
            connected_array[14][6] <= 0;
            connected_array[15][6] <= 0;
            connected_array[0][7] <= 0;
            connected_array[1][7] <= 0;
            connected_array[2][7] <= 0;
            connected_array[3][7] <= 0;
            connected_array[4][7] <= 0;
            connected_array[5][7] <= 0;
            connected_array[6][7] <= 0;
            connected_array[7][7] <= 1;
            connected_array[8][7] <= 1;
            connected_array[9][7] <= 0;
            connected_array[10][7] <= 0;
            connected_array[11][7] <= 0;
            connected_array[12][7] <= 0;
            connected_array[13][7] <= 0;
            connected_array[14][7] <= 0;
            connected_array[15][7] <= 0;
            connected_array[0][8] <= 0;
```

```
            connected_array[1][8] <= 0;
            connected_array[2][8] <= 0;
            connected_array[3][8] <= 0;
            connected_array[4][8] <= 0;
            connected_array[5][8] <= 0;
            connected_array[6][8] <= 0;
            connected_array[7][8] <= 1;
            connected_array[8][8] <= 1;
            connected_array[9][8] <= 0;
            connected_array[10][8] <= 0;
            connected_array[11][8] <= 0;
            connected_array[12][8] <= 0;
            connected_array[13][8] <= 0;
            connected_array[14][8] <= 0;
            connected_array[15][8] <= 0;
            connected_array[0][9] <= 0;
            connected_array[1][9] <= 0;
            connected_array[2][9] <= 0;
            connected_array[3][9] <= 0;
            connected_array[4][9] <= 0;
            connected_array[5][9] <= 0;
            connected_array[6][9] <= 0;
            connected_array[7][9] <= 1;
            connected_array[8][9] <= 1;
            connected_array[9][9] <= 0;
            connected_array[10][9] <= 0;
            connected_array[11][9] <= 0;
            connected_array[12][9] <= 0;
            connected_array[13][9] <= 0;
            connected_array[14][9] <= 0;
            connected_array[15][9] <= 0;
            connected_array[0][10] <= 0;
            connected_array[1][10] <= 0;
            connected_array[2][10] <= 0;
            connected_array[3][10] <= 0;
            connected_array[4][10] <= 0;
            connected_array[5][10] <= 0;
            connected_array[6][10] <= 0;
            connected_array[7][10] <= 0;
            connected_array[8][10] <= 0;
            connected_array[9][10] <= 0;
            connected_array[10][10] <= 0;
            connected_array[11][10] <= 0;
            connected_array[12][10] <= 0;
            connected_array[13][10] <= 0;
            connected_array[14][10] <= 0;
            connected_array[15][10] <= 0;
            connected_array[0][11] <= 0;
            connected_array[1][11] <= 0;
            connected_array[2][11] <= 0;
            connected_array[3][11] <= 0;
            connected_array[4][11] <= 0;
            connected_array[5][11] <= 0;
            connected_array[6][11] <= 0;
            connected_array[7][11] <= 0;
            connected_array[8][11] <= 0;
            connected_array[9][11] <= 0;
            connected_array[10][11] <= 0;
            connected_array[11][11] <= 0;
            connected_array[12][11] <= 0;
            connected_array[13][11] <= 0;
            connected_array[14][11] <= 0;
            connected_array[15][11] <= 0;
            connected_array[0][12] <= 0;
            connected_array[1][12] <= 0;
            connected_array[2][12] <= 0;
            connected_array[3][12] <= 0;
            connected_array[4][12] <= 0;
            connected_array[5][12] <= 0;
            connected_array[6][12] <= 0;
            connected_array[7][12] <= 0;
            connected_array[8][12] <= 0;
            connected_array[9][12] <= 0;
            connected_array[10][12] <= 0;
            connected_array[11][12] <= 0;
            connected_array[12][12] <= 0;
            connected_array[13][12] <= 0;
            connected_array[14][12] <= 0;
            connected_array[15][12] <= 0;
```

```
            connected_array[0][13] <= 0;
            connected_array[1][13] <= 0;
            connected_array[2][13] <= 0;
            connected_array[3][13] <= 0;
            connected_array[4][13] <= 0;
            connected_array[5][13] <= 0;
            connected_array[6][13] <= 0;
            connected_array[7][13] <= 0;
            connected_array[8][13] <= 0;
            connected_array[9][13] <= 0;
            connected_array[10][13] <= 0;
            connected_array[11][13] <= 0;
            connected_array[12][13] <= 0;
            connected_array[13][13] <= 0;
            connected_array[14][13] <= 0;
            connected_array[15][13] <= 0;
            connected_array[0][14] <= 0;
            connected_array[1][14] <= 0;
            connected_array[2][14] <= 0;
            connected_array[3][14] <= 0;
            connected_array[4][14] <= 0;
            connected_array[5][14] <= 0;
            connected_array[6][14] <= 0;
            connected_array[7][14] <= 0;
            connected_array[8][14] <= 0;
            connected_array[9][14] <= 0;
            connected_array[10][14] <= 0;
            connected_array[11][14] <= 0;
            connected_array[12][14] <= 0;
            connected_array[13][14] <= 0;
            connected_array[14][14] <= 0;
            connected_array[15][14] <= 0;
            connected_array[0][15] <= 0;
            connected_array[1][15] <= 0;
            connected_array[2][15] <= 0;
            connected_array[3][15] <= 0;
            connected_array[4][15] <= 0;
            connected_array[5][15] <= 0;
            connected_array[6][15] <= 0;
            connected_array[7][15] <= 0;
            connected_array[8][15] <= 0;
            connected_array[9][15] <= 0;
            connected_array[10][15] <= 0;
            connected_array[11][15] <= 0;
            connected_array[12][15] <= 0;
            connected_array[13][15] <= 0;
            connected_array[14][15] <= 0;
            connected_array[15][15] <= 0;
        end
        // 'random' grid colours
        begin
            pointer_array[0][0] <= 34;
            pointer_array[1][0] <= 40;
            pointer_array[2][0] <= 32;
            pointer_array[3][0] <= 55;
            pointer_array[4][0] <= 50;
            pointer_array[5][0] <= 41;
            pointer_array[6][0] <= 39;
            pointer_array[7][0] <= 45;
            pointer_array[8][0] <= 63;
            pointer_array[9][0] <= 31;
            pointer_array[10][0] <= 19;
            pointer_array[11][0] <= 52;
            pointer_array[12][0] <= 17;
            pointer_array[13][0] <= 0;
            pointer_array[14][0] <= 28;
            pointer_array[15][0] <= 46;
            pointer_array[0][1] <= 31;
            pointer_array[1][1] <= 53;
            pointer_array[2][1] <= 30;
            pointer_array[3][1] <= 60;
            pointer_array[4][1] <= 56;
            pointer_array[5][1] <= 18;
            pointer_array[6][1] <= 26;
            pointer_array[7][1] <= 25;
            pointer_array[8][1] <= 1;
            pointer_array[9][1] <= 5;
            pointer_array[10][1] <= 23;
            pointer_array[11][1] <= 30;
```

```
            pointer_array[12][1] <= 43;
            pointer_array[13][1] <= 8;
            pointer_array[14][1] <= 13;
            pointer_array[15][1] <= 28;
            pointer_array[0][2] <= 51;
            pointer_array[1][2] <= 17;
            pointer_array[2][2] <= 43;
            pointer_array[3][2] <= 30;
            pointer_array[4][2] <= 4;
            pointer_array[5][2] <= 12;
            pointer_array[6][2] <= 27;
            pointer_array[7][2] <= 25;
            pointer_array[8][2] <= 19;
            pointer_array[9][2] <= 8;
            pointer_array[10][2] <= 53;
            pointer_array[11][2] <= 57;
            pointer_array[12][2] <= 30;
            pointer_array[13][2] <= 38;
            pointer_array[14][2] <= 38;
            pointer_array[15][2] <= 60;
            pointer_array[0][3] <= 54;
            pointer_array[1][3] <= 22;
            pointer_array[2][3] <= 37;
            pointer_array[3][3] <= 22;
            pointer_array[4][3] <= 19;
            pointer_array[5][3] <= 10;
            pointer_array[6][3] <= 21;
            pointer_array[7][3] <= 31;
            pointer_array[8][3] <= 40;
            pointer_array[9][3] <= 55;
            pointer_array[10][3] <= 20;
            pointer_array[11][3] <= 57;
            pointer_array[12][3] <= 23;
            pointer_array[13][3] <= 25;
            pointer_array[14][3] <= 9;
            pointer_array[15][3] <= 18;
            pointer_array[0][4] <= 42;
            pointer_array[1][4] <= 17;
            pointer_array[2][4] <= 26;
            pointer_array[3][4] <= 31;
            pointer_array[4][4] <= 12;
            pointer_array[5][4] <= 17;
            pointer_array[6][4] <= 38;
            pointer_array[7][4] <= 34;
            pointer_array[8][4] <= 11;
            pointer_array[9][4] <= 4;
            pointer_array[10][4] <= 51;
            pointer_array[11][4] <= 18;
            pointer_array[12][4] <= 32;
            pointer_array[13][4] <= 38;
            pointer_array[14][4] <= 40;
            pointer_array[15][4] <= 4;
            pointer_array[0][5] <= 45;
            pointer_array[1][5] <= 18;
            pointer_array[2][5] <= 12;
            pointer_array[3][5] <= 41;
            pointer_array[4][5] <= 36;
            pointer_array[5][5] <= 44;
            pointer_array[6][5] <= 0;
            pointer_array[7][5] <= 53;
            pointer_array[8][5] <= 9;
            pointer_array[9][5] <= 16;
            pointer_array[10][5] <= 2;
            pointer_array[11][5] <= 7;
            pointer_array[12][5] <= 0;
            pointer_array[13][5] <= 47;
            pointer_array[14][5] <= 41;
            pointer_array[15][5] <= 33;
            pointer_array[0][6] <= 21;
            pointer_array[1][6] <= 32;
            pointer_array[2][6] <= 4;
            pointer_array[3][6] <= 38;
            pointer_array[4][6] <= 56;
            pointer_array[5][6] <= 61;
            pointer_array[6][6] <= 1;
            pointer_array[7][6] <= 43;
            pointer_array[8][6] <= 31;
            pointer_array[9][6] <= 5;
            pointer_array[10][6] <= 14;
```

```
            pointer_array[11][6] <= 40;
            pointer_array[12][6] <= 1;
            pointer_array[13][6] <= 16;
            pointer_array[14][6] <= 54;
            pointer_array[15][6] <= 3;
            pointer_array[0][7] <= 55;
            pointer_array[1][7] <= 38;
            pointer_array[2][7] <= 28;
            pointer_array[3][7] <= 46;
            pointer_array[4][7] <= 5;
            pointer_array[5][7] <= 47;
            pointer_array[6][7] <= 51;
            pointer_array[7][7] <= 60;
            pointer_array[8][7] <= 52;
            pointer_array[9][7] <= 52;
            pointer_array[10][7] <= 33;
            pointer_array[11][7] <= 18;
            pointer_array[12][7] <= 42;
            pointer_array[13][7] <= 45;
            pointer_array[14][7] <= 41;
            pointer_array[15][7] <= 26;
            pointer_array[0][8] <= 26;
            pointer_array[1][8] <= 20;
            pointer_array[2][8] <= 58;
            pointer_array[3][8] <= 38;
            pointer_array[4][8] <= 55;
            pointer_array[5][8] <= 41;
            pointer_array[6][8] <= 48;
            pointer_array[7][8] <= 27;
            pointer_array[8][8] <= 5;
            pointer_array[9][8] <= 45;
            pointer_array[10][8] <= 32;
            pointer_array[11][8] <= 54;
            pointer_array[12][8] <= 56;
            pointer_array[13][8] <= 11;
            pointer_array[14][8] <= 11;
            pointer_array[15][8] <= 55;
            pointer_array[0][9] <= 15;
            pointer_array[1][9] <= 58;
            pointer_array[2][9] <= 4;
            pointer_array[3][9] <= 55;
            pointer_array[4][9] <= 33;
            pointer_array[5][9] <= 62;
            pointer_array[6][9] <= 56;
            pointer_array[7][9] <= 50;
            pointer_array[8][9] <= 39;
            pointer_array[9][9] <= 9;
            pointer_array[10][9] <= 23;
            pointer_array[11][9] <= 38;
            pointer_array[12][9] <= 16;
            pointer_array[13][9] <= 37;
            pointer_array[14][9] <= 54;
            pointer_array[15][9] <= 51;
            pointer_array[0][10] <= 19;
            pointer_array[1][10] <= 61;
            pointer_array[2][10] <= 41;
            pointer_array[3][10] <= 25;
            pointer_array[4][10] <= 25;
            pointer_array[5][10] <= 46;
            pointer_array[6][10] <= 9;
            pointer_array[7][10] <= 31;
            pointer_array[8][10] <= 3;
            pointer_array[9][10] <= 27;
            pointer_array[10][10] <= 63;
            pointer_array[11][10] <= 42;
            pointer_array[12][10] <= 47;
            pointer_array[13][10] <= 18;
            pointer_array[14][10] <= 0;
            pointer_array[15][10] <= 63;
            pointer_array[0][11] <= 28;
            pointer_array[1][11] <= 4;
            pointer_array[2][11] <= 45;
            pointer_array[3][11] <= 31;
            pointer_array[4][11] <= 55;
            pointer_array[5][11] <= 30;
            pointer_array[6][11] <= 22;
            pointer_array[7][11] <= 13;
            pointer_array[8][11] <= 32;
            pointer_array[9][11] <= 16;
```

```
            pointer_array[10][11] <= 40;
            pointer_array[11][11] <= 2;
            pointer_array[12][11] <= 15;
            pointer_array[13][11] <= 60;
            pointer_array[14][11] <= 40;
            pointer_array[15][11] <= 39;
            pointer_array[0][12] <= 6;
            pointer_array[1][12] <= 22;
            pointer_array[2][12] <= 23;
            pointer_array[3][12] <= 8;
            pointer_array[4][12] <= 1;
            pointer_array[5][12] <= 30;
            pointer_array[6][12] <= 44;
            pointer_array[7][12] <= 12;
            pointer_array[8][12] <= 56;
            pointer_array[9][12] <= 3;
            pointer_array[10][12] <= 5;
            pointer_array[11][12] <= 41;
            pointer_array[12][12] <= 18;
            pointer_array[13][12] <= 28;
            pointer_array[14][12] <= 8;
            pointer_array[15][12] <= 44;
            pointer_array[0][13] <= 11;
            pointer_array[1][13] <= 29;
            pointer_array[2][13] <= 2;
            pointer_array[3][13] <= 24;
            pointer_array[4][13] <= 44;
            pointer_array[5][13] <= 37;
            pointer_array[6][13] <= 40;
            pointer_array[7][13] <= 26;
            pointer_array[8][13] <= 61;
            pointer_array[9][13] <= 26;
            pointer_array[10][13] <= 52;
            pointer_array[11][13] <= 20;
            pointer_array[12][13] <= 12;
            pointer_array[13][13] <= 27;
            pointer_array[14][13] <= 0;
            pointer_array[15][13] <= 55;
            pointer_array[0][14] <= 46;
            pointer_array[1][14] <= 10;
            pointer_array[2][14] <= 2;
            pointer_array[3][14] <= 36;
            pointer_array[4][14] <= 43;
            pointer_array[5][14] <= 33;
            pointer_array[6][14] <= 60;
            pointer_array[7][14] <= 5;
            pointer_array[8][14] <= 33;
            pointer_array[9][14] <= 44;
            pointer_array[10][14] <= 5;
            pointer_array[11][14] <= 26;
            pointer_array[12][14] <= 23;
            pointer_array[13][14] <= 8;
            pointer_array[14][14] <= 17;
            pointer_array[15][14] <= 49;
            pointer_array[0][15] <= 16;
            pointer_array[1][15] <= 35;
            pointer_array[2][15] <= 4;
            pointer_array[3][15] <= 62;
            pointer_array[4][15] <= 11;
            pointer_array[5][15] <= 20;
            pointer_array[6][15] <= 13;
            pointer_array[7][15] <= 51;
            pointer_array[8][15] <= 53;
            pointer_array[9][15] <= 34;
            pointer_array[10][15] <= 39;
            pointer_array[11][15] <= 41;
            pointer_array[12][15] <= 31;
            pointer_array[13][15] <= 61;
            pointer_array[14][15] <= 26;
            pointer_array[15][15] <= 24;
        end
    end

    always@(posedge clk) begin
        // check currently connected cells
        if(connected_array[xcount][ycount] == 1) begin
            //if matched
            if(pointer_array[xcount][ycount] == selected_pointer) begin
                // 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
     +1) &&
582                     (y_coord >= BORDER + BANNER_HEIGHT + k*CELL_HEIGHT) && (y_coord < BORDER +
    BANNER_HEIGHT + k*CELL_HEIGHT + CELL_HEIGHT+1)
583                     )begin col_pointer <= pointer_array[h][k]; end
584             end
585         end
586 
587         // cell position counters
588         if(ycount > 15) begin
589             ycount <= 0;
590             xcount <= 0;
591         end else if(xcount > 15) begin
592             xcount <= 0;
593             ycount <= ycount + 1;
594         end else begin
595             xcount <= xcount + 1;
596         end
597     end
598 
599     assign pointer = col_pointer;
600 endmodule
```

A.3 Display

Listing 6: vga_out.v

```
module vga_controller(
    input clk, rst,
    output [11:0] colour,
    input [11:0] draw,
    output hsync, vsync,
    output [10:0] x_coord, y_coord
    );

    //display parameters
    // horizontal parameters
    parameter DISPLAYH_MAX = 11'd1824;
    parameter DISPLAYH_MIN = 11'd384;
    parameter HSYNC_START = 11'd151;
    parameter HCOUNT_MAX = 11'd1903;

    // vertical parameters
    parameter DISPLAYV_MAX = 10'd931;
    parameter DISPLAYV_MIN = 10'd31;
    parameter VSYNC_START = 10'd2;
    parameter VCOUNT_MAX = 10'd931;

    // background colour
    parameter COL_BLACK = 12'h000;

    //internal signals
    reg [10:0] hcount;
    reg [9:0] vcount;
    reg [10:0] x_reg;
    reg [10:0] y_reg;

    wire display_region;
    wire v_region;
    wire h_end = (hcount == HCOUNT_MAX);
    wire v_end = (vcount == VCOUNT_MAX);

    wire [3:0]draw_r;
    wire [3:0]draw_g;
    wire [3:0]draw_b;

    //hsync vsync combinational
    assign hsync = ((hcount >= 11'd0) && (hcount <= HSYNC_START));
    assign vsync = ((vcount >= 10'd0) && (vcount <= VSYNC_START));

    assign display_region = ((hcount >= DISPLAYH_MIN) && (hcount <= DISPLAYH_MAX) && (vcount >=
    DISPLAYV_MIN) && (vcount <= DISPLAYV_MAX));

    assign colour = (display_region) ? draw : COL_BLACK;

    always@(posedge clk) begin
        if (!rst) begin
            hcount <= 11'd0;
            vcount <= 10'd0;
            x_reg <= 11'd0;
            y_reg <= 11'd0;
        end
        else begin
            // hcount counter
            if (h_end) begin
                hcount <= 11'd0;
            end else begin
                hcount <= hcount + 11'd1;
            end
            // vcount counter
            if (v_end) begin
                vcount <= 10'd0;
            end else if (h_end) begin
                vcount <= vcount + 10'd1;
            end
            //x coordinate counter
            if((hcount >= DISPLAYH_MIN) && (hcount <= DISPLAYH_MAX)) begin
                x_reg <= x_reg + 11'd1;
            end else begin
                x_reg <= 11'd0;
            end
            //y coordinate counter
            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

```
module seginterface(
        input clk, rst,
        output div_clk,
        output a, b, c, d, e, f, g,
        output [7:0] an
    );

    wire led_clk;
    reg [3:0] dig_sel;

    reg [28:0] clk_count = 11'd0;

    always @(posedge clk)
        clk_count <= clk_count + 1'b1;

    assign led_clk = clk_count[16];
    assign div_clk = clk_count[25];

    reg [7:0] led_strobe = 8'b11111110;
    always @(posedge led_clk)
        led_strobe <= {led_strobe[6:0],led_strobe[7]};
    assign an = led_strobe;

    reg [2:0] led_index = 3'd0;
    always @(posedge led_clk)
        led_index <= led_index + 1'b1;

    always@*
        case (led_index)
            3'd0: dig_sel = 4'hd;
            3'd1: dig_sel = 4'h0;
            3'd2: dig_sel = 4'h0;
            3'd3: dig_sel = 4'h0;
            3'd4: dig_sel = 4'h0;
            3'd5: dig_sel = 4'h0;
            3'd6: dig_sel = 4'h1;
            3'd7: dig_sel = 4'hf;
        endcase

    sevenseg M1(.num(dig_sel), .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g));

endmodule
```

Listing 8: sevenseg.v

```
`timescale 1ns / 1ps

module sevenseg(
    input [3:0] num,
    output a,
    output b,
    output c,
    output d,
    output e,
    output f,
    output g
    );

    reg [6:0] intseg;
    assign {a,b,c,d,e,f,g} = ~intseg;

    always@*
    begin
        case(num)
            4'h0: intseg = 7'b0011101;
            4'h1: intseg = 7'b0110000;
            4'h2: intseg = 7'b1101101;
            4'h3: intseg = 7'b1111001;
            4'h4: intseg = 7'b0110011;
            4'h5: intseg = 7'b1011011;
            4'h6: intseg = 7'b1011111;
            4'h7: intseg = 7'b1110000;
            4'h8: intseg = 7'b1111111;
            4'h9: intseg = 7'b1111011;
            4'ha: intseg = 7'b1110111;
            4'hb: intseg = 7'b0011111;
            4'hc: intseg = 7'b1001110;
            4'hd: intseg = 7'b0111101;
            4'he: intseg = 7'b1001111;
            4'hf: intseg = 7'b1000111;
        endcase
    end


endmodule
```