6.111 Final Project:

day-of-Flight Planning Guidance Simulator (FPGA)

Sophia Wang, Pranav Arunandhi

day-of-Flight Planning Guidance simulAtor (FPGA)

Pranav Arunandhi, Sophia Wang

Overview/Motivation

As aspiring AeroAstro students, we watch firsthand with excitement the incredible growth of the space field in recent years. Part of this growth has resulted in the rapid launch of payloads into space. With the amount of payloads currently in space, from active satellites such as GPS and Starlink, to space debris and junk from projects that have gone past their prime, selecting an optimal launch time is a prevalent challenge to aerospace endeavors. Our project seeks to address this problem by computing and displaying information regarding the optimal launch time and altitude for a given time window and goal altitude defined by the user, so that they can make the best decision for their launch.

Total System Diagram

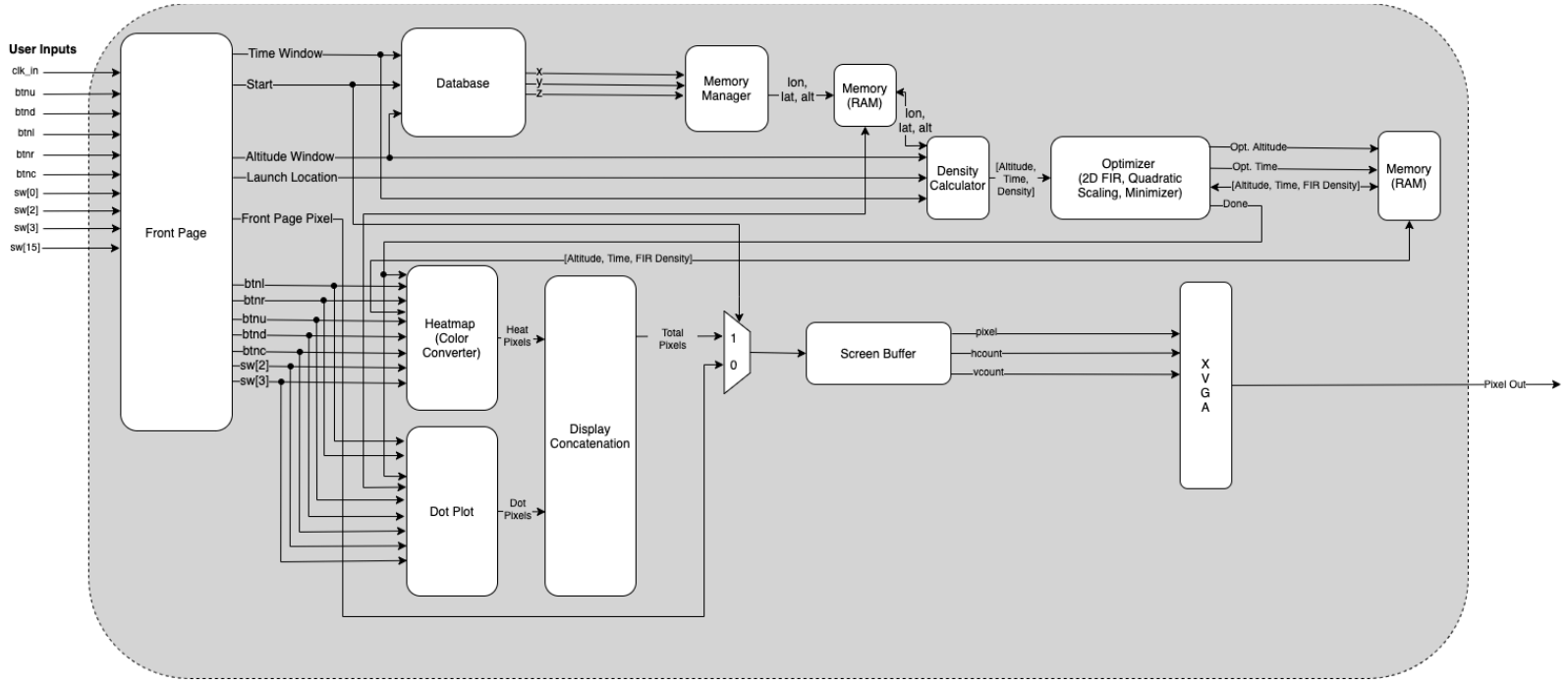

Figure 1. System Diagram

Using button and switch inputs, the user will input time and altitude parameters (e.g. 0 - 60 minutes, 400 - 900 km) that they would like to launch within. Time is discretized by 10 minute

intervals, while altitude is discretized by 50 km intervals. The goal of this system is to find the optimal time and altitude within the user's input range to launch their payload into orbit. This logic is controlled by the Front Page module. When the user submits the parameters via switch, a start pulse is sent to the Database module, which retrieves all relevant location data for active satellites in Low Earth Orbit (~4,000 satellites actively tracked). This data is then propagated forward in time and sent to the Memory Manager module, which writes this data into the BRAM of the FPGA. Once all the relevant satellite data is stored into memory, the density calculator module begins calculating the number of payloads found within a circle centered at the launch location (pre-set as the Johnson Space Center in Houston, TX) for each discretized (time, altitude) coordinate within the user's input parameters. Following this density calculation, the optimizer module runs a 2D FIR scheme, a quadratic scaler, and a real-time minimizer to determine the optimal (time, altitude) coordinate. Once these computations are done, the minimum will be passed to the display module, which will use the heatmap and dot plot modules to determine what color should be displayed in the background and where dots representing satellites should be on the screen, and display the time and altitude using a font ROM on the screen. The user can also use the buttons to move to other time and altitude values (or reset back to the optimum), or flip a switch to animate in one of those directions, and the display will update accordingly.

Major Modules

Front Page (Sophia)

Taking as input four buttons, two switches (left, right, up, down, sw[0], sw[15]) and the system clock, this module will allow users to choose a launch altitude window (e.g. 800 to 850 kilometers) and a launch time window (e.g. one hour from the current time to two hours from the current time). The landing screen, displayed below in Figure 2, works when the user switches off the reset switch (sw[15]), and begins moving up and down. The up and down buttons control which parameters the user is inputting, while the left and right buttons control the values for each of these parameters. For time, pressing and releasing the button will increment the value by 10 minutes, with a maximum range of 0 to 60 minutes and optimization range of one hour. For altitude, pressing and releasing the button will increment the value by 50 kilometers, with a maximum range from 250 kilometers to 2000 kilometers (the boundaries of Low Earth Orbit (LEO) where most payloads, specifically satellite constellations and CubeSats, are launched) and maximum optimization range of 500 kilometers. As part of the user interface, an arrow appears at the parameter of whichever quantity the user is editing. When the user is satisfied with their input parameters, they switch on the optimization switch (sw[0]), and the screen displayed by the XVGA transitions from the front page/landing page to a page which graphically displays the results of the launch window optimization. Within this switch submission action, there is also basic logic which prevents a user from submitted a set of parameters where: (time_end < time_start) || (time_end - time_start < 20 minutes) || (alt_end < alt_start) || (alt_end - alt_start < 100 km).

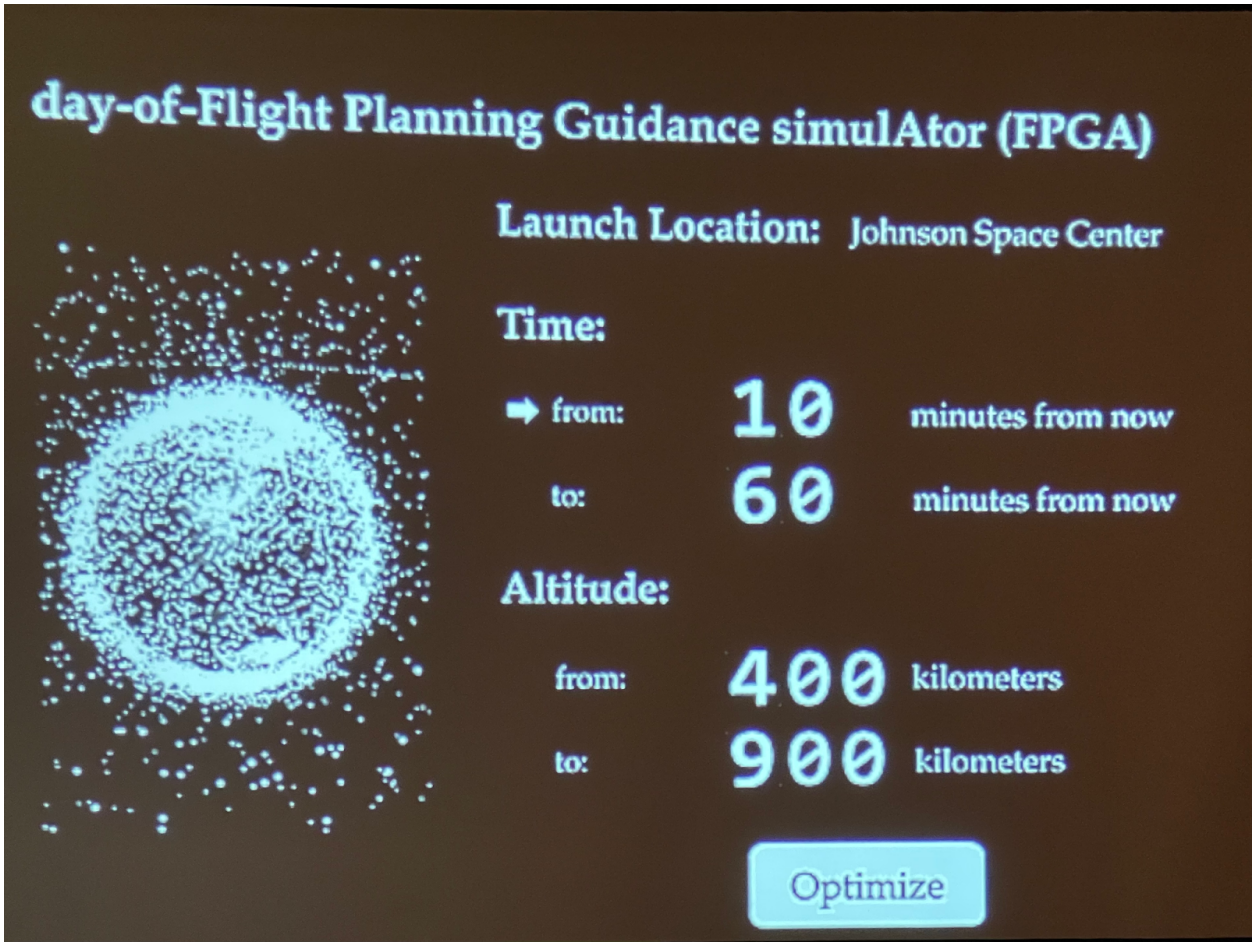

Figure 2: Landing page for user inputs

This front page was implemented using three ROMs: a 512x384 pixel background ROM, a digit ROM (24x24 pixels for each digit, with a total of ten digits), and an arrow ROM (45x45 pixels). The background and digit ROMs were each scaled down by a factor of two in each dimension, therefore reducing the overall size of each ROM by a factor of four. This meant that when the address to read the ROM was fed into each instance of the relevant ROM, we needed to right shift by one:

Background Image:

`address = (hcount>>1) + (vcount>>1)*image_height`

Digits:

`address = (hcount>>1) + (vcount>>1)*image_height + digit_factor`, where `digit_factor` is an integer which determines which address to start reading from, since the digit ROM contains the bits used for 10 different digits.

A total of 14 instances of ROMs were used to create the front page display, shown below in Figure 3. Each red line and arrow represents a different ROM instance.

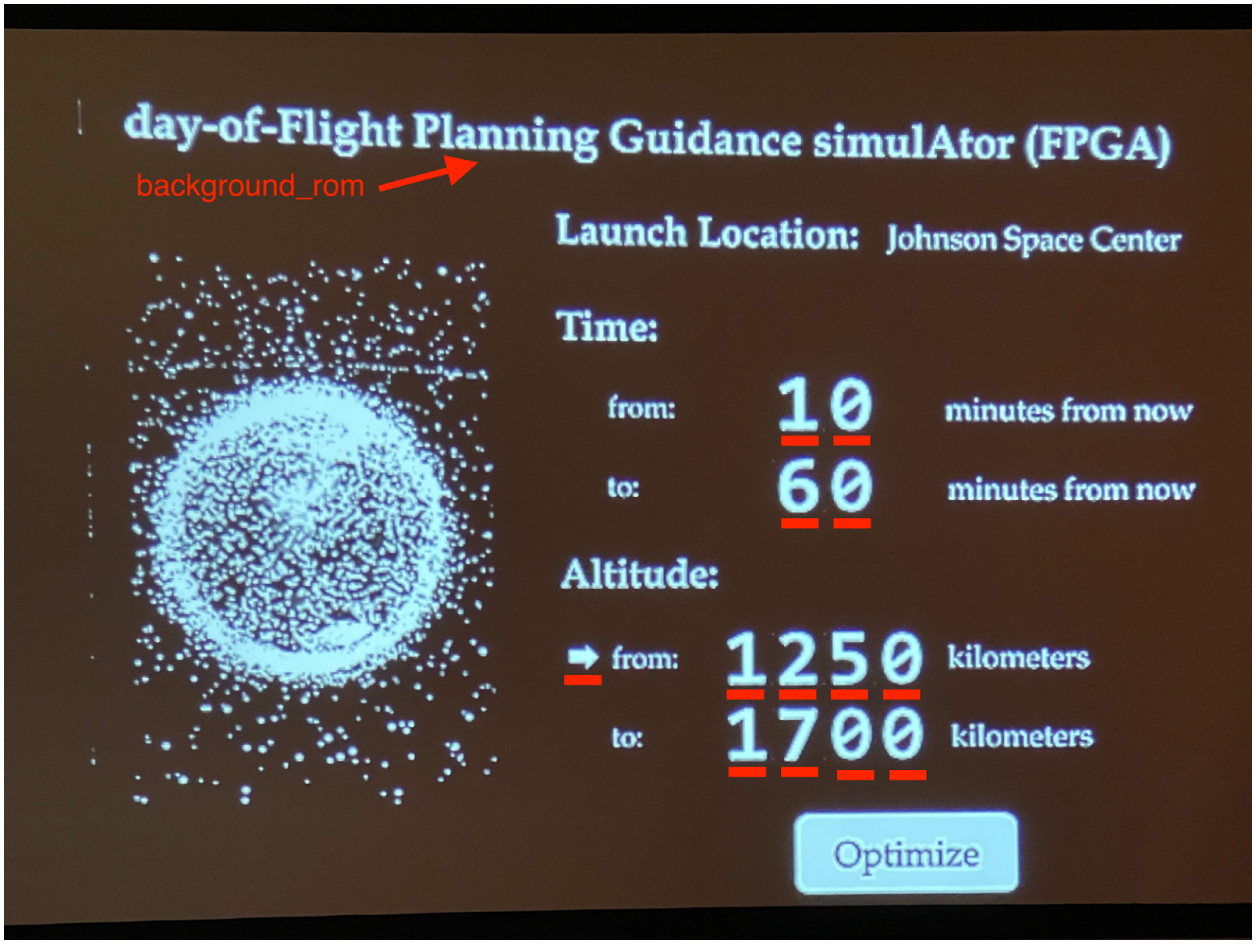

Figure 3: Instances of ROM

There were several memory constraints in the generation of the above-described ROMs, which proved to be a major challenge in the development of our project. These memory limitations and challenges are described in the *Challenges* section of our report. Succinctly put, to get around the memory limitations, many images (e.g. landing screen background, digits) were reduced in size and then put through an anti-aliasing filter and no color maps were used (the pixel information was reduced to grayscale, where the eight bits in the original image_rom for the picture could be used to generate the desired display).

The outputs to this module are the altitude window, time window, launch location, start (start = 0 if launch parameters are not yet chosen, start = 1 if launch parameters have been chosen and submitted), the pixels for the front page/selection screen, and the buttons and switches (these system inputs are relevant to the heatmap and dot plot modules described in detail below, as users can scrub through time and altitude to visualize changes in launch performance).

Database (Sophia)

When a start pulse is asserted, indicating that optimization can begin, the system starts by taking the most recent data of actively tracked satellites in LEO propagated forward in time. This module is the only part of our overall system which uses a Python script instead of SystemVerilog. The reason why we decided in our project design to include this component of non-digital design is based on the complexity of orbital propagation. The Simplified General Perturbations (SGP4) propagator used internationally, including by the United States Space Surveillance Network, is a complex algorithm composed of five mathematical models. For the purposes of time and complexity, as well as the focus of our project (optimization vs. propagation), we opted to use the SGP4 Python package to perform this computation. Implementing the SGP4 algorithm could possibly be considered a project in and of itself (and it is certainly a topic we are interested in pursuing in future projects!).

Since this course is focused on digital design, the description of the Python script used to generate satellite location data will only be described briefly below:

Using the Celestrak API, the most recent Two-Line Element (TLE) data describing the orbital elements (e.g. apogee, perigee, etc.) of each actively tracked satellite in LEO is pulled. Then, the Julian Date (and its corresponding fractional time) is calculated both for the current time and for each time the satellites are propagated forward. Using the SGP4 package, we input the current time frame, the propagated time of interest, and the TLE data to retrieve the location (in True Equation Mean Equinox (TEME) format), the velocity (also in TEME format), and the expected error in propagation. The TEME formatted data is then transformed into International Terrestrial Reference Frame (ITRS) (a geocentric coordinate system) format, where we can retrieve the longitude, latitude, and altitude of each time-propagated satellite. Following this transformation, the location data is written into a .csv file. An important note is that we added 90 and 180 degrees, respectively, to every latitude and longitude coordinate. When writing into memory, we aimed to avoid signed logic, which means that an offset would be added to all negative values. An example of an offset being applied is shown below for our pre-set launch location, the Johnson Space Center:

Johnson Space Center: 29.5593° N, 95.0900° W → 120°, 85°

Then, the data from each satellite coordinate (satellite, time, altitude) is transformed into a 33-bit number (explained in detail in the next section/module), which is then written into memory using the Memory Manager module.

As a check that this satellite data propagated through Python is reasonable, we generated the plot below (Figure 4) which shows the longitude and latitude of different satellites in a 2D heat map (note: latitude and longitude are offset by 90°, 180°, respectively). This is as we expected, with the orbital shape of denser occurrences of satellites, as well as the bright band at the equator, where most payloads are launched (the surface of the Earth travels faster there, so that the spacecraft moves faster when launched around this location).

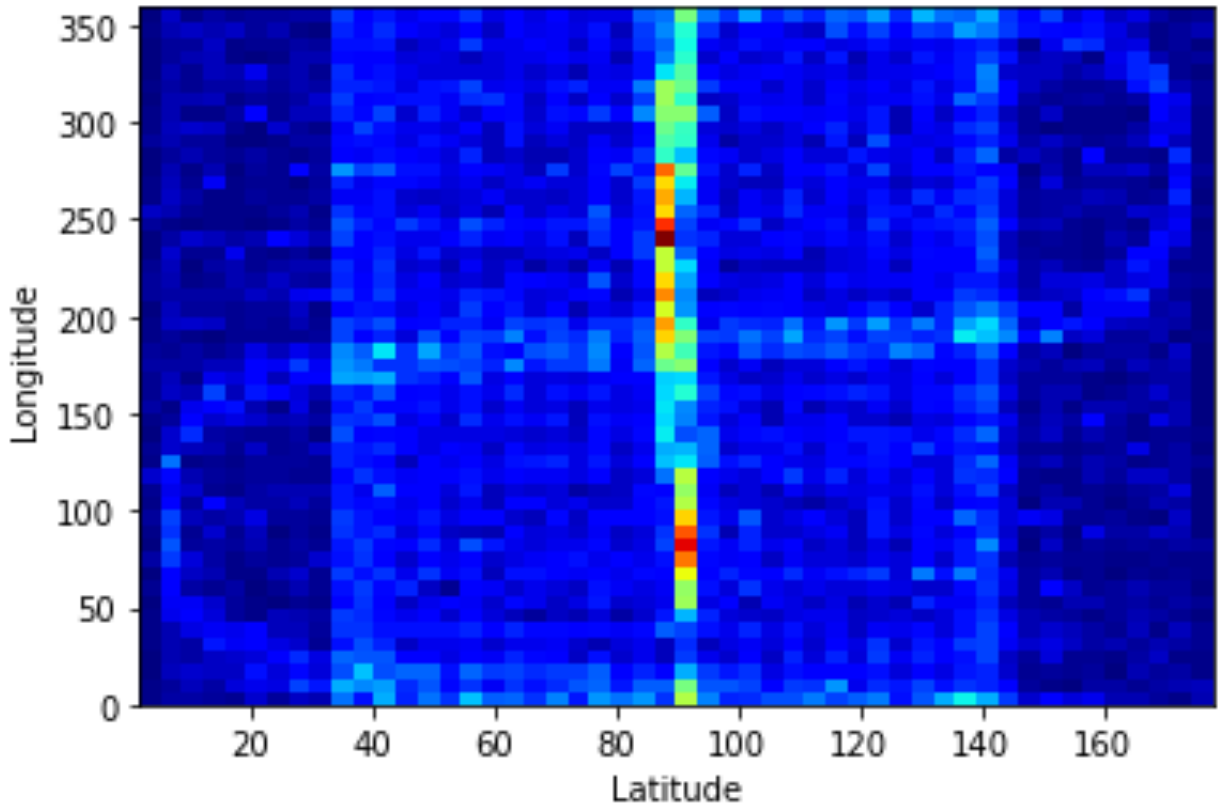

Figure 4: Heat Map of Propagated Satellites

Memory Manager (Sophia)

After fetching the propagated satellite data generated from the Python script, we write the data into BRAM. The size of our BRAM was a 33 bit width and 30303 sample depth (this accounted for 4329 actively tracked satellites). When writing into the BRAM, 33 bits were allocated, in which the most significant 8 bits represented the latitude, the next 9 bits represented the longitude, and the least significant 16 bits represented the altitude. An example 33-bit data is shown below in Figure 5:

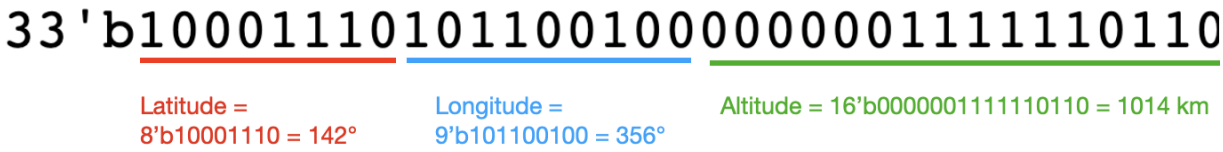

Figure 5: 33 bits representing satellite location

When working with the BRAM, we waited two cycles between every read and write. After all ~30,000 data points are written into BRAM, a start pulse was asserted, which indicated to the next module (Density Calculator) that further computations could be performed.

Density Calculator (Sophia)

This module calculates the density of satellites within the volume of a cylinder with a circular base of radius 32° (both in longitude and in latitude) centered at the launch location, and a height of a discretized altitude coordinate. The calculation (Figure 6), as well as a visualization of this cylinder (Figure 7) is shown below:

$$(1) \quad (sat_{lat}(time) - launch_{lat})^2 + (sat_{lon}(time) - launch_{lon})^2 \leq radius^2$$

$$(2) \quad sat_{alt} \leq altitude$$

$$where\ time, altitude = \ current\ (time, location)\ coordinate$$

$$if\ (1)\ \&\&\ (2)\ are\ satisfied, then\ increment\ satellite\_counter$$

Figure 6: Density counter logic

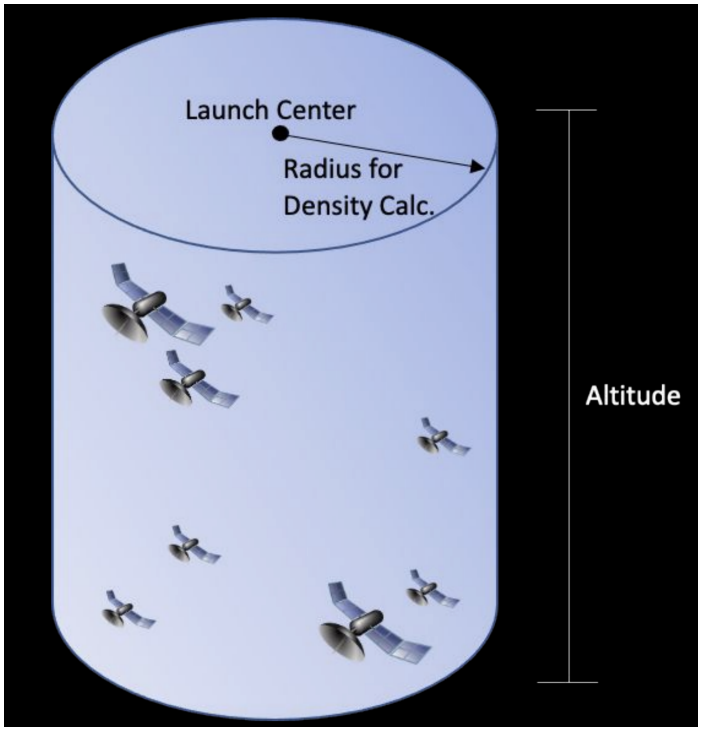

Figure 7: Cylinder used for density Calculation

At every (time, altitude) coordinate, this density is computed, then written into a separate BRAM (density_ram). The size of density_ram is: 10 bit width (no more than 1023 satellites will reasonably be within this volume), 77 sample depth.

Optimizer (Pranav)

Part 1: 2D-FIR Filter

This module is a standard 9 input FIR filter that returns the sum of the previous 9 inputs. Its behaviour therefore depends on the data values that are passed into it by the top level module controlling it. We begin by treating the density BRAM, which has depth 77, as a 7-by-11 array, as shown in Figure 8. Then, we iterate through altitude, then increment time when the altitude reaches the end of its range and needs to reset. This allows us to minimize the time spent waiting for the FIR to have the correct 9 values. With this scheme, each time we increment altitude, we have to wait for 3 new inputs to be sent to the FIR, and each time we increment time (and reset altitude), we have to wait for 9 new inputs to be sent to the FIR.

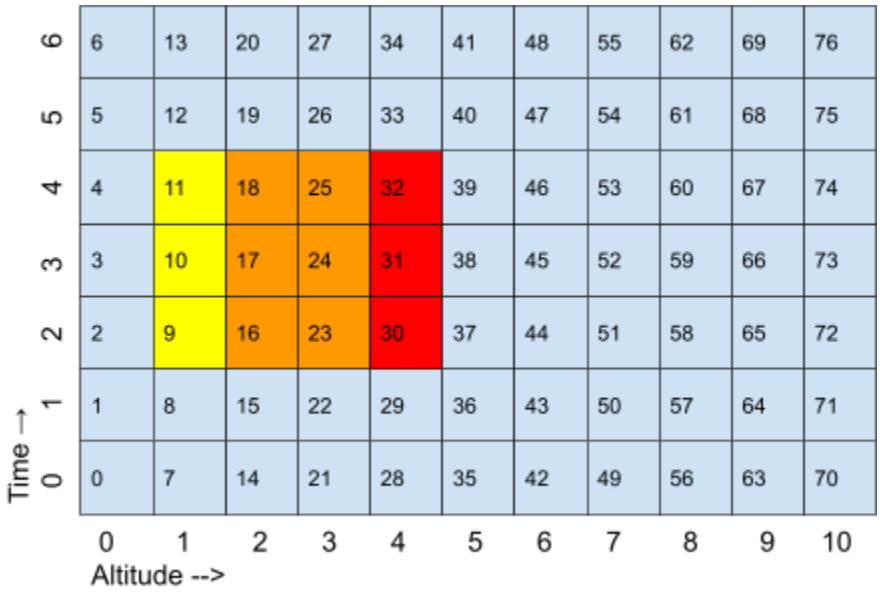

Figure 8: 2D-FIR data input scheme. Marked in yellow and orange are the data addresses that need to be used to have a valid convolution at the (altitude, time) coordinate (2, 3), and marked in orange and red are the addresses that are needed for (3, 3).

Part 2: Quadratic Scaler

When we have a valid output from the FIR module (i.e., the correct number of data points have been passed in and the computation is complete), then we pass the value into a quadratic scaler which conducts the following computation over multiple cycles:

```
output = input + input*(input - (2^14))*weight/(2^18)
```

This will allow us to approximate the set of curves shown in Figure 9. The weight in the formula represents the discretized altitude bucket of the current density, and as such the quadratic term effectively acts as a regularizer that is more significant at higher altitudes; this allows us to

account for the fact that there will be more satellites as our altitude increases, yet we still want our launch to go as high into the window as possible.

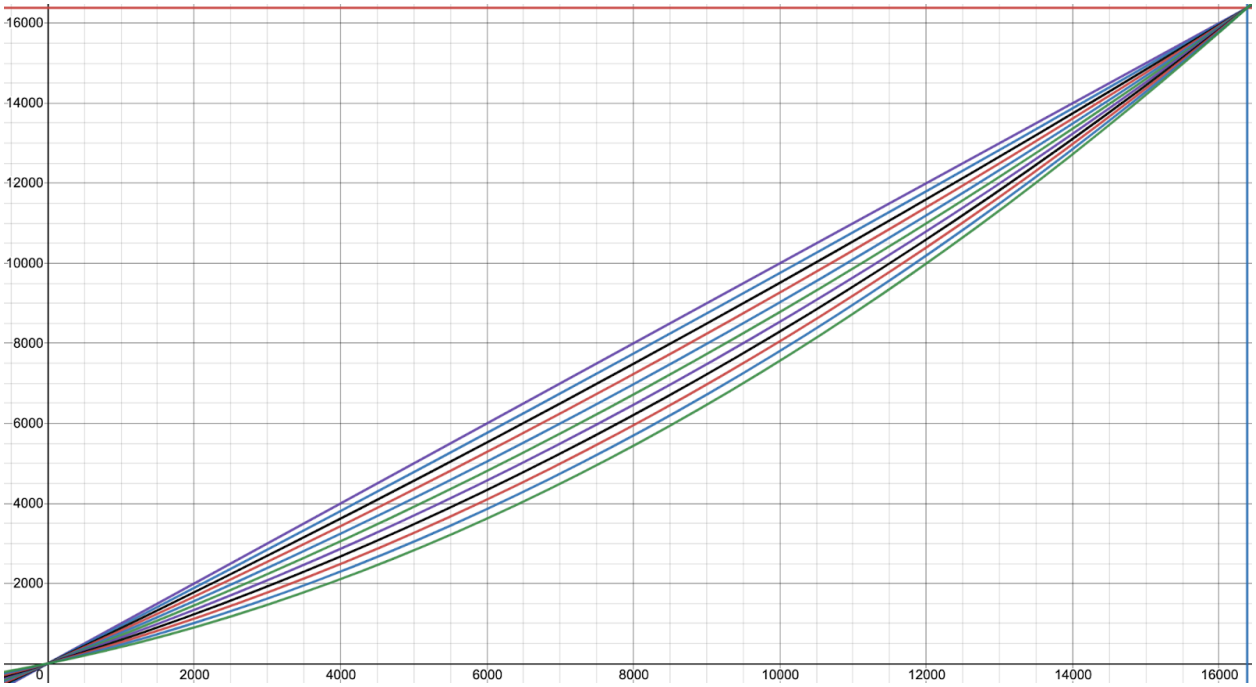

Figure 9: Quadratic scaling curves

Part 3: Real-Time Minimizer

When the quadratic scaling is done, the value is then passed into the minimizer, which checks if the passed in value is less than the stored (previous) minimum convolved density value. If it is, then we update our stored minimum convolved density value, time, and altitude. This is maintained until a new minimum value is found, if it is found.

Display

When the minimizer is done, we are left with an optimal (time, altitude) coordinate. The below modules will start at this coordinate, but the user can use the buttons and switches to change where they are. The up and down buttons will increase and decrease the altitude considered, and the left and right buttons will move the time forwards and backwards. The centre button will reset to the optimal coordinate. Furthermore, we implemented animation, controlled by two switches; sw[2] and sw[3] control animation through altitude and time, respectively, and will cycle through the user-submitted range, updating the color map, dot plot, and values displayed twice a second.

Our display uses the control signals of a 1024 by 768 XVGA display module. This required the use of a clock wizard IP to convert our built-in 100 MHz clock to a 65 MHz clock.

Part 1: Color Map (Pranav)

We use our current (time, altitude) coordinate to get an address into the density BRAM, then convert the data from the BRAM to an RGB value along the gradient shown in Figure 10, such that a value of 0 maps to pure green, with red increasing linearly until 255, which maps to yellow (maximum of red and green), and then green decreases linearly until 767, which maps to pure red, and values greater than 767 stay at pure red.

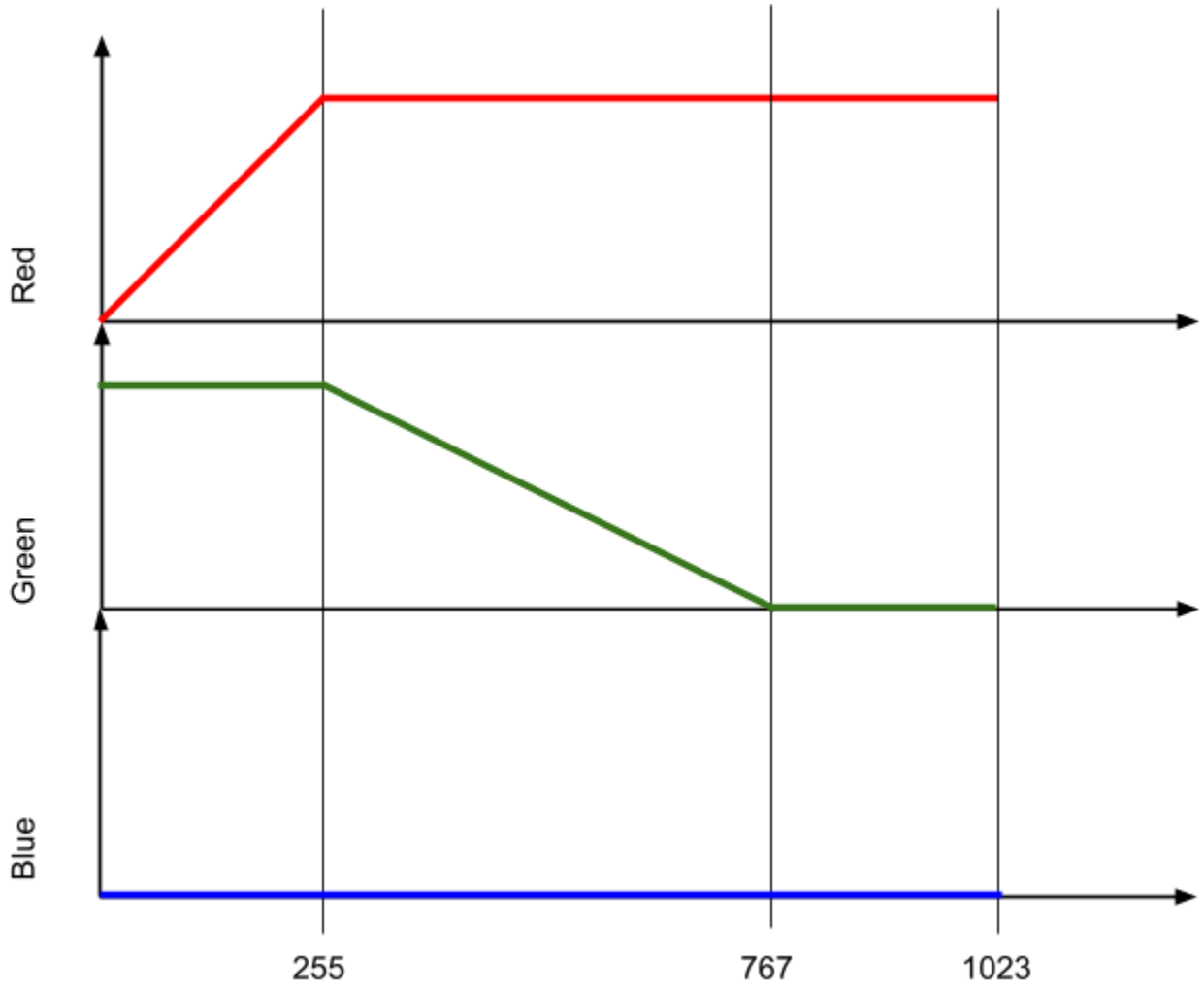

Figure 10: RGB Color Gradient

Part 2: Dot Plot (Pranav)

We also use the (time, altitude) coordinate to iterate through the satellite BRAM and determine which satellites are in range. If a satellite is in range, we convert the latitude and longitude values to a pixel value, assuming a 512 by 384 display, and write a 1 to the dot BRAM at that address. Then, when the XVGA module requests the pixel value for a certain (vcount, hcount) pixel location, we first check if the reduced pixel value, calculated as shown below, has a 1 in the dot BRAM at that address, and choose blue as our color output if so.

```
reduced_pixel = floor(vcount_768/2)*512 + floor(hcount_1024/2)
```

Part 3: Optimized Display Message (Sophia)

When the launch window optimization is complete, we display the optimal time and altitude in a message above the dot plot and heat map. This display message is composed of 6 instances of the digit ROM (24x24 pixels for each digit) and 1 instance of the message string (128x53 pixels) (these instances are shown in Figure 11 where every red underline and arrow represents an instance of the ROM). As described in the Landing Page module section, for the digits ROM, when we read from the ROM, we scale down the address using one bit shift right and then add an offset.

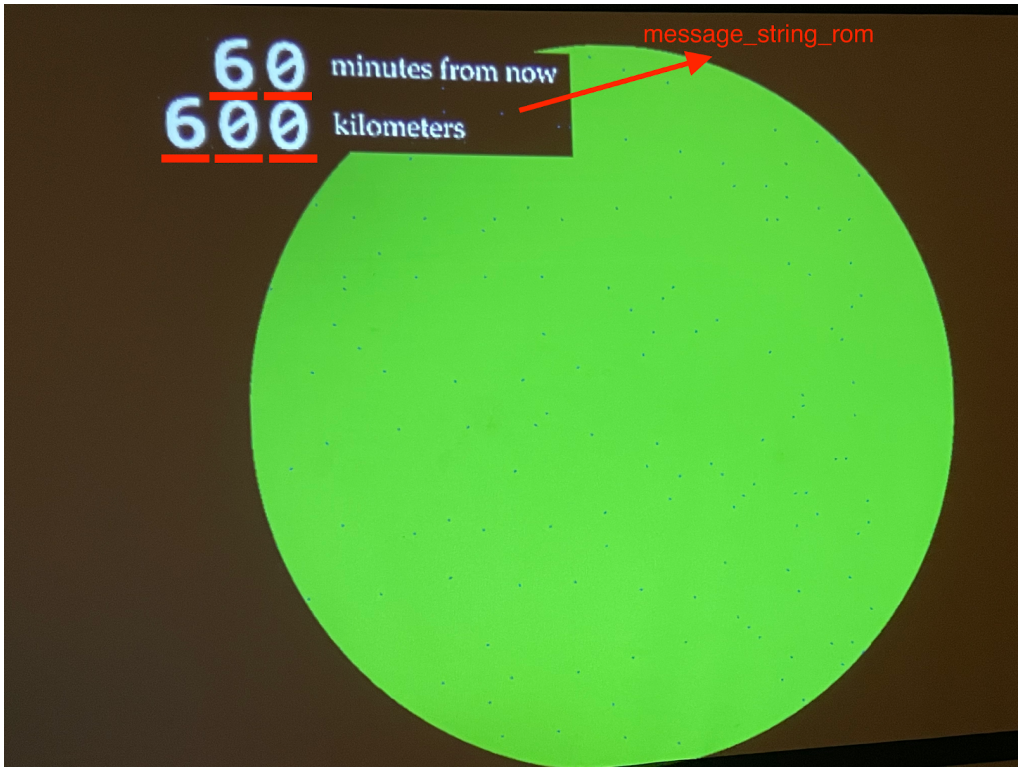

Figure 11: ROM instances used in optimized message

Challenges

Memory Limitations

Meeting the memory limitations of the FPGA was a major challenge for this project since we needed display ROMs for the landing screen and the optimized result screen. Originally, we began by having color images, which required full red, green, and blue color maps. However, we were met with several implementation errors which concerned the total memory on the

board. We approached this memory challenge by reducing the size of our images by a factor of two in each dimension. To reduce the blurriness of this image, we then applied a 5x5 pixel convolution filter to smooth the edges -- this was especially relevant when we had messages (letters) and digits. We reverted each image to black and white to eliminate the need for three additional .coe files (no color maps needed). We also saved BRAM usage by reducing the resolution of our dot plot from the full 1024 by 768 to a reduced 512 by 384.

The following figure, Figure 12, shows how we allocated memory by the conclusion of our project:

Memory	Size (blocks)	Quantity
Digits ROM	1	18
Arrow ROM	1	1
Landing Screen	48	1
Message String	2	1
Density RAM	1	1
Satellite RAM	28	1
Dot RAM	6	1

Total Memory used by image ROMs and data storage BRAMs: 104 blocks

Figure 12: Memory Allocation

SGP4 Computation

As discussed in the Database module section above, our satellite propagation was the only non-digital component of the system, where we used Python to implement this forward propagation in time. We faced significant challenges with memory when we were rendering our display as well as when we were storing data in different stages of our optimization process (e.g. density, dot locations, etc.). The SGP4 algorithm used for payload propagation is composed of five mathematical models, each of which is highly complex and overall considers periodic variations in location due to solar and lunar gravitational effects, resonance, orbital decay using a drag model, etc. When we used a SGP4 package in Python (optimized to perform this propagation as efficiently as possible), the script took ~15 minutes to complete propagating forward 4,000+ satellites in seven time increments (from 0 to 60 minutes in 10 minute increments). This performance signaled to our group that computing SGP4 on the FPGA would most likely not be possible without significant IP catalog usage, which would then reduce the memory requirements from other modules in our project. Additionally, the complexity of the algorithm would be difficult to accurately reach.

Another consideration within our satellite propagation process was the distance each satellite drifted in time. SGP4 drifts between 1-3 km per day it is propagated forward in. Noting this, we were careful to load in our data at maximum one day ahead of time so as to not significantly affect our optimization results.

Conclusion

Working on this project as a team was a challenging but rewarding exercise which helped the both of us learn more about digital design. As aerospace engineering students, much of our time is spent considering the applications of current technology in the aeronautics and space field. Optimization within the context of LEO launch windows is a task often performed through software. Being able to apply the skills we learned to ultimately create a working prototype of a FPGA-based optimization scheme was a rewarding and satisfying experience. A major lesson that we learned as a group was making allowances for unforeseen circumstances. Throughout the last month of the semester, we were working on separate modules which ultimately needed to be integrated into a system. Although we had each tested our individual modules, we had banked on integration within the last two weeks of the project work time. However, due to sickness, one week was cut off from that time, and major debugging efforts were made in the last week of the semester. In the future, we note that it is important to perform integration earlier, if not at the start of the project. Even if this means piecing together only two or three modules together at a time, this practice can help reduce the stress of circumstances out of our control which may prevent original plans from falling through.

We were able to meet and exceed the goals we had set out to achieve for this project, but if we were to continue working on this project, we would like to make a few extensions. Firstly, given more computational resources, an implementation of the SGP-4 algorithm would be an interesting task. Doing this would also allow us to conduct multiple propagations in parallel, which would allow us to consider time ranges greater than within 60 minutes from now; that is, we would be able to consider a wider range of user inputs. On the note of extending user inputs, we would also like to allow the user to select the launch location instead of leaving that fixed. Finally, given more storage capacity, we would like to improve the resolution of our displays, which will allow for a better user experience.

```
`timescale 1ns / 1ps
`default_nettype none

module top_level(
                 input wire          clk_100mhz,
                 input wire [15:0]   sw,
                 input wire          btnl,
                 input wire          btnr,
                 input wire          btnc,
                 input wire          btnu,
                 input wire          btnd,

                 output logic [3:0]  vga_r,
                 output logic [3:0]  vga_g,
                 output logic [3:0]  vga_b,
                 output logic        vga_hs,
                 output logic        vga_vs
                );

    // system reset
    logic reset;
    assign reset = sw[15];

    // debounce button presses
    logic up;
    debounce up_debounce(.clk_in(clk_100mhz),
                         .rst_in(reset),
                         .bouncey_in(btnu),
                         .clean_out(up));
    logic down;
    debounce down_debounce(.clk_in(clk_100mhz),
                           .rst_in(reset),
                           .bouncey_in(btnd),
                           .clean_out(down));
    logic right;
    debounce right_debounce(.clk_in(clk_100mhz),
                            .rst_in(reset),
                            .bouncey_in(btnr),
                            .clean_out(right));
    logic left;
    debounce left_debounce(.clk_in(clk_100mhz),
                           .rst_in(reset),
                           .bouncey_in(btnl),
                           .clean_out(left));
    logic centre;
    debounce centre_debounce(.clk_in(clk_100mhz),
                             .rst_in(reset),
                             .bouncey_in(btnc),
                             .clean_out(centre));

    // satellite location data BRAM
    logic sat_wea;
    logic [14:0] sat_addra;
    logic [32:0] sat_dina;
    logic [32:0] sat_douta;
    blk_mem_gen_0 satellite_ram(.clka(clk_100mhz),
                                .ena(1),
                                .wea(sat_wea),
                                .addra(sat_addra),
```

```
                            .dina(sat_dina),
                            .douta(sat_douta));

    // computed density BRAM
    logic d_wea;
    logic [6:0] d_addra;
    logic [9:0] d_dina;
    logic [9:0] d_douta;
    density_mem_gen density_ram(.clka(clk_100mhz),
                            .ena(1),
                            .wea(d_wea),
                            .addra(d_addra),
                            .dina(d_dina),
                            .douta(d_douta));

    // dot location data BRAM
    logic dot_wea;
    logic [17:0] dot_addra;
    logic dot_dina;
    logic dot_douta;
    dot_plot_ram dot_ram(.clka(clk_100mhz),
                        .ena(1),
                        .wea(dot_wea),
                        .addra(dot_addra),
                        .dina(dot_dina),
                        .douta(dot_douta));

    //Initializing Satellite RAM
    logic ram_start;
    logic [14:0] ram_addra;
    logic [32:0] ram_dina;
    logic ram_wea;
    logic ram_done;
    logic ram_ongoing;
    ram_initialize initialize_satellite_ram (.clk_in(clk_100mhz),
                                             .start(ram_start),
                                             .addr_in(ram_addra),
                                             .data_in(ram_dina),
                                             .wea(ram_wea),
                                             .done(ram_done),
                                             .ongoing(ram_ongoing));
    logic [14:0] max_sat_address;
    logic reading_d;
    logic done_reading_d;

    logic density_ready_in;
    logic [10:0] start_alt;
    logic [32:0] density_data_in;
    logic density_all_data_in;
    logic density_data_out;
    logic [6:0] density_bram_addr;
    logic [9:0] density_bram_data;
    logic density_end_pulse;
    density_calculator project_density (.clk_in(clk_100mhz),
                                        .start(ram_done),
                                        .ready_in(density_ready_in),
                                        .start_alt(start_alt),
                                        .data_in(density_data_in),
                                        .all_data_in(density_all_data_in),
                                        .data_out(density_data_out),
```

```
                                    .bram_addr(density_bram_addr),
                                    .bram_data(density_bram_data),
                                    .end_pulse(density_end_pulse));
logic reached_fir;

logic fir_ready;
logic [9:0] fir_data_in;
logic [13:0] fir_data_out;
logic fir_done;
fir_2d conv_filter (.clk_in(clk_100mhz),
                    .rst_in(density_end_pulse),
                    .ready_in(fir_ready),
                    .x_in(fir_data_in),
                    .y_out(fir_data_out),
                    .done_out(fir_done));
logic done_all_fir;

logic start_quad;
logic [3:0] scaler_alt_in;
logic [2:0] scaler_time_in;
logic [13:0] scaler_in;
logic [13:0] scaler_density_out;
logic quad_done;
logic [3:0] minimizer_alt_in;
logic [2:0] minimizer_time_in;
quad_scaler scaler (.clk_in(clk_100mhz),
                    .start_in(start_quad),
                    .alt_in(scaler_alt_in),
                    .time_in(scaler_time_in),
                    .val_in(scaler_in),
                    .weighting_in(min_alt_out),
                    .done_out(quad_done),
                    .alt_out(minimizer_alt_in),
                    .time_out(minimizer_time_in),
                    .val_out(scaler_density_out));

logic [3:0] min_alt_in;
logic [2:0] min_time_in;
logic [3:0] min_alt_out;
logic [2:0] min_time_out;
logic [13:0] min_density_out;
minimizer rt_min_finder (.clk_in(clk_100mhz),
                         .rst_in(density_end_pulse),
                         .valid_in(quad_done),
                         .cur_alt(minimizer_alt_in),
                         .cur_time(minimizer_time_in),
                         .cur_density(scaler_density_out),
                         .min_alt(min_alt_out),
                         .min_time(min_time_out),
                         .min_density(min_density_out));

logic dot_start;
logic [32:0] dot_data_in;
logic dot_all_data_in;
logic [3:0] sel_alt_in;
logic [2:0] sel_time_in;
logic dot_ongoing;
logic dot_data_out;
logic [17:0] dot_bram_addr;
logic dot_bram_data;
```

```
    logic dot_end_pulse;
    dot_determiner rt_dot_writer(.clk_in(clk_100mhz),
                                 .start(dot_start),
                                 .data_in(dot_data_in),
                                 .all_data_in(dot_all_data_in),
                                 .start_alt(start_alt),
                                 .sel_alt_in(sel_alt_in),
                                 .sel_time_in(sel_time_in),
                                 .ongoing_out(dot_ongoing),
                                 .data_out(dot_data_out),
                                 .bram_addr(dot_bram_addr),
                                 .bram_data(dot_bram_data),
                                 .end_pulse(dot_end_pulse));

    logic [9:0] color_density_in;
    logic [11:0] converted_rgb;
    gradient color_converter (.clk_in(clk_100mhz),
                              .density_in(color_density_in),
                              .rgb_out(converted_rgb));

    logic started_in;
    logic [11:0] converted_disp_rgb;
    logic xvga_bram_data_in;
    logic [17:0] xvga_bram_addr_out;
    logic [5:0] time_start_counter;
    logic [5:0] time_end_counter;
    logic [10:0] alt_start_counter;
    logic [10:0] alt_end_counter;
    xvga_display img_display(.clk_in(clk_100mhz),
                             .started_in(started_in),
                             .rgb_in(converted_disp_rgb),
                             .bram_data_in(xvga_bram_data_in),
                             .reset(reset),
                             .up(up),
                             .down(down),
                             .left(left),
                             .right(right),
                             .start_alt(start_alt),
                             .sel_alt_in(sel_alt_in),
                             .sel_time_in(sel_time_in),
                             .bram_addr_out(xvga_bram_addr_out),
                             .vga_r(vga_r),
                             .vga_g(vga_g),
                             .vga_b(vga_b),
                             .vga_hs(vga_hs),
                             .vga_vs(vga_vs),
                             .time_start_counter(time_start_counter),
                             .time_end_counter(time_end_counter),
                             .alt_start_counter(alt_start_counter),
                             .alt_end_counter(alt_end_counter));
    logic [3:0] max_alt_increment;
    logic [2:0] max_time_increment;
    logic [2:0] min_time_increment;

    // registers for when data takes time to move
    logic [1:0] await_d;
    logic [1:0] await;
    logic [3:0] setup;
    logic sent;
```

```
// previous state registers
logic old_sw_0;
logic prev_ram_ongoing;
logic prev_done_reading_d;
logic prev_done_all_fir;
logic prev_dot_all_data_in;
logic prev_up;
logic prev_down;
logic prev_right;
logic prev_left;
logic prev_centre;

// counter for animation
logic [27:0] two_hz_counter;

// only change display when fir is done and
// one of the inputs for changing display is valid!
logic change_display;
assign change_display = (
                         done_all_fir &&
                         (
                          ~prev_done_all_fir ||
                          (centre && ~prev_centre) ||
                          (
                           up &&
                           ~prev_up &&
                           (sel_alt_in < max_alt_increment)
                          ) ||
                          (
                           down &&
                           ~prev_down &&
                           (sel_alt_in > 0)
                          ) ||
                          (
                           right &&
                           ~prev_right &&
                           (sel_time_in < max_time_increment)
                          ) ||
                          (
                           left &&
                           ~prev_left &&
                           (sel_time_in > min_time_increment)
                          ) ||
                          (
                           (two_hz_counter == 0) &&
                           (sw[2] || sw[3])
                          )
                         )
                        );

// combinationally computed parameters based on user input
logic [2:0] comb_min_time_increment, comb_max_time_increment;
logic [3:0] comb_max_alt_increment;
logic [10:0] diff;
assign diff = alt_end_counter - alt_start_counter;
always_comb begin
    //time_start/10
    case(time_start_counter)
        6'd0 : comb_min_time_increment = 3'd0;
        6'd10 : comb_min_time_increment = 3'd1;
```

```
            6'd20 : comb_min_time_increment = 3'd2;
            6'd30 : comb_min_time_increment = 3'd3;
            6'd40 : comb_min_time_increment = 3'd4;
            6'd50 : comb_min_time_increment = 3'd5;
            6'd60 : comb_min_time_increment = 3'd6;
            default : comb_min_time_increment = 3'd0;
        endcase

        //time_end/10
        case(time_end_counter)
            6'd0 : comb_max_time_increment = 3'd0;
            6'd10 : comb_max_time_increment = 3'd1;
            6'd20 : comb_max_time_increment = 3'd2;
            6'd30 : comb_max_time_increment = 3'd3;
            6'd40 : comb_max_time_increment = 3'd4;
            6'd50 : comb_max_time_increment = 3'd5;
            6'd60 : comb_max_time_increment = 3'd6;
            default : comb_max_time_increment = 3'd0;
        endcase

        //(alt_end - alt_start)/50
        case(diff)
            11'd100 : comb_max_alt_increment = 4'd2;
            11'd150 : comb_max_alt_increment = 4'd3;
            11'd200 : comb_max_alt_increment = 4'd4;
            11'd250 : comb_max_alt_increment = 4'd5;
            11'd300 : comb_max_alt_increment = 4'd6;
            11'd350 : comb_max_alt_increment = 4'd7;
            11'd400 : comb_max_alt_increment = 4'd8;
            11'd450 : comb_max_alt_increment = 4'd9;
            11'd500 : comb_max_alt_increment = 4'd10;
            default : comb_max_alt_increment = 4'd0;
        endcase
    end

    always_ff @(posedge clk_100mhz) begin
        if (!sw[0]) begin
            // wait for this switch to start optimizing
            started_in <= 1'b0;
        end else if (~old_sw_0) begin
            // if the switch was just flipped,
            // check for valid values from start screen
            if (
                (time_end_counter >= time_start_counter + 20) &&
                (alt_end_counter >= alt_start_counter + 100) &&
                (alt_end_counter <= alt_start_counter + 500)
               ) begin
                // reset system
                ram_start <= 1;
                done_all_fir <= 0;
                //min_valid <= 0;
                start_quad <= 0;
                reached_fir <= 0;
                start_alt <= alt_start_counter;
                max_alt_increment <= comb_max_alt_increment;
                max_time_increment <= comb_max_time_increment;
                min_time_increment <= comb_min_time_increment;
                started_in <= 1'b1;
                two_hz_counter <= 0;
            end
```

```
        end else if (started_in) begin
            // each of these conditionals are such that they are true sequentially,
            // and never simultaneously
            if (ram_ongoing) begin
                // while reading hard-coded data into BRAM
                ram_start <= 0;
                // pass values from initializer module to satellite BRAM
                sat_wea <= ram_wea;
                sat_addra <= ram_addra;
                sat_dina <= ram_dina;
                done_reading_d <= 1'b0;
                await_d <= 2'd2;
            end else if (prev_ram_ongoing) begin
                // start reading data from satellite BRAM into density calc module
                sat_wea <= 0;
                max_sat_address <= sat_addra;
                sat_addra <= 15'd0;
                density_all_data_in <= 1'b0;
                await_d <= 2'd2;
            end else if (~done_reading_d) begin
                if (await_d >= 2'd1) begin
                    density_ready_in <= 0;
                    await_d <= await_d - 1'b1;
                end else begin
                    density_data_in <= sat_douta;
                    density_ready_in <= 1;
                    if (sat_addra < max_sat_address) begin
                        // increment satellite address
                        sat_addra <= sat_addra + 1'b1;
                    end else begin
                        //assert done sending data to density calc
                        density_all_data_in <= 1'b1;
                        done_reading_d <= 1'b1;
                    end
                    await_d <= 2'd2;
                end
            end else if (~prev_done_reading_d) begin
                density_all_data_in <= 1'b0; //one-clock cycle assert
            end else if (density_data_out) begin
                // when the density module is outputting data,
                // write it to the density BRAM
                d_wea <= 1;
                d_addra <= density_bram_addr;
                d_dina <= density_bram_data;
            end else begin
                // stop writing to the density BRAM
                d_wea <= 0;
                if (!done_all_fir) begin
                    // still conducting FIR
                    if (!reached_fir) begin
                        // start FIR
                        reached_fir <= 1;
                        d_addra <= min_time_increment;
                        sent <= 0;
                        await <= 2;
                        setup <= 8;
                        min_alt_in <= 1;
                        min_time_in <= min_time_increment+1;
                        // min_valid <= 0;
                        start_quad <= 0;
```

```
                // min_density_in <= 14'b11_1111_1111_1111;
            end else if (fir_done) begin
                // a FIR computation is done
                if (setup > 0) begin
                    // we have to wait for the 9 data points
                    // to be in the FIR
                    if (setup == 6 || setup == 3) begin
                        d_addra <= d_addra + 5;
                    end else begin
                        d_addra <= d_addra + 1;
                    end
                    setup <= setup - 1;
                end else begin
                    // pass data to quadratic scaler module
                    scaler_in <= fir_data_out;
                    scaler_alt_in <= min_alt_in;
                    scaler_time_in <= min_time_in;
                    start_quad <= 1;
                    // move to next BRAM address
                    if (min_alt_in < max_alt_increment - 1) begin
                        // move to next altitude
                        d_addra <= d_addra + 5;
                        setup <= 2;
                        min_alt_in <= min_alt_in + 1;
                    end else begin
                        // reached end of altitude
                        if (min_time_in < max_time_increment - 1) begin
                            // move to next time
                            d_addra <= min_time_in;
                            min_time_in <= min_time_in + 1;
                            setup <= 8;
                            min_alt_in <= 1;
                        end else begin
                            // reached end of time
                            // done with FIR
                            done_all_fir <= 1;
                        end
                    end
                end
                sent <= 0;
                await <= 2;
            end else begin
                start_quad <= 0;
                // min_valid <= 0;
                if (await > 0) begin
                    await <= await - 1;
                end else begin
                    // ensure data is only sent to FIR module once
                    if (!sent) begin
                        sent <= 1;
                        fir_ready <= 1;
                    end else begin
                        fir_ready <= 0;
                    end
                    fir_data_in <= d_douta;
                end
            end
        end else begin
            // ensure that quadratic scaler is not still being updated
            start_quad <= 0;
```

```
            if (change_display) begin
                // change the time/alt/BRAM address as desired
                if (!prev_done_all_fir || centre) begin
                    // back to optimum/minimum
                    sel_alt_in <= min_alt_out;
                    sel_time_in <= min_time_out;
                    d_addra <= min_alt_out*7 + min_time_out;
                end else if (up) begin
                    // increase altitude
                    sel_alt_in <= sel_alt_in + 1;
                    d_addra <= d_addra + 7;
                end else if (down) begin
                    // decrease altitude
                    sel_alt_in <= sel_alt_in - 1;
                    d_addra <= d_addra - 7;
                end else if (right) begin
                    // forward in time
                    sel_time_in <= sel_time_in + 1;
                    d_addra <= d_addra + 1;
                end else if (left) begin
                    // backward in time
                    sel_time_in <= sel_time_in - 1;
                    d_addra <= d_addra - 1;
                end else if (two_hz_counter == 0) begin
                    // animation
                    if (sw[2]) begin
                        // increase and cycle altitude
                        if (sel_alt_in == max_alt_increment) begin
                            sel_alt_in <= 0;
                            d_addra <= sel_time_in;
                        end else begin
                            sel_alt_in <= sel_alt_in + 1;
                            d_addra <= d_addra + 7;
                        end
                    end else if (sw[3]) begin
                        // increase and cycle time
                        if (sel_time_in == max_time_increment) begin
                            sel_time_in <= min_time_increment;
                            d_addra <= sel_alt_in*7 + min_time_increment;
                        end else begin
                            sel_time_in <= sel_time_in + 1;
                            d_addra <= d_addra + 1;
                        end
                    end
                end
                // get ready to start sending data into the
                // dot determiner module and dot BRAM
                dot_start <= 1;
                sat_addra <= 0;
                await <= 2;
                dot_all_data_in <= 0;
                dot_wea <= 1;
            end else begin
                if (await > 0) begin
                    await <= await - 1;
                    if (await == 1) begin
                        // single cycle assert
                        dot_start <= 0;
                    end
                end else begin
```

```
                        // pass density data to color converter
                        // and satellite data to dot determiner modules
                        color_density_in <= d_douta;
                        dot_data_in <= sat_douta;
                    end
                    if (dot_ongoing && (sat_addra < max_sat_address)) begin
                        // increment satellite address
                        sat_addra <= sat_addra + 1;
                    end else if (dot_all_data_in && !prev_dot_all_data_in) begin
                        // ensure single cycle assert and
                        // that no more data is written to dot BRAM
                        dot_wea <= 0;
                        dot_all_data_in <= 0;
                    end else if (dot_wea && sat_addra == max_sat_address) begin
                        // all data has been passed in
                        dot_all_data_in <= 1;
                    end
                    if (dot_wea) begin
                        // write to the dot BRAM with the address/data
                        // from the dot determiner module
                        dot_addra <= dot_bram_addr;
                        dot_dina <= dot_bram_data;
                    end else begin
                        // read from the dot BRAM with the address
                        // from (and send the data to) the xvga module
                        dot_addra <= xvga_bram_addr_out;
                        xvga_bram_data_in <= dot_douta;
                    end
                end
                // send the data from the color gradient module to the xvga module
                converted_disp_rgb <= converted_rgb;
            end
        end
        // increment until it reaches 50_000_000
        // which should happen twice a second
        if (two_hz_counter < 28'd50_000_000) begin
            two_hz_counter <= two_hz_counter + 1;
        end else begin
            two_hz_counter <= 0;
        end
    end

    // update previous value registers
    old_sw_0 <= sw[0];
    prev_ram_ongoing <= ram_ongoing;
    prev_done_reading_d <= done_reading_d;
    prev_done_all_fir <= done_all_fir;
    prev_dot_all_data_in <= dot_all_data_in;
    prev_up <= up;
    prev_down <= down;
    prev_right <= right;
    prev_left <= left;
    prev_centre <= centre;
  end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none
```

```
module debounce(
                input  wire  clk_in,
                input  wire  rst_in,
                input  wire  bouncey_in,

                output logic clean_out
               );

    // counter to determine if input has been held
    logic [19:0] count;
    // store previous input
    logic old;

    always_ff @(posedge clk_in) begin
        if (rst_in) begin
            // reset counter
            count <= 20'b0;
        end else begin
            if (bouncey_in == old) begin
                if (count >= 20'd999999)begin
                    // clean signal, update output
                    clean_out <= bouncey_in;
                end
                // increment count
                count <= count + 1;
            end else begin
                // different signal, reset counter
                count <= 20'b0;
            end
            // update last signal
            old <= bouncey_in;
        end
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module density_calculator(
                          input  wire         clk_in,
                          input  wire         start,
                          input  wire         ready_in,
                          input  wire  [10:0] start_alt,
                          input  wire  [32:0] data_in,
                          input  wire         all_data_in,

                          output logic        data_out,
                          output logic [6:0]  bram_addr,
                          output logic [9:0]  bram_data,
                          output logic        end_pulse
                         );

    parameter HEIGHT_INCREMENT = 50; // 50 km
    parameter LAUNCH_LAT = 120; // 30 degrees North
    parameter LAUNCH_LON = 85; // 95 deg West
    parameter RADIUS = 32; // 32 deg
```

```
    // state indicators
    logic ongoing, returning;
    logic [2:0] time_counter;
    logic [3:0] alt;

    // arrays to temporarily hold data to send to BRAM
    logic [9:0] time_0_ram [10:0];
    logic [9:0] time_1_ram [10:0];
    logic [9:0] time_2_ram [10:0];
    logic [9:0] time_3_ram [10:0];
    logic [9:0] time_4_ram [10:0];
    logic [9:0] time_5_ram [10:0];
    logic [9:0] time_6_ram [10:0];

    // combinationally compute next address
    logic [6:0] next_addr;

    always_comb begin
        if (!data_out) begin
            next_addr = 0;
        end else begin
            next_addr = bram_addr + 1;
        end
        case (time_counter)
            // send out data based on the time and altitude counters
            0: bram_data = time_0_ram[alt];
            1: bram_data = time_1_ram[alt];
            2: bram_data = time_2_ram[alt];
            3: bram_data = time_3_ram[alt];
            4: bram_data = time_4_ram[alt];
            5: bram_data = time_5_ram[alt];
            6: bram_data = time_6_ram[alt];
            default: bram_data = 0;
        endcase;
    end

    always_ff @(posedge clk_in) begin
        if (start) begin
            // reset module
            ongoing <= 1'b1;
            returning <= 1'b0;
            time_counter <= 0;
            alt <= 4'b0;
            for (int j = 0; j < 11; j++) begin
                time_0_ram[j] <= 9'd0;
                time_1_ram[j] <= 9'd0;
                time_2_ram[j] <= 9'd0;
                time_3_ram[j] <= 9'd0;
                time_4_ram[j] <= 9'd0;
                time_5_ram[j] <= 9'd0;
                time_6_ram[j] <= 9'd0;
            end
            data_out <= 1'b0;
            bram_addr <= 0;
            end_pulse <= 1'b0;
        end else if (returning) begin
            // send out data from the arrays at the next address
            bram_addr <= next_addr;
            data_out <= 1'b1;
            // step through the time counter each cycle
```

```
            if (time_counter == 3'd6) begin
                time_counter <= 3'b0;
                // step through the altitude counter
                // when we hit the end of the time counter
                if (alt < 4'd10) begin
                    alt <= alt + 1;
                end else begin
                    // stop returning
                    end_pulse <= 1'b1;
                    returning <= 1'b0;
                end
            end else begin
                time_counter <= time_counter + 1;
            end
        end else if (end_pulse) begin
            // end pulse is single cycle
            end_pulse <= 1'b0;
            data_out <= 1'b0;
        end else if (ongoing) begin
            // step through the time counter each cycle
            if (all_data_in || time_counter == 3'd6) begin
                time_counter <= 3'b0;
            end else begin
                time_counter <= time_counter + 1;
            end
            if (all_data_in) begin
                // move to returning
                ongoing <= 1'b0;
                returning <= 1'b1;
                alt <= 0;
            end
            // check if the satellite latitude/longitude is near the centre
            if (ready_in &&
                    (((((data_in[32:25])-LAUNCH_LAT)**2) +
                      (((data_in[24:16])-LAUNCH_LON)**2)) <= (RADIUS**2))) begin
                // determine which bucket of altitude the satellite is in,
                // and update the arrays accordingly
                if (data_in[15:0] <= start_alt) begin
                    if (time_counter == 3'd0) begin
                        if (time_0_ram[0] < 10'd1023) begin
                            time_0_ram[0] <= time_0_ram[0] + 1'b1;
                        end
                        if (time_0_ram[1] < 10'd1023) begin
                            time_0_ram[1] <= time_0_ram[1] + 1'b1;
                        end
                        if (time_0_ram[2] < 10'd1023) begin
                            time_0_ram[2] <= time_0_ram[2] + 1'b1;
                        end
                        if (time_0_ram[3] < 10'd1023) begin
                            time_0_ram[3] <= time_0_ram[3] + 1'b1;
                        end
                        if (time_0_ram[4] < 10'd1023) begin
                            time_0_ram[4] <= time_0_ram[4] + 1'b1;
                        end
                        if (time_0_ram[5] < 10'd1023) begin
                            time_0_ram[5] <= time_0_ram[5] + 1'b1;
                        end
                        if (time_0_ram[6] < 10'd1023) begin
                            time_0_ram[6] <= time_0_ram[6] + 1'b1;
                        end
```

```
                if (time_0_ram[7] < 10'd1023) begin
                    time_0_ram[7] <= time_0_ram[7] + 1'b1;
                end
                if (time_0_ram[8] < 10'd1023) begin
                    time_0_ram[8] <= time_0_ram[8] + 1'b1;
                end
                if (time_0_ram[9] < 10'd1023) begin
                    time_0_ram[9] <= time_0_ram[9] + 1'b1;
                end
                if (time_0_ram[10] < 10'd1023) begin
                    time_0_ram[10] <= time_0_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd1) begin
                if (time_1_ram[0] < 10'd1023) begin
                    time_1_ram[0] <= time_1_ram[0] + 1'b1;
                end
                if (time_1_ram[1] < 10'd1023) begin
                    time_1_ram[1] <= time_1_ram[1] + 1'b1;
                end
                if (time_1_ram[2] < 10'd1023) begin
                    time_1_ram[2] <= time_1_ram[2] + 1'b1;
                end
                if (time_1_ram[3] < 10'd1023) begin
                    time_1_ram[3] <= time_1_ram[3] + 1'b1;
                end
                if (time_1_ram[4] < 10'd1023) begin
                    time_1_ram[4] <= time_1_ram[4] + 1'b1;
                end
                if (time_1_ram[5] < 10'd1023) begin
                    time_1_ram[5] <= time_1_ram[5] + 1'b1;
                end
                if (time_1_ram[6] < 10'd1023) begin
                    time_1_ram[6] <= time_1_ram[6] + 1'b1;
                end
                if (time_1_ram[7] < 10'd1023) begin
                    time_1_ram[7] <= time_1_ram[7] + 1'b1;
                end
                if (time_1_ram[8] < 10'd1023) begin
                    time_1_ram[8] <= time_1_ram[8] + 1'b1;
                end
                if (time_1_ram[9] < 10'd1023) begin
                    time_1_ram[9] <= time_1_ram[9] + 1'b1;
                end
                if (time_1_ram[10] < 10'd1023) begin
                    time_1_ram[10] <= time_1_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd2) begin
                if (time_2_ram[0] < 10'd1023) begin
                    time_2_ram[0] <= time_2_ram[0] + 1'b1;
                end
                if (time_2_ram[1] < 10'd1023) begin
                    time_2_ram[1] <= time_2_ram[1] + 1'b1;
                end
                if (time_2_ram[2] < 10'd1023) begin
                    time_2_ram[2] <= time_2_ram[2] + 1'b1;
                end
                if (time_2_ram[3] < 10'd1023) begin
                    time_2_ram[3] <= time_2_ram[3] + 1'b1;
                end
                if (time_2_ram[4] < 10'd1023) begin
```

```
                time_2_ram[4] <= time_2_ram[4] + 1'b1;
            end
            if (time_2_ram[5] < 10'd1023) begin
                time_2_ram[5] <= time_2_ram[5] + 1'b1;
            end
            if (time_2_ram[6] < 10'd1023) begin
                time_2_ram[6] <= time_2_ram[6] + 1'b1;
            end
            if (time_2_ram[7] < 10'd1023) begin
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[0] < 10'd1023) begin
                time_3_ram[0] <= time_3_ram[0] + 1'b1;
            end
            if (time_3_ram[1] < 10'd1023) begin
                time_3_ram[1] <= time_3_ram[1] + 1'b1;
            end
            if (time_3_ram[2] < 10'd1023) begin
                time_3_ram[2] <= time_3_ram[2] + 1'b1;
            end
            if (time_3_ram[3] < 10'd1023) begin
                time_3_ram[3] <= time_3_ram[3] + 1'b1;
            end
            if (time_3_ram[4] < 10'd1023) begin
                time_3_ram[4] <= time_3_ram[4] + 1'b1;
            end
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[0] < 10'd1023) begin
                time_4_ram[0] <= time_4_ram[0] + 1'b1;
            end
            if (time_4_ram[1] < 10'd1023) begin
                time_4_ram[1] <= time_4_ram[1] + 1'b1;
```

```
            end
            if (time_4_ram[2] < 10'd1023) begin
                time_4_ram[2] <= time_4_ram[2] + 1'b1;
            end
            if (time_4_ram[3] < 10'd1023) begin
                time_4_ram[3] <= time_4_ram[3] + 1'b1;
            end
            if (time_4_ram[4] < 10'd1023) begin
                time_4_ram[4] <= time_4_ram[4] + 1'b1;
            end
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[0] < 10'd1023) begin
                time_5_ram[0] <= time_5_ram[0] + 1'b1;
            end
            if (time_5_ram[1] < 10'd1023) begin
                time_5_ram[1] <= time_5_ram[1] + 1'b1;
            end
            if (time_5_ram[2] < 10'd1023) begin
                time_5_ram[2] <= time_5_ram[2] + 1'b1;
            end
            if (time_5_ram[3] < 10'd1023) begin
                time_5_ram[3] <= time_5_ram[3] + 1'b1;
            end
            if (time_5_ram[4] < 10'd1023) begin
                time_5_ram[4] <= time_5_ram[4] + 1'b1;
            end
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
```

```
                    time_5_ram[10] <= time_5_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd6) begin
                if (time_6_ram[0] < 10'd1023) begin
                    time_6_ram[0] <= time_6_ram[0] + 1'b1;
                end
                if (time_6_ram[1] < 10'd1023) begin
                    time_6_ram[1] <= time_6_ram[1] + 1'b1;
                end
                if (time_6_ram[2] < 10'd1023) begin
                    time_6_ram[2] <= time_6_ram[2] + 1'b1;
                end
                if (time_6_ram[3] < 10'd1023) begin
                    time_6_ram[3] <= time_6_ram[3] + 1'b1;
                end
                if (time_6_ram[4] < 10'd1023) begin
                    time_6_ram[4] <= time_6_ram[4] + 1'b1;
                end
                if (time_6_ram[5] < 10'd1023) begin
                    time_6_ram[5] <= time_6_ram[5] + 1'b1;
                end
                if (time_6_ram[6] < 10'd1023) begin
                    time_6_ram[6] <= time_6_ram[6] + 1'b1;
                end
                if (time_6_ram[7] < 10'd1023) begin
                    time_6_ram[7] <= time_6_ram[7] + 1'b1;
                end
                if (time_6_ram[8] < 10'd1023) begin
                    time_6_ram[8] <= time_6_ram[8] + 1'b1;
                end
                if (time_6_ram[9] < 10'd1023) begin
                    time_6_ram[9] <= time_6_ram[9] + 1'b1;
                end
                if (time_6_ram[10] < 10'd1023) begin
                    time_6_ram[10] <= time_6_ram[10] + 1'b1;
                end
            end
        end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*1) begin
            if (time_counter == 3'd0) begin
                if (time_0_ram[1] < 10'd1023) begin
                    time_0_ram[1] <= time_0_ram[1] + 1'b1;
                end
                if (time_0_ram[2] < 10'd1023) begin
                    time_0_ram[2] <= time_0_ram[2] + 1'b1;
                end
                if (time_0_ram[3] < 10'd1023) begin
                    time_0_ram[3] <= time_0_ram[3] + 1'b1;
                end
                if (time_0_ram[4] < 10'd1023) begin
                    time_0_ram[4] <= time_0_ram[4] + 1'b1;
                end
                if (time_0_ram[5] < 10'd1023) begin
                    time_0_ram[5] <= time_0_ram[5] + 1'b1;
                end
                if (time_0_ram[6] < 10'd1023) begin
                    time_0_ram[6] <= time_0_ram[6] + 1'b1;
                end
                if (time_0_ram[7] < 10'd1023) begin
                    time_0_ram[7] <= time_0_ram[7] + 1'b1;
                end
```

```
            if (time_0_ram[8] < 10'd1023) begin
                time_0_ram[8] <= time_0_ram[8] + 1'b1;
            end
            if (time_0_ram[9] < 10'd1023) begin
                time_0_ram[9] <= time_0_ram[9] + 1'b1;
            end
            if (time_0_ram[10] < 10'd1023) begin
                time_0_ram[10] <= time_0_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd1) begin
            if (time_1_ram[1] < 10'd1023) begin
                time_1_ram[1] <= time_1_ram[1] + 1'b1;
            end
            if (time_1_ram[2] < 10'd1023) begin
                time_1_ram[2] <= time_1_ram[2] + 1'b1;
            end
            if (time_1_ram[3] < 10'd1023) begin
                time_1_ram[3] <= time_1_ram[3] + 1'b1;
            end
            if (time_1_ram[4] < 10'd1023) begin
                time_1_ram[4] <= time_1_ram[4] + 1'b1;
            end
            if (time_1_ram[5] < 10'd1023) begin
                time_1_ram[5] <= time_1_ram[5] + 1'b1;
            end
            if (time_1_ram[6] < 10'd1023) begin
                time_1_ram[6] <= time_1_ram[6] + 1'b1;
            end
            if (time_1_ram[7] < 10'd1023) begin
                time_1_ram[7] <= time_1_ram[7] + 1'b1;
            end
            if (time_1_ram[8] < 10'd1023) begin
                time_1_ram[8] <= time_1_ram[8] + 1'b1;
            end
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[1] < 10'd1023) begin
                time_2_ram[1] <= time_2_ram[1] + 1'b1;
            end
            if (time_2_ram[2] < 10'd1023) begin
                time_2_ram[2] <= time_2_ram[2] + 1'b1;
            end
            if (time_2_ram[3] < 10'd1023) begin
                time_2_ram[3] <= time_2_ram[3] + 1'b1;
            end
            if (time_2_ram[4] < 10'd1023) begin
                time_2_ram[4] <= time_2_ram[4] + 1'b1;
            end
            if (time_2_ram[5] < 10'd1023) begin
                time_2_ram[5] <= time_2_ram[5] + 1'b1;
            end
            if (time_2_ram[6] < 10'd1023) begin
                time_2_ram[6] <= time_2_ram[6] + 1'b1;
            end
            if (time_2_ram[7] < 10'd1023) begin
```

```
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[1] < 10'd1023) begin
                time_3_ram[1] <= time_3_ram[1] + 1'b1;
            end
            if (time_3_ram[2] < 10'd1023) begin
                time_3_ram[2] <= time_3_ram[2] + 1'b1;
            end
            if (time_3_ram[3] < 10'd1023) begin
                time_3_ram[3] <= time_3_ram[3] + 1'b1;
            end
            if (time_3_ram[4] < 10'd1023) begin
                time_3_ram[4] <= time_3_ram[4] + 1'b1;
            end
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[1] < 10'd1023) begin
                time_4_ram[1] <= time_4_ram[1] + 1'b1;
            end
            if (time_4_ram[2] < 10'd1023) begin
                time_4_ram[2] <= time_4_ram[2] + 1'b1;
            end
            if (time_4_ram[3] < 10'd1023) begin
                time_4_ram[3] <= time_4_ram[3] + 1'b1;
            end
            if (time_4_ram[4] < 10'd1023) begin
                time_4_ram[4] <= time_4_ram[4] + 1'b1;
            end
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
```

```
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[1] < 10'd1023) begin
                time_5_ram[1] <= time_5_ram[1] + 1'b1;
            end
            if (time_5_ram[2] < 10'd1023) begin
                time_5_ram[2] <= time_5_ram[2] + 1'b1;
            end
            if (time_5_ram[3] < 10'd1023) begin
                time_5_ram[3] <= time_5_ram[3] + 1'b1;
            end
            if (time_5_ram[4] < 10'd1023) begin
                time_5_ram[4] <= time_5_ram[4] + 1'b1;
            end
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd6) begin
            if (time_6_ram[1] < 10'd1023) begin
                time_6_ram[1] <= time_6_ram[1] + 1'b1;
            end
            if (time_6_ram[2] < 10'd1023) begin
                time_6_ram[2] <= time_6_ram[2] + 1'b1;
            end
            if (time_6_ram[3] < 10'd1023) begin
                time_6_ram[3] <= time_6_ram[3] + 1'b1;
            end
            if (time_6_ram[4] < 10'd1023) begin
                time_6_ram[4] <= time_6_ram[4] + 1'b1;
            end
            if (time_6_ram[5] < 10'd1023) begin
                time_6_ram[5] <= time_6_ram[5] + 1'b1;
            end
```

```
                    if (time_6_ram[6] < 10'd1023) begin
                        time_6_ram[6] <= time_6_ram[6] + 1'b1;
                    end
                    if (time_6_ram[7] < 10'd1023) begin
                        time_6_ram[7] <= time_6_ram[7] + 1'b1;
                    end
                    if (time_6_ram[8] < 10'd1023) begin
                        time_6_ram[8] <= time_6_ram[8] + 1'b1;
                    end
                    if (time_6_ram[9] < 10'd1023) begin
                        time_6_ram[9] <= time_6_ram[9] + 1'b1;
                    end
                    if (time_6_ram[10] < 10'd1023) begin
                        time_6_ram[10] <= time_6_ram[10] + 1'b1;
                    end
                end
            end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*2) begin
                if (time_counter == 3'd0) begin
                    if (time_0_ram[2] < 10'd1023) begin
                        time_0_ram[2] <= time_0_ram[2] + 1'b1;
                    end
                    if (time_0_ram[3] < 10'd1023) begin
                        time_0_ram[3] <= time_0_ram[3] + 1'b1;
                    end
                    if (time_0_ram[4] < 10'd1023) begin
                        time_0_ram[4] <= time_0_ram[4] + 1'b1;
                    end
                    if (time_0_ram[5] < 10'd1023) begin
                        time_0_ram[5] <= time_0_ram[5] + 1'b1;
                    end
                    if (time_0_ram[6] < 10'd1023) begin
                        time_0_ram[6] <= time_0_ram[6] + 1'b1;
                    end
                    if (time_0_ram[7] < 10'd1023) begin
                        time_0_ram[7] <= time_0_ram[7] + 1'b1;
                    end
                    if (time_0_ram[8] < 10'd1023) begin
                        time_0_ram[8] <= time_0_ram[8] + 1'b1;
                    end
                    if (time_0_ram[9] < 10'd1023) begin
                        time_0_ram[9] <= time_0_ram[9] + 1'b1;
                    end
                    if (time_0_ram[10] < 10'd1023) begin
                        time_0_ram[10] <= time_0_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd1) begin
                    if (time_1_ram[2] < 10'd1023) begin
                        time_1_ram[2] <= time_1_ram[2] + 1'b1;
                    end
                    if (time_1_ram[3] < 10'd1023) begin
                        time_1_ram[3] <= time_1_ram[3] + 1'b1;
                    end
                    if (time_1_ram[4] < 10'd1023) begin
                        time_1_ram[4] <= time_1_ram[4] + 1'b1;
                    end
                    if (time_1_ram[5] < 10'd1023) begin
                        time_1_ram[5] <= time_1_ram[5] + 1'b1;
                    end
                    if (time_1_ram[6] < 10'd1023) begin
                        time_1_ram[6] <= time_1_ram[6] + 1'b1;
```

```
            end
            if (time_1_ram[7] < 10'd1023) begin
                time_1_ram[7] <= time_1_ram[7] + 1'b1;
            end
            if (time_1_ram[8] < 10'd1023) begin
                time_1_ram[8] <= time_1_ram[8] + 1'b1;
            end
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[2] < 10'd1023) begin
                time_2_ram[2] <= time_2_ram[2] + 1'b1;
            end
            if (time_2_ram[3] < 10'd1023) begin
                time_2_ram[3] <= time_2_ram[3] + 1'b1;
            end
            if (time_2_ram[4] < 10'd1023) begin
                time_2_ram[4] <= time_2_ram[4] + 1'b1;
            end
            if (time_2_ram[5] < 10'd1023) begin
                time_2_ram[5] <= time_2_ram[5] + 1'b1;
            end
            if (time_2_ram[6] < 10'd1023) begin
                time_2_ram[6] <= time_2_ram[6] + 1'b1;
            end
            if (time_2_ram[7] < 10'd1023) begin
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[2] < 10'd1023) begin
                time_3_ram[2] <= time_3_ram[2] + 1'b1;
            end
            if (time_3_ram[3] < 10'd1023) begin
                time_3_ram[3] <= time_3_ram[3] + 1'b1;
            end
            if (time_3_ram[4] < 10'd1023) begin
                time_3_ram[4] <= time_3_ram[4] + 1'b1;
            end
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
```

```
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[2] < 10'd1023) begin
                time_4_ram[2] <= time_4_ram[2] + 1'b1;
            end
            if (time_4_ram[3] < 10'd1023) begin
                time_4_ram[3] <= time_4_ram[3] + 1'b1;
            end
            if (time_4_ram[4] < 10'd1023) begin
                time_4_ram[4] <= time_4_ram[4] + 1'b1;
            end
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[2] < 10'd1023) begin
                time_5_ram[2] <= time_5_ram[2] + 1'b1;
            end
            if (time_5_ram[3] < 10'd1023) begin
                time_5_ram[3] <= time_5_ram[3] + 1'b1;
            end
            if (time_5_ram[4] < 10'd1023) begin
                time_5_ram[4] <= time_5_ram[4] + 1'b1;
            end
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
```

```
                    time_5_ram[9] <= time_5_ram[9] + 1'b1;
                end
                if (time_5_ram[10] < 10'd1023) begin
                    time_5_ram[10] <= time_5_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd6) begin
                if (time_6_ram[2] < 10'd1023) begin
                    time_6_ram[2] <= time_6_ram[2] + 1'b1;
                end
                if (time_6_ram[3] < 10'd1023) begin
                    time_6_ram[3] <= time_6_ram[3] + 1'b1;
                end
                if (time_6_ram[4] < 10'd1023) begin
                    time_6_ram[4] <= time_6_ram[4] + 1'b1;
                end
                if (time_6_ram[5] < 10'd1023) begin
                    time_6_ram[5] <= time_6_ram[5] + 1'b1;
                end
                if (time_6_ram[6] < 10'd1023) begin
                    time_6_ram[6] <= time_6_ram[6] + 1'b1;
                end
                if (time_6_ram[7] < 10'd1023) begin
                    time_6_ram[7] <= time_6_ram[7] + 1'b1;
                end
                if (time_6_ram[8] < 10'd1023) begin
                    time_6_ram[8] <= time_6_ram[8] + 1'b1;
                end
                if (time_6_ram[9] < 10'd1023) begin
                    time_6_ram[9] <= time_6_ram[9] + 1'b1;
                end
                if (time_6_ram[10] < 10'd1023) begin
                    time_6_ram[10] <= time_6_ram[10] + 1'b1;
                end
            end
        end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*3) begin
            if (time_counter == 3'd0) begin
                if (time_0_ram[3] < 10'd1023) begin
                    time_0_ram[3] <= time_0_ram[3] + 1'b1;
                end
                if (time_0_ram[4] < 10'd1023) begin
                    time_0_ram[4] <= time_0_ram[4] + 1'b1;
                end
                if (time_0_ram[5] < 10'd1023) begin
                    time_0_ram[5] <= time_0_ram[5] + 1'b1;
                end
                if (time_0_ram[6] < 10'd1023) begin
                    time_0_ram[6] <= time_0_ram[6] + 1'b1;
                end
                if (time_0_ram[7] < 10'd1023) begin
                    time_0_ram[7] <= time_0_ram[7] + 1'b1;
                end
                if (time_0_ram[8] < 10'd1023) begin
                    time_0_ram[8] <= time_0_ram[8] + 1'b1;
                end
                if (time_0_ram[9] < 10'd1023) begin
                    time_0_ram[9] <= time_0_ram[9] + 1'b1;
                end
                if (time_0_ram[10] < 10'd1023) begin
                    time_0_ram[10] <= time_0_ram[10] + 1'b1;
                end
```

```
        end else if (time_counter == 3'd1) begin
            if (time_1_ram[3] < 10'd1023) begin
                time_1_ram[3] <= time_1_ram[3] + 1'b1;
            end
            if (time_1_ram[4] < 10'd1023) begin
                time_1_ram[4] <= time_1_ram[4] + 1'b1;
            end
            if (time_1_ram[5] < 10'd1023) begin
                time_1_ram[5] <= time_1_ram[5] + 1'b1;
            end
            if (time_1_ram[6] < 10'd1023) begin
                time_1_ram[6] <= time_1_ram[6] + 1'b1;
            end
            if (time_1_ram[7] < 10'd1023) begin
                time_1_ram[7] <= time_1_ram[7] + 1'b1;
            end
            if (time_1_ram[8] < 10'd1023) begin
                time_1_ram[8] <= time_1_ram[8] + 1'b1;
            end
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[3] < 10'd1023) begin
                time_2_ram[3] <= time_2_ram[3] + 1'b1;
            end
            if (time_2_ram[4] < 10'd1023) begin
                time_2_ram[4] <= time_2_ram[4] + 1'b1;
            end
            if (time_2_ram[5] < 10'd1023) begin
                time_2_ram[5] <= time_2_ram[5] + 1'b1;
            end
            if (time_2_ram[6] < 10'd1023) begin
                time_2_ram[6] <= time_2_ram[6] + 1'b1;
            end
            if (time_2_ram[7] < 10'd1023) begin
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[3] < 10'd1023) begin
                time_3_ram[3] <= time_3_ram[3] + 1'b1;
            end
            if (time_3_ram[4] < 10'd1023) begin
                time_3_ram[4] <= time_3_ram[4] + 1'b1;
            end
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
```

```
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[3] < 10'd1023) begin
                time_4_ram[3] <= time_4_ram[3] + 1'b1;
            end
            if (time_4_ram[4] < 10'd1023) begin
                time_4_ram[4] <= time_4_ram[4] + 1'b1;
            end
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[3] < 10'd1023) begin
                time_5_ram[3] <= time_5_ram[3] + 1'b1;
            end
            if (time_5_ram[4] < 10'd1023) begin
                time_5_ram[4] <= time_5_ram[4] + 1'b1;
            end
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
```

```
                    time_5_ram[9] <= time_5_ram[9] + 1'b1;
                end
                if (time_5_ram[10] < 10'd1023) begin
                    time_5_ram[10] <= time_5_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd6) begin
                if (time_6_ram[3] < 10'd1023) begin
                    time_6_ram[3] <= time_6_ram[3] + 1'b1;
                end
                if (time_6_ram[4] < 10'd1023) begin
                    time_6_ram[4] <= time_6_ram[4] + 1'b1;
                end
                if (time_6_ram[5] < 10'd1023) begin
                    time_6_ram[5] <= time_6_ram[5] + 1'b1;
                end
                if (time_6_ram[6] < 10'd1023) begin
                    time_6_ram[6] <= time_6_ram[6] + 1'b1;
                end
                if (time_6_ram[7] < 10'd1023) begin
                    time_6_ram[7] <= time_6_ram[7] + 1'b1;
                end
                if (time_6_ram[8] < 10'd1023) begin
                    time_6_ram[8] <= time_6_ram[8] + 1'b1;
                end
                if (time_6_ram[9] < 10'd1023) begin
                    time_6_ram[9] <= time_6_ram[9] + 1'b1;
                end
                if (time_6_ram[10] < 10'd1023) begin
                    time_6_ram[10] <= time_6_ram[10] + 1'b1;
                end
            end
        end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*4) begin
            if (time_counter == 3'd0) begin
                if (time_0_ram[4] < 10'd1023) begin
                    time_0_ram[4] <= time_0_ram[4] + 1'b1;
                end
                if (time_0_ram[5] < 10'd1023) begin
                    time_0_ram[5] <= time_0_ram[5] + 1'b1;
                end
                if (time_0_ram[6] < 10'd1023) begin
                    time_0_ram[6] <= time_0_ram[6] + 1'b1;
                end
                if (time_0_ram[7] < 10'd1023) begin
                    time_0_ram[7] <= time_0_ram[7] + 1'b1;
                end
                if (time_0_ram[8] < 10'd1023) begin
                    time_0_ram[8] <= time_0_ram[8] + 1'b1;
                end
                if (time_0_ram[9] < 10'd1023) begin
                    time_0_ram[9] <= time_0_ram[9] + 1'b1;
                end
                if (time_0_ram[10] < 10'd1023) begin
                    time_0_ram[10] <= time_0_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd1) begin
                if (time_1_ram[4] < 10'd1023) begin
                    time_1_ram[4] <= time_1_ram[4] + 1'b1;
                end
                if (time_1_ram[5] < 10'd1023) begin
                    time_1_ram[5] <= time_1_ram[5] + 1'b1;
```

```
            end
            if (time_1_ram[6] < 10'd1023) begin
                time_1_ram[6] <= time_1_ram[6] + 1'b1;
            end
            if (time_1_ram[7] < 10'd1023) begin
                time_1_ram[7] <= time_1_ram[7] + 1'b1;
            end
            if (time_1_ram[8] < 10'd1023) begin
                time_1_ram[8] <= time_1_ram[8] + 1'b1;
            end
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[4] < 10'd1023) begin
                time_2_ram[4] <= time_2_ram[4] + 1'b1;
            end
            if (time_2_ram[5] < 10'd1023) begin
                time_2_ram[5] <= time_2_ram[5] + 1'b1;
            end
            if (time_2_ram[6] < 10'd1023) begin
                time_2_ram[6] <= time_2_ram[6] + 1'b1;
            end
            if (time_2_ram[7] < 10'd1023) begin
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[4] < 10'd1023) begin
                time_3_ram[4] <= time_3_ram[4] + 1'b1;
            end
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
```

```
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[4] < 10'd1023) begin
                time_4_ram[4] <= time_4_ram[4] + 1'b1;
            end
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[4] < 10'd1023) begin
                time_5_ram[4] <= time_5_ram[4] + 1'b1;
            end
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd6) begin
            if (time_6_ram[4] < 10'd1023) begin
                time_6_ram[4] <= time_6_ram[4] + 1'b1;
            end
            if (time_6_ram[5] < 10'd1023) begin
                time_6_ram[5] <= time_6_ram[5] + 1'b1;
            end
            if (time_6_ram[6] < 10'd1023) begin
                time_6_ram[6] <= time_6_ram[6] + 1'b1;
            end
            if (time_6_ram[7] < 10'd1023) begin
                time_6_ram[7] <= time_6_ram[7] + 1'b1;
            end
            if (time_6_ram[8] < 10'd1023) begin
                time_6_ram[8] <= time_6_ram[8] + 1'b1;
            end
```

```
                    if (time_6_ram[9] < 10'd1023) begin
                        time_6_ram[9] <= time_6_ram[9] + 1'b1;
                    end
                    if (time_6_ram[10] < 10'd1023) begin
                        time_6_ram[10] <= time_6_ram[10] + 1'b1;
                    end
                end
            end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*5) begin
                if (time_counter == 3'd0) begin
                    if (time_0_ram[5] < 10'd1023) begin
                        time_0_ram[5] <= time_0_ram[5] + 1'b1;
                    end
                    if (time_0_ram[6] < 10'd1023) begin
                        time_0_ram[6] <= time_0_ram[6] + 1'b1;
                    end
                    if (time_0_ram[7] < 10'd1023) begin
                        time_0_ram[7] <= time_0_ram[7] + 1'b1;
                    end
                    if (time_0_ram[8] < 10'd1023) begin
                        time_0_ram[8] <= time_0_ram[8] + 1'b1;
                    end
                    if (time_0_ram[9] < 10'd1023) begin
                        time_0_ram[9] <= time_0_ram[9] + 1'b1;
                    end
                    if (time_0_ram[10] < 10'd1023) begin
                        time_0_ram[10] <= time_0_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd1) begin
                    if (time_1_ram[5] < 10'd1023) begin
                        time_1_ram[5] <= time_1_ram[5] + 1'b1;
                    end
                    if (time_1_ram[6] < 10'd1023) begin
                        time_1_ram[6] <= time_1_ram[6] + 1'b1;
                    end
                    if (time_1_ram[7] < 10'd1023) begin
                        time_1_ram[7] <= time_1_ram[7] + 1'b1;
                    end
                    if (time_1_ram[8] < 10'd1023) begin
                        time_1_ram[8] <= time_1_ram[8] + 1'b1;
                    end
                    if (time_1_ram[9] < 10'd1023) begin
                        time_1_ram[9] <= time_1_ram[9] + 1'b1;
                    end
                    if (time_1_ram[10] < 10'd1023) begin
                        time_1_ram[10] <= time_1_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd2) begin
                    if (time_2_ram[5] < 10'd1023) begin
                        time_2_ram[5] <= time_2_ram[5] + 1'b1;
                    end
                    if (time_2_ram[6] < 10'd1023) begin
                        time_2_ram[6] <= time_2_ram[6] + 1'b1;
                    end
                    if (time_2_ram[7] < 10'd1023) begin
                        time_2_ram[7] <= time_2_ram[7] + 1'b1;
                    end
                    if (time_2_ram[8] < 10'd1023) begin
                        time_2_ram[8] <= time_2_ram[8] + 1'b1;
                    end
                    if (time_2_ram[9] < 10'd1023) begin
```

```
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[5] < 10'd1023) begin
                time_3_ram[5] <= time_3_ram[5] + 1'b1;
            end
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[5] < 10'd1023) begin
                time_4_ram[5] <= time_4_ram[5] + 1'b1;
            end
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[5] < 10'd1023) begin
                time_5_ram[5] <= time_5_ram[5] + 1'b1;
            end
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
```

```
                    time_5_ram[10] <= time_5_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd6) begin
                if (time_6_ram[5] < 10'd1023) begin
                    time_6_ram[5] <= time_6_ram[5] + 1'b1;
                end
                if (time_6_ram[6] < 10'd1023) begin
                    time_6_ram[6] <= time_6_ram[6] + 1'b1;
                end
                if (time_6_ram[7] < 10'd1023) begin
                    time_6_ram[7] <= time_6_ram[7] + 1'b1;
                end
                if (time_6_ram[8] < 10'd1023) begin
                    time_6_ram[8] <= time_6_ram[8] + 1'b1;
                end
                if (time_6_ram[9] < 10'd1023) begin
                    time_6_ram[9] <= time_6_ram[9] + 1'b1;
                end
                if (time_6_ram[10] < 10'd1023) begin
                    time_6_ram[10] <= time_6_ram[10] + 1'b1;
                end
            end
        end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*6) begin
            if (time_counter == 3'd0) begin
                if (time_0_ram[6] < 10'd1023) begin
                    time_0_ram[6] <= time_0_ram[6] + 1'b1;
                end
                if (time_0_ram[7] < 10'd1023) begin
                    time_0_ram[7] <= time_0_ram[7] + 1'b1;
                end
                if (time_0_ram[8] < 10'd1023) begin
                    time_0_ram[8] <= time_0_ram[8] + 1'b1;
                end
                if (time_0_ram[9] < 10'd1023) begin
                    time_0_ram[9] <= time_0_ram[9] + 1'b1;
                end
                if (time_0_ram[10] < 10'd1023) begin
                    time_0_ram[10] <= time_0_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd1) begin
                if (time_1_ram[6] < 10'd1023) begin
                    time_1_ram[6] <= time_1_ram[6] + 1'b1;
                end
                if (time_1_ram[7] < 10'd1023) begin
                    time_1_ram[7] <= time_1_ram[7] + 1'b1;
                end
                if (time_1_ram[8] < 10'd1023) begin
                    time_1_ram[8] <= time_1_ram[8] + 1'b1;
                end
                if (time_1_ram[9] < 10'd1023) begin
                    time_1_ram[9] <= time_1_ram[9] + 1'b1;
                end
                if (time_1_ram[10] < 10'd1023) begin
                    time_1_ram[10] <= time_1_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd2) begin
                if (time_2_ram[6] < 10'd1023) begin
                    time_2_ram[6] <= time_2_ram[6] + 1'b1;
                end
                if (time_2_ram[7] < 10'd1023) begin
```

```
                time_2_ram[7] <= time_2_ram[7] + 1'b1;
            end
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[6] < 10'd1023) begin
                time_3_ram[6] <= time_3_ram[6] + 1'b1;
            end
            if (time_3_ram[7] < 10'd1023) begin
                time_3_ram[7] <= time_3_ram[7] + 1'b1;
            end
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[6] < 10'd1023) begin
                time_4_ram[6] <= time_4_ram[6] + 1'b1;
            end
            if (time_4_ram[7] < 10'd1023) begin
                time_4_ram[7] <= time_4_ram[7] + 1'b1;
            end
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[6] < 10'd1023) begin
                time_5_ram[6] <= time_5_ram[6] + 1'b1;
            end
            if (time_5_ram[7] < 10'd1023) begin
                time_5_ram[7] <= time_5_ram[7] + 1'b1;
            end
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd6) begin
```

```
                    if (time_6_ram[6] < 10'd1023) begin
                        time_6_ram[6] <= time_6_ram[6] + 1'b1;
                    end
                    if (time_6_ram[7] < 10'd1023) begin
                        time_6_ram[7] <= time_6_ram[7] + 1'b1;
                    end
                    if (time_6_ram[8] < 10'd1023) begin
                        time_6_ram[8] <= time_6_ram[8] + 1'b1;
                    end
                    if (time_6_ram[9] < 10'd1023) begin
                        time_6_ram[9] <= time_6_ram[9] + 1'b1;
                    end
                    if (time_6_ram[10] < 10'd1023) begin
                        time_6_ram[10] <= time_6_ram[10] + 1'b1;
                    end
                end
            end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*7) begin
                if (time_counter == 3'd0) begin
                    if (time_0_ram[7] < 10'd1023) begin
                        time_0_ram[7] <= time_0_ram[7] + 1'b1;
                    end
                    if (time_0_ram[8] < 10'd1023) begin
                        time_0_ram[8] <= time_0_ram[8] + 1'b1;
                    end
                    if (time_0_ram[9] < 10'd1023) begin
                        time_0_ram[9] <= time_0_ram[9] + 1'b1;
                    end
                    if (time_0_ram[10] < 10'd1023) begin
                        time_0_ram[10] <= time_0_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd1) begin
                    if (time_1_ram[7] < 10'd1023) begin
                        time_1_ram[7] <= time_1_ram[7] + 1'b1;
                    end
                    if (time_1_ram[8] < 10'd1023) begin
                        time_1_ram[8] <= time_1_ram[8] + 1'b1;
                    end
                    if (time_1_ram[9] < 10'd1023) begin
                        time_1_ram[9] <= time_1_ram[9] + 1'b1;
                    end
                    if (time_1_ram[10] < 10'd1023) begin
                        time_1_ram[10] <= time_1_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd2) begin
                    if (time_2_ram[7] < 10'd1023) begin
                        time_2_ram[7] <= time_2_ram[7] + 1'b1;
                    end
                    if (time_2_ram[8] < 10'd1023) begin
                        time_2_ram[8] <= time_2_ram[8] + 1'b1;
                    end
                    if (time_2_ram[9] < 10'd1023) begin
                        time_2_ram[9] <= time_2_ram[9] + 1'b1;
                    end
                    if (time_2_ram[10] < 10'd1023) begin
                        time_2_ram[10] <= time_2_ram[10] + 1'b1;
                    end
                end else if (time_counter == 3'd3) begin
                    if (time_3_ram[7] < 10'd1023) begin
                        time_3_ram[7] <= time_3_ram[7] + 1'b1;
                    end
```

```
                if (time_3_ram[8] < 10'd1023) begin
                    time_3_ram[8] <= time_3_ram[8] + 1'b1;
                end
                if (time_3_ram[9] < 10'd1023) begin
                    time_3_ram[9] <= time_3_ram[9] + 1'b1;
                end
                if (time_3_ram[10] < 10'd1023) begin
                    time_3_ram[10] <= time_3_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd4) begin
                if (time_4_ram[7] < 10'd1023) begin
                    time_4_ram[7] <= time_4_ram[7] + 1'b1;
                end
                if (time_4_ram[8] < 10'd1023) begin
                    time_4_ram[8] <= time_4_ram[8] + 1'b1;
                end
                if (time_4_ram[9] < 10'd1023) begin
                    time_4_ram[9] <= time_4_ram[9] + 1'b1;
                end
                if (time_4_ram[10] < 10'd1023) begin
                    time_4_ram[10] <= time_4_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd5) begin
                if (time_5_ram[7] < 10'd1023) begin
                    time_5_ram[7] <= time_5_ram[7] + 1'b1;
                end
                if (time_5_ram[8] < 10'd1023) begin
                    time_5_ram[8] <= time_5_ram[8] + 1'b1;
                end
                if (time_5_ram[9] < 10'd1023) begin
                    time_5_ram[9] <= time_5_ram[9] + 1'b1;
                end
                if (time_5_ram[10] < 10'd1023) begin
                    time_5_ram[10] <= time_5_ram[10] + 1'b1;
                end
            end else if (time_counter == 3'd6) begin
                if (time_6_ram[7] < 10'd1023) begin
                    time_6_ram[7] <= time_6_ram[7] + 1'b1;
                end
                if (time_6_ram[8] < 10'd1023) begin
                    time_6_ram[8] <= time_6_ram[8] + 1'b1;
                end
                if (time_6_ram[9] < 10'd1023) begin
                    time_6_ram[9] <= time_6_ram[9] + 1'b1;
                end
                if (time_6_ram[10] < 10'd1023) begin
                    time_6_ram[10] <= time_6_ram[10] + 1'b1;
                end
            end
        end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*8) begin
            if (time_counter == 3'd0) begin
                if (time_0_ram[8] < 10'd1023) begin
                    time_0_ram[8] <= time_0_ram[8] + 1'b1;
                end
                if (time_0_ram[9] < 10'd1023) begin
                    time_0_ram[9] <= time_0_ram[9] + 1'b1;
                end
                if (time_0_ram[10] < 10'd1023) begin
                    time_0_ram[10] <= time_0_ram[10] + 1'b1;
                end
```

```
        end else if (time_counter == 3'd1) begin
            if (time_1_ram[8] < 10'd1023) begin
                time_1_ram[8] <= time_1_ram[8] + 1'b1;
            end
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[8] < 10'd1023) begin
                time_2_ram[8] <= time_2_ram[8] + 1'b1;
            end
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[8] < 10'd1023) begin
                time_3_ram[8] <= time_3_ram[8] + 1'b1;
            end
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[8] < 10'd1023) begin
                time_4_ram[8] <= time_4_ram[8] + 1'b1;
            end
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[8] < 10'd1023) begin
                time_5_ram[8] <= time_5_ram[8] + 1'b1;
            end
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd6) begin
            if (time_6_ram[8] < 10'd1023) begin
                time_6_ram[8] <= time_6_ram[8] + 1'b1;
            end
            if (time_6_ram[9] < 10'd1023) begin
                time_6_ram[9] <= time_6_ram[9] + 1'b1;
            end
            if (time_6_ram[10] < 10'd1023) begin
                time_6_ram[10] <= time_6_ram[10] + 1'b1;
            end
```

```
        end
    end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*9) begin
        if (time_counter == 3'd0) begin
            if (time_0_ram[9] < 10'd1023) begin
                time_0_ram[9] <= time_0_ram[9] + 1'b1;
            end
            if (time_0_ram[10] < 10'd1023) begin
                time_0_ram[10] <= time_0_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd1) begin
            if (time_1_ram[9] < 10'd1023) begin
                time_1_ram[9] <= time_1_ram[9] + 1'b1;
            end
            if (time_1_ram[10] < 10'd1023) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd2) begin
            if (time_2_ram[9] < 10'd1023) begin
                time_2_ram[9] <= time_2_ram[9] + 1'b1;
            end
            if (time_2_ram[10] < 10'd1023) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd3) begin
            if (time_3_ram[9] < 10'd1023) begin
                time_3_ram[9] <= time_3_ram[9] + 1'b1;
            end
            if (time_3_ram[10] < 10'd1023) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd4) begin
            if (time_4_ram[9] < 10'd1023) begin
                time_4_ram[9] <= time_4_ram[9] + 1'b1;
            end
            if (time_4_ram[10] < 10'd1023) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd5) begin
            if (time_5_ram[9] < 10'd1023) begin
                time_5_ram[9] <= time_5_ram[9] + 1'b1;
            end
            if (time_5_ram[10] < 10'd1023) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end
        end else if (time_counter == 3'd6) begin
            if (time_6_ram[9] < 10'd1023) begin
                time_6_ram[9] <= time_6_ram[9] + 1'b1;
            end
            if (time_6_ram[10] < 10'd1023) begin
                time_6_ram[10] <= time_6_ram[10] + 1'b1;
            end
        end
    end else if (data_in[15:0] <= start_alt + HEIGHT_INCREMENT*10) begin
        if (
            (time_counter == 3'd0) &&
            (time_0_ram[10] < 10'd1023)
           ) begin
            time_0_ram[10] <= time_0_ram[10] + 1'b1;
        end else if (
                     (time_counter == 3'd1) &&
```

```
                        (time_1_ram[10] < 10'd1023)
                    ) begin
                time_1_ram[10] <= time_1_ram[10] + 1'b1;
            end else if (
                        (time_counter == 3'd2) &&
                        (time_2_ram[10] < 10'd1023)
                    ) begin
                time_2_ram[10] <= time_2_ram[10] + 1'b1;
            end else if (
                        (time_counter == 3'd3) &&
                        (time_3_ram[10] < 10'd1023)
                    ) begin
                time_3_ram[10] <= time_3_ram[10] + 1'b1;
            end else if (
                        (time_counter == 3'd4) &&
                        (time_4_ram[10] < 10'd1023)
                    ) begin
                time_4_ram[10] <= time_4_ram[10] + 1'b1;
            end else if (
                        (time_counter == 3'd5) &&
                        (time_5_ram[10] < 10'd1023)
                    ) begin
                time_5_ram[10] <= time_5_ram[10] + 1'b1;
            end else if (
                        (time_counter == 3'd6) &&
                        (time_6_ram[10] < 10'd1023)
                    ) begin
                time_6_ram[10] <= time_6_ram[10] + 1'b1;
            end
        end
    end
  end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module xvga_display(
                    input  wire         clk_in,
                    input  wire         started_in,
                    input  wire  [11:0] rgb_in,
                    input  wire         bram_data_in,
                    input  wire         reset,
                    input  wire         up,
                    input  wire         down,
                    input  wire         left,
                    input  wire         right,
                    input  wire  [2:0]  sel_time_in,
                    input  wire  [10:0] start_alt,
                    input  wire  [3:0]  sel_alt_in,

                    output logic [17:0] bram_addr_out,
                    output logic [3:0]  vga_r,
                    output logic [3:0]  vga_g,
                    output logic [3:0]  vga_b,
                    output logic        vga_hs,
                    output logic        vga_vs,
```

```
                output logic [5:0]  time_start_counter,
                output logic [5:0]  time_end_counter,
                output logic [10:0] alt_start_counter,
                output logic [10:0] alt_end_counter
               );

// IP: convert clock to 65MHz clock for XVGA control
logic clk_65mhz;
clk_wiz_0 clkconverter (.clk_in1(clk_in),
                        .clk_out1(clk_65mhz));

// use XVGA module to determine control signals
logic [10:0] hcount;
logic [9:0] vcount;
logic hsync, vsync, blank;
logic [11:0] rgb;
xvga xvga1 (.vclock_in(clk_65mhz),
            .hcount_out(hcount),
            .vcount_out(vcount),
            .hsync_out(hsync),
            .vsync_out(vsync),
            .blank_out(blank));

// determine pixel output for and selected values from landing screen
logic [11:0] landing_screen_pixel;
landing_screen_picture_blob #(.WIDTH(512),
                              .HEIGHT(384),
                              .ARROW_WIDTH(45),
                              .ARROW_HEIGHT(45),
                              .DIGIT_WIDTH(24),
                              .DIGIT_HEIGHT(24))
    input_launch_parameters (.pixel_clk_in(clk_65mhz),
                             .hcount_in(hcount),
                             .vcount_in(vcount),
                             .rst_in(reset),
                             .up(up),
                             .down(down),
                             .left(left),
                             .right(right),
                             .pixel_out(landing_screen_pixel),
                             .time_start_counter(time_start_counter),
                             .time_end_counter(time_end_counter),
                             .alt_start_counter(alt_start_counter),
                             .alt_end_counter(alt_end_counter));

// determine pixel output for the optimized display, excluding dots
logic [11:0] optimized_pixel;
logic [5:0] optimal_time;
assign optimal_time = sel_time_in*(4'd10);
logic [10:0] optimal_altitude;
assign optimal_altitude = start_alt + sel_alt_in*(6'd50);
optimized_picture_blob #(.WIDTH(512),
                         .HEIGHT(384),
                         .DIGIT_WIDTH(24),
                         .DIGIT_HEIGHT(24),
                         .OPT_WIDTH(128),
                         .OPT_HEIGHT(53))
    display_optimized (.pixel_clk_in(clk_65mhz),
                       .hcount_in(hcount),
                       .vcount_in(vcount),
```

```
                        .optimal_time(optimal_time),
                        .optimal_altitude(optimal_altitude),
                        .rgb_in(rgb_in),
                        .pixel_out(optimized_pixel));

    // get address to dot BRAM based on hcount and vcount
    dot_plot dots (.vclock_in(clk_in),
                   .hcount_in(hcount),
                   .vcount_in(vcount),
                   .bram_addr_out(bram_addr_out));

    // set up stages to wait for data from dots BRAM
    logic b_stage1, hs_stage1, vs_stage1;
    logic [11:0] rgb_stage1;
    logic b_stage2, hs_stage2, vs_stage2;
    logic [11:0] rgb_stage2;
    logic b,hs,vs;

    always_ff @(posedge clk_65mhz) begin
        // delay signals for one cycle
        hs_stage1 <= hsync;
        vs_stage1 <= vsync;
        b_stage1 <= blank;

        // delay signals for another cycle, and select pixel value
        // between optimized and landing screen
        hs_stage2 <= hs_stage1;
        vs_stage2 <= vs_stage1;
        b_stage2 <= b_stage1;
        rgb_stage2 <= (started_in) ? optimized_pixel : landing_screen_pixel;

        // output signals, and further select between dots and previous stage
        hs <= hs_stage2;
        vs <= vs_stage2;
        b <= b_stage2;
        rgb <= (started_in && (bram_data_in == 1)) ? 12'b0000_0000_1111 : rgb_stage2;
    end

    // logic to convert XVGA signals
    assign vga_r = ~b? rgb[11:8] : 0;
    assign vga_g = ~b? rgb[7:4] : 0;
    assign vga_b = ~b? rgb[3:0] : 0;
    assign vga_hs = ~hs;
    assign vga_vs = ~vs;
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module dot_determiner(
                      input  wire         clk_in,
                      input  wire         start,
                      input  wire  [32:0] data_in,
                      input  wire         all_data_in,
                      input  wire  [10:0] start_alt,
                      input  wire  [3:0]  sel_alt_in,
                      input  wire  [2:0]  sel_time_in,
```

```
                    output logic        ongoing_out,
                    output logic        data_out,
                    output logic [17:0] bram_addr,
                    output logic        bram_data,
                    output logic        end_pulse
                   );

// same parameters as density calculator
    parameter HEIGHT_INCREMENT = 50; //10 km
    parameter LAUNCH_LAT = 120; //60 deg South
    parameter LAUNCH_LON = 85; //150 deg East
    parameter RADIUS = 32; //2 deg

// state indicators
logic resetting;
logic [2:0] time_counter;
logic [1:0] await;

// combinationally compute next addr to write into dot BRAM based on
// where the x- and y- coordinates of the satellite would map to
// in a reduced (512 x 384) pixel space
logic signed [18:0] v_diff, h_diff;
assign v_diff = data_in[24:16] - LAUNCH_LON;
assign h_diff = data_in[32:25] - LAUNCH_LAT;
logic signed [18:0] next_addr;
assign next_addr = (
                    (((v_diff + RADIUS)*3) << 9) +
                    ((h_diff + RADIUS)*3) +
                    RADIUS
                   ) << 1;

// find the maximum altitude we're considering
// based on the starting and selected altitude
logic [15:0] alt_cap;
assign alt_cap = start_alt + HEIGHT_INCREMENT*sel_alt_in;

always_ff @(posedge clk_in) begin
    if (start) begin
        // reset module
        resetting <= 1'b1;
        time_counter <= 0;
        await <= 2;
        ongoing_out <= 1'b0;
        data_out <= 1'b1;
        bram_addr <= 0;
        bram_data <= 0;
        end_pulse <= 1'b0;
    end else if (resetting) begin
        // start by clearing BRAM to remove old dots
        bram_data <= 0;
        if (bram_addr < 196608) begin
            bram_addr <= bram_addr + 1;
        end else begin
            bram_addr <= 0;
            resetting <= 0;
            ongoing_out <= 1;
        end
    end else if (ongoing_out) begin
        // change the BRAM address only when the satellite is in range
        if ((((((data_in[32:25])-LAUNCH_LAT)**2) +
```

```
                (((data_in[24:16])-LAUNCH_LON)**2)) <= (RADIUS**2)) &&
                  (data_in[15:0] <= alt_cap) &&
                  (time_counter == sel_time_in)) begin
                bram_addr <= next_addr[17:0];
                bram_data <= 1;
            end
            // step through the time counter each cycle
            if (all_data_in || time_counter == 3'd6) begin
                time_counter <= 3'b0;
            end else begin
                time_counter <= time_counter + 1;
            end
            if (all_data_in) begin
                // done on next cycle
                ongoing_out <= 1'b0;
                end_pulse <= 1'b1;
            end
        end else if (end_pulse) begin
            // end pulse is single cycle
            end_pulse <= 1'b0;
            data_out <= 1'b0;
        end
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module dot_plot (
                 input  wire         vclock_in,
                 input  wire  [10:0] hcount_in,
                 input  wire  [9:0]  vcount_in,

                 output logic [17:0] bram_addr_out
                );

    logic [17:0] hcount_18, vcount_18;
    // divide input hcount by 2 and floor
    assign hcount_18 = {8'b0, hcount_in[10:1]};
    // divide input vcount by 2 and floor, then multiply by 512
    assign vcount_18 = {vcount_in[9:1], 9'b0};

    always @(posedge vclock_in) begin
        // the pixel address is the sum of the modified inputs
        bram_addr_out <= vcount_18 + hcount_18;
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module fir_2d (
               input  wire         clk_in,
               input  wire         rst_in,
               input  wire         ready_in,
               input  wire  [9:0]  x_in,
```

```
            output logic [13:0] y_out,
            output logic        done_out
           );

// state indicators
logic waiting;
logic [3:0] offset;
logic signed [4:0] fir_cycle;
logic [4:0] input_counter;
integer i;

// array to store samples
logic [9:0] samples [8:0];

// compute next sum combinationally
logic [13:0] res;
logic [13:0] new_res;
assign new_res = res + samples[offset-fir_cycle[3:0]];

always_ff @(posedge clk_in) begin
    if (rst_in) begin
        // reset module
        waiting <= 1;
        offset <= 0;
        fir_cycle <= -1;
        input_counter <= 0;
        for (i = 0; i < 9; i = i+1) begin
            samples[i] <= 0;
        end
        y_out <= 0;
        done_out <= 0;
    end else if (ready_in) begin
        // insert new value to array
        offset <= offset + 1;
        samples[offset+1] <= x_in;
        // reset computation
        fir_cycle <= 0;
        res <= 0;
        done_out <= 0;
        if (input_counter < 9) begin
            input_counter <= input_counter + 1;
        end
        waiting <= 0;
    end else begin
        // reassign res on the clock cycle
        res <= new_res;
        if ((input_counter < 9) && (!waiting)) begin
            // before 9 inputs have been given, output dummy data
            y_out <= -1;
            done_out <= 1;
            waiting <= 1;
        end else if (fir_cycle == 8) begin
            // otherwise output actual data
            y_out <= new_res;
            done_out <= 1;
        end else begin
            // done is a single cycle pulse
            done_out <= 0;
        end
        // step through the fir_cycle (index offset) each cycle
```

```
            if ((fir_cycle >= 0) && (fir_cycle != 9)) begin
                fir_cycle <= fir_cycle + 1;
            end
        end
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module gradient (
                 input  wire         clk_in,
                 input  wire  [9:0]  density_in,

                 output logic [11:0] rgb_out
                );

    // combinationally computed red and green values
    logic [3:0] red;
    logic [3:0] green;

    always_comb begin
        if (density_in < 10'd256) begin
            // linearly scale red on {0..255} -> {0..15}
            red = density_in[7:4];
            green = 4'b1111;
        end else if (density_in < 10'd768) begin
            red = 4'b1111;
            // linearly scale green on {256..767} to {15..0}
            green = 5'd23 - density_in[9:5];
        end else begin
            // output pure red on {767..1023}
            red = 4'b1111;
            green = 4'b0000;
        end
    end

    always_ff @(posedge clk_in) begin
        rgb_out <= {red, green, 4'b0};
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module landing_screen_picture_blob #(parameter              // display dimensions
                                     WIDTH        = 512,
                                     HEIGHT       = 384,
                                     ARROW_WIDTH  = 45,
                                     ARROW_HEIGHT = 45,
                                     DIGIT_WIDTH  = 24,
                                     DIGIT_HEIGHT = 24
                                    )
                                    (
                                     input  wire         pixel_clk_in,
                                     input  wire  [10:0] hcount_in,
                                     input  wire  [9:0]  vcount_in,
                                     input  wire         rst_in,
                                     input  wire         up,
                                     input  wire         down,
                                     input  wire         left,
                                     input  wire         right,

                                     output logic [11:0] pixel_out,
```

```
                                output logic [5:0]  time_start_counter,
                                output logic [5:0]  time_end_counter,
                                output logic [10:0] alt_start_counter,
                                output logic [10:0] alt_end_counter
                               );

    logic [17:0] image_addr;   // num of bits for 512*384 ROM (18 bits)
    logic [7:0] image_bits;

    // calculate rom address and read the location
    // (scaled down by a factor of 2 in each dimension for memory constraints)
    assign image_addr = (hcount_in>>1) + ((vcount_in>>1) * WIDTH);
    reduced_landing_screen rom1 (.clka(pixel_clk_in),
                                 .addra(image_addr),
                                 .douta(image_bits));

    logic [10:0] arrow_addr; // num of bits for 45x45 ROM (11 bits)
    logic [7:0] arrow_image_bits;
    arrow_bw_image_rom red_arrow (.clka(pixel_clk_in),
                                  .addra(arrow_addr),
                                  .douta(arrow_image_bits));

    // 12 digits
    logic [12:0] digit_a_addr,
                 digit_b_addr,
                 digit_c_addr,
                 digit_d_addr,
                 digit_e_addr,
                 digit_f_addr,
                 digit_g_addr,
                 digit_h_addr,
                 digit_i_addr,
                 digit_j_addr,
                 digit_k_addr,
                 digit_l_addr;
    logic [7:0] digit_a_image_bits,
                digit_b_image_bits,
                digit_c_image_bits,
                digit_d_image_bits,
                digit_e_image_bits,
                digit_f_image_bits,
                digit_g_image_bits,
                digit_h_image_bits,
                digit_i_image_bits,
                digit_j_image_bits,
                digit_k_image_bits,
                digit_l_image_bits;
    digits_reduced_image_rom digit_a (.clka(pixel_clk_in),
                                      .addra(digit_a_addr),
                                      .douta(digit_a_image_bits));
    digits_reduced_image_rom digit_b (.clka(pixel_clk_in),
                                      .addra(digit_b_addr),
                                      .douta(digit_b_image_bits));
    digits_reduced_image_rom digit_c (.clka(pixel_clk_in),
                                      .addra(digit_c_addr),
                                      .douta(digit_c_image_bits));
    digits_reduced_image_rom digit_d (.clka(pixel_clk_in),
                                      .addra(digit_d_addr),
                                      .douta(digit_d_image_bits));
    digits_reduced_image_rom digit_e (.clka(pixel_clk_in),
                                      .addra(digit_e_addr),
                                      .douta(digit_e_image_bits));
    digits_reduced_image_rom digit_f (.clka(pixel_clk_in),
                                      .addra(digit_f_addr),
                                      .douta(digit_f_image_bits));
    digits_reduced_image_rom digit_g (.clka(pixel_clk_in),
                                      .addra(digit_g_addr),
                                      .douta(digit_g_image_bits));
    digits_reduced_image_rom digit_h (.clka(pixel_clk_in),
                                      .addra(digit_h_addr),
                                      .douta(digit_h_image_bits));
    digits_reduced_image_rom digit_i (.clka(pixel_clk_in),
                                      .addra(digit_i_addr),
                                      .douta(digit_i_image_bits));
    digits_reduced_image_rom digit_j (.clka(pixel_clk_in),
                                      .addra(digit_j_addr),
                                      .douta(digit_j_image_bits));
    digits_reduced_image_rom digit_k (.clka(pixel_clk_in),
                                      .addra(digit_k_addr),
                                      .douta(digit_k_image_bits));
    digits_reduced_image_rom digit_l (.clka(pixel_clk_in),
                                      .addra(digit_l_addr),
                                      .douta(digit_l_image_bits));

    // last button inputs
    logic last_up, last_down, last_left, last_right;
```

```
    logic [2:0] arrow_counter;

    always_comb begin
        case(arrow_counter)
            // shift the arrow location based on arrow_counter
            3'd0 : arrow_addr = 11'd0;
            3'd1 : arrow_addr = (hcount_in - 11'd408) + ((vcount_in - 10'd280)*ARROW_WIDTH);
            3'd2 : arrow_addr = (hcount_in - 11'd408) + ((vcount_in - 10'd360)*ARROW_WIDTH);
            3'd3 : arrow_addr = (hcount_in - 11'd408) + ((vcount_in - 10'd508)*ARROW_WIDTH);
            3'd4 : arrow_addr = (hcount_in - 11'd408) + ((vcount_in - 10'd580)*ARROW_WIDTH);
            3'd5 : arrow_addr = (hcount_in - 11'd570) + ((vcount_in - 10'd690)*ARROW_WIDTH);
            default : arrow_addr = 11'd0;
        endcase

        case(time_start_counter)
            // for each digit (2 digits -- tenths, ones place -- find proper address)
            6'd0 : begin
                digit_a_addr = 13'd0;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd10 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd576;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd20 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd1152;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd30 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd1728;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd40 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd2304;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd50 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd2880;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            6'd60 : begin
                digit_a_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH) + 15'd3456;
                digit_b_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd275)>>1)*DIGIT_WIDTH);
            end
            default : begin
                digit_a_addr = 13'd0;
                digit_b_addr = 13'd0;
            end
        endcase

        case(time_end_counter)
            // for each digit (2 digits -- tenths, ones place -- find proper address)
            6'd0 : begin
                digit_c_addr = 13'd0;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd10 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd576;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd20 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd1152;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd30 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd1728;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd40 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd2304;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd50 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd2880;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            6'd60 : begin
                digit_c_addr = ((hcount_in - 11'd610)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH) + 13'd3456;
                digit_d_addr = ((hcount_in - 11'd658)>>1) + (((vcount_in - 10'd350)>>1)*DIGIT_WIDTH);
            end
            default : begin
                digit_c_addr = 13'd0;
                digit_d_addr = 13'd0;
            end
        endcase

        case(alt_start_counter)
```

```
// for each digit (4 digits -- thousandths, hundredths, tenths, ones place -- find proper address)
11'd0 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = 13'd0;
    digit_g_addr = 13'd0;
    digit_h_addr = 13'd0;
end
11'd250 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1152; //200
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd300 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1728; //300
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd350 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1728; //300
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd400 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2304; //400
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd450 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2304; //400
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd500 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //500
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd550 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //500
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd600 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd3456; //600
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd650 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd3456; //600
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd700 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4032; //700
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd750 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 15'd4032; //700
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd800 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4608; //800
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd850 : begin
    digit_e_addr = 13'd0;
    digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4608; //800
    digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
    digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
end
11'd900 : begin
    digit_e_addr = 13'd0;
```

```
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd950 : begin
            digit_e_addr = 13'd0;
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1000 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1050 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1100 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //100
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1150 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //100
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1200 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1250 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1300 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1350 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1400 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1450 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1500 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1550 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1600 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
```

```
        end
        11'd1650 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1700 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4032; //700
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1750 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 15'd4032; //700
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1800 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1850 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1900 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd1950 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        11'd2000 : begin
            digit_e_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH) + 13'd1152; //2000
            digit_f_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_g_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
            digit_h_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd510)>>1)*DIGIT_WIDTH); //0
        end
        default : begin
            digit_e_addr = 13'd0;
            digit_f_addr = 13'd0;
            digit_g_addr = 13'd0;
            digit_h_addr = 13'd0;
        end
    endcase

    case(alt_end_counter)
        // for each digit (4 digits -- thousandths, hundredths, tenths, ones place -- find proper address)
        11'd0 : begin
            digit_i_addr = 13'd0;
            digit_j_addr = 13'd0;
            digit_k_addr = 13'd0;
            digit_l_addr = 13'd0;
        end
        11'd250 : begin
            digit_i_addr = 13'd0;
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd300 : begin
            digit_i_addr = 13'd0;
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd350 : begin
            digit_i_addr = 13'd0;
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd400 : begin
            digit_i_addr = 13'd0;
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
```

```
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd450 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2304; //400
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd500 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //500
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd550 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //500
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd600 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd3456; //600
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd650 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd3456; //600
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd700 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4032; //700
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd750 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 15'd4032; //700
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd800 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4608; //800
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd850 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4608; //800
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd900 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd5184; //900
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd950 : begin
        digit_i_addr = 13'd0;
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd5184; //900
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd1000 : begin
        digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd1050 : begin
        digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd1100 : begin
        digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //100
        digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
    end
    11'd1150 : begin
```

```
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //100
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1200 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1250 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1300 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1350 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1400 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1450 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1500 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1550 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1600 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1650 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1700 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4032; //700
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1750 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 15'd4032; //700
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1800 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1850 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
```

```
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1900 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd1950 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd576; //1000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        11'd2000 : begin
            digit_i_addr = ((hcount_in - 11'd560)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH) + 13'd1152; //2000
            digit_j_addr = ((hcount_in - 11'd608)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_k_addr = ((hcount_in - 11'd656)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
            digit_l_addr = ((hcount_in - 11'd704)>>1) + (((vcount_in - 10'd580)>>1)*DIGIT_WIDTH); //0
        end
        default : begin
            digit_i_addr = 13'd0;
            digit_j_addr = 13'd0;
            digit_k_addr = 13'd0;
            digit_l_addr = 13'd0;
        end
    endcase
end

always_ff @ (posedge pixel_clk_in) begin
    if (rst_in) begin
        // reset counters
        arrow_counter <= 3'd0;
        time_start_counter <= 6'd0;
        time_end_counter <= 6'd0;
        alt_start_counter <= 11'd0;
        alt_end_counter <= 11'd0;
    end else if ((down == 1'b0) && (last_down == 1'b1)) begin
        // down button pressed and released
        // increment counter, changes arrow location
        if (arrow_counter == 3'd5) begin
            arrow_counter <= 3'd0;
        end else begin
            arrow_counter <= arrow_counter + 1'b1;
        end
    end else if ((up == 1'b0) && (last_up == 1'b1)) begin
        // up button pressed and released
        // decrement counter, changes arrow location
        if (arrow_counter == 3'd0) begin
            arrow_counter <= 3'd5;
        end else begin
            arrow_counter <= arrow_counter - 1'b1;
        end
    end else if ((right == 1'b0) && (last_right == 1'b1)) begin
        // right button pressed and released
        // increment timers, changes digits displayed
        if (arrow_counter == 3'd1) begin
            if (time_start_counter == 6'd60) begin
                time_start_counter <= 6'd0;
            end else begin
                time_start_counter <= time_start_counter + 6'd10;
            end
        end else if (arrow_counter == 3'd2) begin
            if (time_end_counter == 6'd60) begin
                time_end_counter <= 6'd0;
            end else begin
                time_end_counter <= time_end_counter + 6'd10;
            end
        end else if (arrow_counter == 3'd3) begin
            if (alt_start_counter == 11'd0) begin // capture reset state
                alt_start_counter <= 11'd250;
            end else if (alt_start_counter == 11'd2000) begin
                alt_start_counter <= 11'd250;
            end else begin
                alt_start_counter <= alt_start_counter + 11'd50;
            end
        end else if (arrow_counter == 3'd4) begin
            if (alt_end_counter == 11'd0) begin // capture reset state
                alt_end_counter <= 11'd250;
            end else if (alt_end_counter == 11'd2000) begin
                alt_end_counter <= 11'd250;
            end else begin
                alt_end_counter <= alt_end_counter + 11'd50;
            end
        end
    end else if ((left == 1'b0) && (last_left == 1'b1)) begin
        // left button pressed and released
```

```
        // decrement timers, changes digits displayed
        if (arrow_counter == 3'd1) begin
            if (time_start_counter == 6'd0) begin
                time_start_counter <= 6'd60;
            end else begin
                time_start_counter <= time_start_counter - 6'd10;
            end
        end else if (arrow_counter == 3'd2) begin
            if (time_end_counter == 6'd0) begin
                time_end_counter <= 6'd60;
            end else begin
                time_end_counter <= time_end_counter - 6'd10;
            end
        end else if (arrow_counter == 3'd3) begin
            if (alt_start_counter == 11'd0) begin // capture reset state
                alt_start_counter <= 11'd250;
            end else if (alt_start_counter == 11'd250) begin
                alt_start_counter <= 11'd2000;
            end else begin
                alt_start_counter <= alt_start_counter - 11'd50;
            end
        end else if (arrow_counter == 3'd4) begin
            if (alt_end_counter == 11'd0) begin // capture reset state
                alt_end_counter <= 11'd250;
            end else if (alt_end_counter == 11'd250) begin
                alt_end_counter <= 11'd2000;
            end else begin
                alt_end_counter <= alt_end_counter - 11'd50;
            end
        end
    end

    // logic for displaying pixels
    if ((hcount_in <= 1023) && (vcount_in <= 767)) begin
        if (arrow_counter == 3'd0) begin // no arrows
            pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
        end else if (arrow_counter == 3'd1) begin // time begin
            if ((hcount_in >= 11'd408) && (hcount_in <= 11'd452) && (vcount_in >= 10'd280) && (vcount_in <= 10'd322)) begin
                pixel_out <= {arrow_image_bits[7:4], arrow_image_bits[7:4], arrow_image_bits[7:4]}; // red
            end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
                pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
            end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
                pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
            end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
                pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
            end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
                pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
            end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
            end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
            end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_g_image_bits[7:4], digit_g_image_bits[7:4], digit_g_image_bits[7:4]};
            end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_h_image_bits[7:4], digit_h_image_bits[7:4], digit_h_image_bits[7:4]};
            end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
                pixel_out <= {digit_i_image_bits[7:4], digit_i_image_bits[7:4], digit_i_image_bits[7:4]};
            end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
                pixel_out <= {digit_j_image_bits[7:4], digit_j_image_bits[7:4], digit_j_image_bits[7:4]};
            end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
                pixel_out <= {digit_k_image_bits[7:4], digit_k_image_bits[7:4], digit_k_image_bits[7:4]};
            end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
                pixel_out <= {digit_l_image_bits[7:4], digit_l_image_bits[7:4], digit_l_image_bits[7:4]};
            end else begin
                pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
            end
        end else if (arrow_counter == 3'd2) begin // time end
            if ((hcount_in >= 11'd408) && (hcount_in <= 11'd452) && (vcount_in >= 10'd360) && (vcount_in <= 10'd404)) begin
                pixel_out <= {arrow_image_bits[7:4], arrow_image_bits[7:4], arrow_image_bits[7:4]}; // red
            end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
                pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
            end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
                pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
            end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
                pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
            end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
                pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
            end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
            end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
            end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_g_image_bits[7:4], digit_g_image_bits[7:4], digit_g_image_bits[7:4]};
            end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
                pixel_out <= {digit_h_image_bits[7:4], digit_h_image_bits[7:4], digit_h_image_bits[7:4]};
            end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
                pixel_out <= {digit_i_image_bits[7:4], digit_i_image_bits[7:4], digit_i_image_bits[7:4]};
```

```
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_j_image_bits[7:4], digit_j_image_bits[7:4], digit_j_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_k_image_bits[7:4], digit_k_image_bits[7:4], digit_k_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_l_image_bits[7:4], digit_l_image_bits[7:4], digit_l_image_bits[7:4]};
        end else begin
            pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
        end
    end else if (arrow_counter == 3'd3) begin // altitude begin
        if ((hcount_in >= 11'd408) && (hcount_in <= 11'd452) && (vcount_in >= 10'd508) && (vcount_in <= 10'd552)) begin
            pixel_out <= {arrow_image_bits[7:4], arrow_image_bits[7:4], arrow_image_bits[7:4]}; // red
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
        end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_g_image_bits[7:4], digit_g_image_bits[7:4], digit_g_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_h_image_bits[7:4], digit_h_image_bits[7:4], digit_h_image_bits[7:4]};
        end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_i_image_bits[7:4], digit_i_image_bits[7:4], digit_i_image_bits[7:4]};
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_j_image_bits[7:4], digit_j_image_bits[7:4], digit_j_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_k_image_bits[7:4], digit_k_image_bits[7:4], digit_k_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_l_image_bits[7:4], digit_l_image_bits[7:4], digit_l_image_bits[7:4]};
        end else begin
            pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
        end
    end else if (arrow_counter == 3'd4) begin // altitude end
        if ((hcount_in >= 11'd408) && (hcount_in <= 11'd452) && (vcount_in >= 10'd580) && (vcount_in <= 10'd624)) begin
            pixel_out <= {arrow_image_bits[7:4], arrow_image_bits[7:4], arrow_image_bits[7:4]}; // red
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
        end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_g_image_bits[7:4], digit_g_image_bits[7:4], digit_g_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_h_image_bits[7:4], digit_h_image_bits[7:4], digit_h_image_bits[7:4]};
        end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_i_image_bits[7:4], digit_i_image_bits[7:4], digit_i_image_bits[7:4]};
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_j_image_bits[7:4], digit_j_image_bits[7:4], digit_j_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_k_image_bits[7:4], digit_k_image_bits[7:4], digit_k_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) be
            pixel_out <= {digit_l_image_bits[7:4], digit_l_image_bits[7:4], digit_l_image_bits[7:4]};
        end else begin
            pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
        end
    end else if (arrow_counter == 3'd5) begin // optimize
        if ((hcount_in >= 11'd570) && (hcount_in <= 11'd614) && (vcount_in >= 10'd690) && (vcount_in <= 10'd734)) begin
            pixel_out <= {arrow_image_bits[7:4], arrow_image_bits[7:4], arrow_image_bits[7:4]}; // red
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd275) && (vcount_in <= 10'd322)) be
            pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
        end else if ((hcount_in >= 11'd610) && (hcount_in <= 11'd657) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
        end else if ((hcount_in >= 11'd658) && (hcount_in <= 11'd705) && (vcount_in >= 10'd350) && (vcount_in <= 10'd397)) be
            pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
        end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
        end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
        end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
            pixel_out <= {digit_g_image_bits[7:4], digit_g_image_bits[7:4], digit_g_image_bits[7:4]};
        end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd510) && (vcount_in <= 10'd557)) be
```

```
                pixel_out <= {digit_h_image_bits[7:4], digit_h_image_bits[7:4], digit_h_image_bits[7:4]};
            end else if ((hcount_in >= 11'd560) && (hcount_in <= 11'd607) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) b
                pixel_out <= {digit_i_image_bits[7:4], digit_i_image_bits[7:4], digit_i_image_bits[7:4]};
            end else if ((hcount_in >= 11'd608) && (hcount_in <= 11'd655) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) b
                pixel_out <= {digit_j_image_bits[7:4], digit_j_image_bits[7:4], digit_j_image_bits[7:4]};
            end else if ((hcount_in >= 11'd656) && (hcount_in <= 11'd703) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) b
                pixel_out <= {digit_k_image_bits[7:4], digit_k_image_bits[7:4], digit_k_image_bits[7:4]};
            end else if ((hcount_in >= 11'd704) && (hcount_in <= 11'd751) && (vcount_in >= 10'd580) && (vcount_in <= 10'd627)) b
                pixel_out <= {digit_l_image_bits[7:4], digit_l_image_bits[7:4], digit_l_image_bits[7:4]};
            end else begin
                pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
            end
        end else begin // catch-all (if not reset yet, don't show any arrows)
            pixel_out <= {image_bits[7:4], image_bits[7:4], image_bits[7:4]}; // greyscale
        end
    end else begin
        pixel_out <= 0;
    end

    // update last up, down, left, and right button user inputs
    last_up <= up;
    last_down <= down;
    last_left <= left;
    last_right <= right;
  end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module minimizer (
                  input  wire         clk_in,
                  input  wire         rst_in,
                  input  wire         valid_in,
                  input  wire  [3:0]  cur_alt,
                  input  wire  [2:0]  cur_time,
                  input  wire  [13:0] cur_density,

                  output logic [3:0]  min_alt,
                  output logic [2:0]  min_time,
                  output logic [13:0] min_density
                 );

    always_ff @(posedge clk_in) begin
        if (rst_in) begin
            // reset stored altitude, time, and density to filler
            min_alt <= -1;
            min_time <= -1;
            min_density <= 14'b11_1111_1111_1111;
        end else if (valid_in && (cur_density < min_density)) begin
            // on a valid input, check if the density is less than our minimum
            // update our outputs if so
            min_alt <= cur_alt;
            min_time <= cur_time;
            min_density <= cur_density;
        end
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module optimized_picture_blob #(parameter       // display dimensions
                                WIDTH        = 512,
                                HEIGHT       = 384,
                                DIGIT_WIDTH  = 24,
                                DIGIT_HEIGHT = 24,
                                OPT_WIDTH    = 128,
                                OPT_HEIGHT   = 53
```

```
                            )
                            (
                             input  wire         pixel_clk_in,
                             input  wire  [10:0] hcount_in,
                             input  wire  [9:0]  vcount_in,
                             input  wire  [5:0]  optimal_time,
                             input  wire  [10:0] optimal_altitude,
                             input  wire  [11:0] rgb_in,

                             output logic [11:0] pixel_out
                            );

logic [12:0] opt_addr; // num of bits for 128x53 ROM (13 bits)
logic [7:0] opt_bits;

// minutes, km string
assign opt_addr = ((hcount_in-11'd212) >>1) + (((vcount_in-10'd10)>>1)*OPT_WIDTH);
opt_bw_image_rom optimize_string (.clka(pixel_clk_in), .addra(opt_addr), .douta(opt_bits));

// 6 digits
logic [12:0] digit_a_addr,
             digit_b_addr,
             digit_c_addr,
             digit_d_addr,
             digit_e_addr,
             digit_f_addr;

logic [7:0] digit_a_image_bits,
            digit_b_image_bits,
            digit_c_image_bits,
            digit_d_image_bits,
            digit_e_image_bits,
            digit_f_image_bits;

digits_reduced_image_rom digit_a (.clka(pixel_clk_in), .addra(digit_a_addr), .douta(digit_a_image_bits));
digits_reduced_image_rom digit_b (.clka(pixel_clk_in), .addra(digit_b_addr), .douta(digit_b_image_bits));
digits_reduced_image_rom digit_c (.clka(pixel_clk_in), .addra(digit_c_addr), .douta(digit_c_image_bits));
digits_reduced_image_rom digit_d (.clka(pixel_clk_in), .addra(digit_d_addr), .douta(digit_d_image_bits));
digits_reduced_image_rom digit_e (.clka(pixel_clk_in), .addra(digit_e_addr), .douta(digit_e_image_bits));
digits_reduced_image_rom digit_f (.clka(pixel_clk_in), .addra(digit_f_addr), .douta(digit_f_image_bits));

always_comb begin
    case(optimal_time)
        // for each digit (2 digits -- tenths, ones place -- find proper address)
        6'd0 : begin
            digit_a_addr = 13'd0;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd10 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd576;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd20 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd1152;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd30 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd1728;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd40 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd2304;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd50 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd2880;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        6'd60 : begin
            digit_a_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH) + 13'd3456;
            digit_b_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd10)>>1)*DIGIT_WIDTH);
        end
        default : begin
            digit_a_addr = 13'd0;
            digit_b_addr = 13'd0;
        end
    endcase

    case(optimal_altitude)
        // for each digit (4 digits -- thousandths, hundredths, tenths, ones place -- find proper address)
        11'd0 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = 13'd0;
            digit_e_addr = 13'd0;
            digit_f_addr = 13'd0;
        end
        11'd250 : begin
```

```
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1152; //200
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd300 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd350 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1728; //300
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd400 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd450 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2304; //400
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd500 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd550 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //500
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd600 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd650 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd3456; //600
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd700 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4032; //700
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd750 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 15'd4032; //700
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd800 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd850 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4608; //800
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd900 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        end
        11'd950 : begin
            digit_c_addr = 13'd0;
            digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd5184; //900
            digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
```

```
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1000 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1050 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1100 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //100
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1150 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //100
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1200 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1152; //200
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1250 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1152; //200
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1300 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1728; //300
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1350 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1728; //300
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1400 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2304; //400
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1450 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2304; //400
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1500 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //500
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1550 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //500
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1600 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd3456; //600
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1650 : begin
        digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
        digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd3456; //600
        digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
        digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
    end
    11'd1700 : begin
```

```
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4032; //700
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd1750 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 15'd4032; //700
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd1800 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4608; //800
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd1850 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd4608; //800
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd1900 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd5184; //900
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd1950 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd576; //1000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd5184; //900
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd2880; //50
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            11'd2000 : begin
                digit_c_addr = ((hcount_in - 11'd10)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH) + 13'd1152; //2000
                digit_d_addr = ((hcount_in - 11'd58)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
                digit_e_addr = ((hcount_in - 11'd106)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
                digit_f_addr = ((hcount_in - 11'd154)>>1) + (((vcount_in - 10'd68)>>1)*DIGIT_WIDTH); //0
            end
            default : begin
                digit_c_addr = 13'd0;
                digit_d_addr = 13'd0;
                digit_e_addr = 13'd0;
                digit_f_addr = 13'd0;
            end
        endcase
    end

    always_ff @ (posedge pixel_clk_in) begin
        // logic for displaying pixels
        if ((hcount_in <= 1023) && (vcount_in <= 767)) begin
            if ((hcount_in >= 11'd212) && (hcount_in <= 11'd467) && (vcount_in >= 10'd10) && (vcount_in <= 10'd115)) begin
                //optimized string
                pixel_out <= {opt_bits[7:4], opt_bits[7:4], opt_bits[7:4]};
            end else if ((hcount_in >= 11'd106) && (hcount_in <= 11'd153) && (vcount_in >= 10'd10) && (vcount_in <= 10'd57)) begin
                // digit a
                pixel_out <= {digit_a_image_bits[7:4], digit_a_image_bits[7:4], digit_a_image_bits[7:4]};
            end else if ((hcount_in >= 11'd154) && (hcount_in <= 11'd201) && (vcount_in >= 10'd10) && (vcount_in <= 10'd57)) begin
                // digit b
                pixel_out <= {digit_b_image_bits[7:4], digit_b_image_bits[7:4], digit_b_image_bits[7:4]};
            end else if ((hcount_in >= 11'd10) && (hcount_in <= 11'd57) && (vcount_in >= 10'd68) && (vcount_in <= 10'd115)) begin
                // digit c
                pixel_out <= {digit_c_image_bits[7:4], digit_c_image_bits[7:4], digit_c_image_bits[7:4]};
            end else if ((hcount_in >= 11'd58) && (hcount_in <= 11'd105) && (vcount_in >= 10'd68) && (vcount_in <= 10'd115)) begin
                // digit d
                pixel_out <= {digit_d_image_bits[7:4], digit_d_image_bits[7:4], digit_d_image_bits[7:4]};
            end else if ((hcount_in >= 11'd106) && (hcount_in <= 11'd153) && (vcount_in >= 10'd68) && (vcount_in <= 10'd115)) begin
                // digit e
                pixel_out <= {digit_e_image_bits[7:4], digit_e_image_bits[7:4], digit_e_image_bits[7:4]};
            end else if ((hcount_in >= 11'd154) && (hcount_in <= 11'd201) && (vcount_in >= 10'd68) && (vcount_in <= 10'd115)) begin
                // digit f
                pixel_out <= {digit_f_image_bits[7:4], digit_f_image_bits[7:4], digit_f_image_bits[7:4]};
            end else begin
                // fill in with color map in circular region around center
                pixel_out <= ((((hcount_in - 512)**2) + ((vcount_in - 384)**2)) < (384**2)) ? rgb_in : 12'b0;
            end
        end else begin
            pixel_out <= 0; // outside of display parameters, assign black
        end
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module quad_scaler(
                   input  wire         clk_in,
                   input  wire         start_in,
                   input  wire  [3:0]  alt_in,
                   input  wire  [2:0]  time_in,
                   input  wire  [13:0] val_in,
                   input  wire  [3:0]  weighting_in,

                   output logic        done_out
                   output logic [3:0]  alt_out,
                   output logic [2:0]  time_out,
                   output logic [13:0] val_out,
                  );

    // make val_in and weighting (positive) signed
    logic signed [14:0] signed_val_in;
    assign signed_val_in = {1'b0, val_in};

    logic signed [4:0] signed_weighting;
    assign signed_weighting = {1'b0, weighting_in};

    // set up stages for multi-cycle computation
    logic signed [14:0] val_stage1, val_stage2, val_stage3;
    logic signed [4:0] weight_stage1, weight_stage2;
    logic start_1, start_2;
    logic [3:0] alt_stage1, alt_stage2, alt_stage3;
    logic [2:0] time_stage1, time_stage2, time_stage3;
    logic signed [14:0] stage1;
    logic signed [29:0] stage2;
    logic signed [29:0] stage3;
    logic signed [14:0] signed_val_out;
    assign signed_val_out = (stage3 >>> 18) + val_stage3;

    always @(posedge clk_in) begin
        val_stage1 <= signed_val_in;
        weight_stage1 <= signed_weighting;
        // x - 2^14
        stage1 <= signed_val_in - 15'b100_0000_0000_0000;
        start_1 <= start_in;
        alt_stage1 <= alt_in;
        time_stage1 <= time_in;

        val_stage2 <= val_stage1;
        weight_stage2 <= weight_stage1;
        // x * (x - 2^14)
        stage2 <= val_stage1 * stage1;
        start_2 <= start_1;
        alt_stage2 <= alt_stage1;
        time_stage2 <= time_stage1;

        val_stage3 <= val_stage2;
        // w * x * (x - 2^14)
        stage3 <= stage2 * weight_stage2;
        start_3 <= start_2;
        alt_stage3 <= alt_stage2;
        time_stage3 <= time_stage2;
```

```
        // x + (w * x * (x - 2^14))/2^18
        val_out <= signed_val_out[13:0];
        done_out <= start_3;
        alt_out <= alt_stage3;
        time_out <= time_stage3;
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none

module ram_initialize(
                      input  wire         clk_in,
                      input  wire         start,

                      output logic [14:0] addr_in,
                      output logic [32:0] data_in,
                      output logic        wea,
                      output logic        done,
                      output logic        ongoing
                     );

    always_ff @(posedge clk_in) begin
        if (start) begin
            // start writing into BRAM
            wea <= 1'b1;
            ongoing <= 1'b1;
            done <= 1'b0;
            addr_in <= 15'b0;
        end else if (ongoing) begin
            if (addr_in < 15'd30302) begin
                // increment address to write another value into memory
                addr_in <= addr_in + 1'b1;
            end else begin
                // reached end of data
                done <= 1'b1;
                ongoing <= 1'b0;
            end
        end else if (done) begin
            // done is a one-cycle pulse
            done <= 1'b0;
        end
    end

    always_comb begin
        case(addr_in)
            15'd0: data_in = 33'b100011101011001000000001111110110;
            15'd1: data_in = 33'b101100001011000010000001111111000;
            15'd2: data_in = 33'b100101010101100000000001111101100;
            15'd3: data_in = 33'b011100110101100000000001111011001;
            15'd4: data_in = 33'b010100000101100000000001111010001;
            15'd5: data_in = 33'b001011100101100000000001111011000;
            15'd6: data_in = 33'b000001011010101111000001111100101;
            15'd7: data_in = 33'b101000110101101000000010000111001;
            15'd8: data_in = 33'b100000010101100110000010000110111;
            15'd9: data_in = 33'b011000000101100110000010000110101;
            15'd10: data_in = 33'b001111100101100110000010000111011;
            // and so on for 30,000 lines
```

```
            default: data_in = 33'd0;
        endcase
    end
endmodule

`default_nettype wire
```

```
`timescale 1ns / 1ps
`default_nettype none
// XVGA Module from Lab 3 for control signals

module xvga(
            input  wire         vclock_in,

            output logic [10:0] hcount_out,
            output logic [9:0]  vcount_out,
            output logic        vsync_out, hsync_out, blank_out
           );

    parameter DISPLAY_WIDTH = 1024;
    parameter DISPLAY_HEIGHT = 768;
    parameter H_FP = 24;
    parameter H_SYNC_PULSE = 136;
    parameter H_BP = 160;
    parameter V_FP = 3;
    parameter V_SYNC_PULSE = 6;
    parameter V_BP = 29;

    logic hblank,vblank;
    logic hsyncon,hsyncoff,hreset,hblankon;
    assign hblankon = (hcount_out == (DISPLAY_WIDTH-1));
    assign hsyncon = (hcount_out == (DISPLAY_WIDTH+H_FP-1));
    assign hsyncoff = (hcount_out == (DISPLAY_WIDTH+H_FP+H_SYNC_PULSE-1));
    assign hreset = (hcount_out == (DISPLAY_WIDTH+H_FP+H_SYNC_PULSE+H_BP-1));

    logic vsyncon,vsyncoff,vreset,vblankon;
    assign vblankon = hreset && (vcount_out == (DISPLAY_HEIGHT-1));
    assign vsyncon = hreset && (vcount_out == (DISPLAY_HEIGHT+V_FP-1));
    assign vsyncoff = hreset && (vcount_out == (DISPLAY_HEIGHT+V_FP+V_SYNC_PULSE-1));
    assign vreset = hreset && (vcount_out == (DISPLAY_HEIGHT+V_FP+V_SYNC_PULSE+V_BP-1));
    logic next_hblank, next_vblank;
    assign next_hblank = hreset ? 0 : hblankon ? 1 : hblank;
    assign next_vblank = vreset ? 0 : vblankon ? 1 : vblank;
    always_ff @(posedge vclock_in) begin
        hcount_out <= hreset ? 0 : hcount_out + 1;
        hblank <= next_hblank;
        hsync_out <= hsyncon ? 0 : hsyncoff ? 1 : hsync_out;

        vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
        vblank <= next_vblank;
        vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;

        blank_out <= next_vblank | (next_hblank & ~hreset);
    end
endmodule

`default_nettype wire
```