Nonogrammer
a nonogram platform for fpga

Adrianna Wojtyna, Joules Ferguson

December 2021

Contents

1 Introduction

A nonogram is a picture logic puzzle that consists of a grid with a series of numbers on the top and one of its sides. The goal of the game is to fill in the fields in the grid in a way that the sequence of colored blocks correspond with the sequence of numbers on the edges of the grid. Throughout the paper we are going to refer to the array number associated with a given row or column as a constraint. In our project, we implemented an iterative algorithm to automatically solve 10x10 version of the puzzle. In addition, our system allows the user to play the game manually and solve the puzzle by using the buttons on the FPGA. Our system has 4 nonograms stored on the FPGA that that user can select to be displayed and either solved manually or automatically. The whole platform is accompanied by UI that facilitate interaction with the system and notifies the user about the sate they are selecting and the action they are to perform. In addition, user can take a photo of either a drawing or anything they like and turn it into a nonogram that they can then solve manually - the size of this nonogram is 30x40.

2 Top Level FSM (Adrianna)

Top Level FSM is the module that encompasses all the functionalities of our projects and is responsible for switching and triggering different states based on user's input and also progress of algorithms.

Here is a list of modules that are included inside the $top - level - FSM$ module:

- top-level-solver (iterative solver interface)
- return-UI (FSM for UI)
- manual-disp-10-10 (display for manual solving 10x10 puzzles)
- manual-disp-30-40 (display for manual solving 30x40 puzzles)
- solve-disp (display solution 10x10 nonogram)
- clk-wiz
- hex8-display
- debounce
- xvga
- $blk - mem - gen - 0$ (16x76800 simple dual port RAM for storing camera frame)

We used clock wizard from lab 3 to change default 100MHz clock into 65 MHz clock due to the requirements of 1024x768 xvga module. We clocked all other modules using 65MHz clock to avoid timing errors. Mainly for debugging purposes, we used hex8-display module in our top level FSM in order to track the state of state machines within the module. For that purpose we used code from lab 3 and 4. Values on the hex display correspond to (from right to left): state-button, state-UI, state-pixel, state, state-FSM

In addition, we used debounce implementation from past labs to debounce all the buttons that then were used by the user to select the state of FSM.

For display functionality, we also used XVGA module that we used in lab3. Top level assigns sw[0] as the reset switch that is the passed to all the module in our project. It also assigns sw[15:14] that allows the user to select the index of the puzzle they want to solve automatically or manually.

2.1 Pixel selection

One of the most important functions of the FSM in terms of user interaction with our system is deciding, which pixel is going to be outputted to vga_r, vga_g and vga_b. This is done by a piece of combinational logic in top level FSM that, based on the value of state-pixel and state-FSM, assigns value to the output pixel that the user sees on the screen. Since all the modules that are responsible for outputting pixel values continuously output some value on their output register, combinational logic resolves that issue. Since pixel selection is based on state-pixel value, below, we present a list of state-pixel values that are possible. Value of state-pixel register changes depending on the state-FSM and the progress within each of the states of FSM, which consist of small, internal FSMs

- GET CAMERA OUTPUT - user can see what the camera outputs but in black-and-white
- DISPLAY MANUAL 30x40 - user sees 30x40 puzzle, based on generated constraints from a photo
- DISPLAY MANUAL 10x10 - user sees the constraints of 10x10 puzzle of their choice and are able to solve it manually with buttons
- DISPLAY OPTIONS 10x10 - display the output of solv-disp module that either shows the solved nonogram with constraints, just constraints or solved puzzle with constraints.
- DISPLAY DIGI PHOTO - displays photo of an original size, but the photo comes from 0-1 encoding of the original picture that was generated once the frame was being saved to the frame buffer
- DISPLAY RESCALED PHOTO - displays the rescaled image (30x40) that comes from a translation from 0-1 encoding of 30x40 photo into RGB values that are then displayed in the monitor
- else - display blue

2.2 FSM states

There are 5 main states top level state machine can be at. States are changed by both a user, using sw[2:1] and the progress of the nonogram generator algorithm. Users presses center button to select a given states. Changing valeu sw[2:1] only changes the UI (pixel-state), but does not really change the state of the top level FSM (state-FSM)

- IDLE - state that the system is on reset (sw[2:1] = 0)
- MANUAL STATE - user can access hardcoded puzzles and solve them by hand (sw[2:1] = 01)
- SOLVER STATE - user can select from hardcoded nonograms and trigger the solver to complete them automatically (sw[2:1] = 10)
- GENERATE STATE - user can generate 30x40 automatically from a photo (sw[2:1] = 11)
    1. CAPTURE PHOTO - user can press a button to take a photo that is going to be rescaled and turned into a puzzle (sw[2:1] = 11 then center)
    2. RUN GENERATOR - user triggers a generator to turn a photo into a puzzle (sw[2:1] = 11 then center then left)
- DISPLAY MANUAL 30 40 - after RUN GENERATOR is done, state switches from RUN GENERATOR to this state automatically

1. DISPLAY EMPTY - user can only see the constraints around the grid and solver the puzzle manually (sw[7] = 0)
2. DISPLAY COMPLETE - user can still solve the nonogram that they generated manually, but now then can also he solution that comes form a photo using blue cells (sw[7] = 1 if user wants to see the optional solution)

3 Top Level Solver(Adrianna)

Top level solver serves as an interface with other modules in the whole system and interacts directly with the top level FSM module. The module decides whether the assignment registry or iterative solver should be started based on the input received from the top level FSM. Module both returns the constraints if $get-constraints$ input is high or it returns a solution from a solver if $get-solution$ register is high. Top level is responsible for returning correct constraints for a given puzzle one per clock cycle or translated solution fo a requested nonogram from a translator module in parallel, (ten 10-bit rows in one clock cycle).

3.1 Assignments registry

The main purpose of the module is to store the encoding of puzzles that the system will be able to solve automatically or display to the user for the manual solution. Assignment registry is responsible for returning a set of constraints when provided the index of the puzzle the user wants to either solve automatically or manually or just want to see the set of constraints before making a decision. Constraints themselves are stored in 20x 80 single port ROM, with each entry corresponding to one constraint. That means, one nonogram takes 20 consecutive entries and entries are returned one per clock cycle and outputted outside the assignment registry module.

Constraints to nonogram of size 10x10 are encoded by 20 bits long array, each number in the constraint encoded by 4 bits. The choice of the encoding is motivated by the invariant of the nonogram format. For 10x10 nonogram, the most numbers within the constraint is 5 (all 1's) and the biggest number of consecutive colored blocks is 10, which is 4 bit long number. Since we are dealing with 10 by 10 puzzles, for each puzzle we are storing 20 constraints, first rows from top to bottom and the constraints for columns for right to left. The puzzle constraints are returned one constraint per clock cycle, so it takes 20 clock cycles to return the set up for the whole puzzle.

3.2 Iterative Solver

Iterative solver module directly interfaces with $top-level-solver$ which provides at initialization of the solver, provided the set of constraints for a given nonogram, one constraints per clock cycle. After the solver received all the constraints from the top level, the algorithm starts. We based our implemetnaiton of the algorithm on the one provided on $https://rosettacode.org/wiki/Nonogram_solver$, but modified it to be implantable in the FPGA with limited memory, by removing recursive parts of the original algorithm.

Architecture of the solver can be treated as a large FSM that triggers small FSM that are responsible for consecutive stages of the algorithm. Main stages of the top level FSM for the solver include:

- Save constraints state
- Generate row and column permutations based on constraints state
- Allowable sequence construction state
- While loop state

- Output state

Each of major states in the FSM was responsible for triggering minor FSMs within each stage of the algorithm:

- Save constraints state
  - Save rows
  - Save columns
- Generate row and column permutations based on constraints state
  - Select constraint
  - Trigger generate rows module
- Allowable sequence construction state
  - Select row
  - Merge row constraints into one sequence - set state of each cell in each to either filled (01), empty (10) or unknown (00) by xo'ring all stored permutations for a given row
  - Save merged constraints into solution 2D array
- While loop state
  - Manage breaking the while loop - continue starting two for loops in the state as long as there are some changes made in the solution 2D array during the for loop progress
  - For loop range width - fix column states, remove permutations for each column that are not possible to be included in the solution, given the current state of the solution
  - For loop range height - fix row states, remove permutations for each row that are not possible to be included in the solution, given the current state of the solution
- Output state

The main idea behind the solver is to firstly generate all the possible states of the rows and columns that comply with the constraints that are given. This action is completed by Generate Rows module, as discussed below. Once the the generate row is ready to return all the states for a particular row, we store them in a 200x20 2D array with all states of a given row next to each other. In addition, in order to be able to get access to the states that correspond to a particular row and taking into account the fact that the number of states differs between constraints, we are also storing:

- Starting address from which the states of a given row begin in the 200x20 storage array
- Total number of permutations for a given row, as supplied by generate rows module

Since the algorithm for nonogram solver is pretty complex, we first started with its implementation in Python which we also attached in the appendix in the code section of our paper.

We decided to implement the whole functionality of the FSM as a single module because otherwise, it would required sending large amount of data between different modules. We thought this approach would not be beneficial because of multiple reasons - we would face memory constraints, synthesis time would increase and we would encounter even more bugs connected with being off by a clock cycle when sending data between modules.It resulted in a very tedious debugging process, however, it still turned out to be easier than our modularized implementation that we started with.

Out of what Adrianna was working on, according to her, this was the most challenging part of the project, and took significantly more time than developing interfacing with camera or developing UI, resizing the images or generating constraints to create a new nonogram.

3.3 Translator

As discussed in the Iterative Solver part of the paper, the module returns the solution in the from of ten 20-bit long arrays, with each cell in the nonogram being encoded by 2 bits. To facilitate processing in the display module of our system, 10x20 bit encoding of a solution is translated into 10x10 encoding of the state of the cells, with 1 corresponding to filled cell and 0 corresponding to empty cell. Translation happens by shifting a 2 bit window throughout each row that was received from the solver. Once all the rows in the original solution are processed, translation output is returned in a single clock cycle by returning 10 arrays of 10 bits in parallel .

3.4 Generate Rows

Generate rows module can be treated as a submodule of iterative solver, since iterative module is the only module that intervals with generate rows directly. The module interacted directly with create a row and get permutations modules which are going to be described in the following section. Generate rows module stores possible break lengths between Consecutive runs for each row and column constrained and store them in the 2D array. Since the number of permutation differ between different rows and column constraints, we are also storing the starting address for permutations corresponding to a given row/column. It also stores intermediate generated rows it that is collecting throughout the process in the 2D array (width - 20 bits, height - 200), as well as starting address of generated states, as in the permutations case . The height of the array being 200 units comes from our calculations of the maximum number of row states that we would be able to obtain from the worst scenario of assignments for a nonogram.

3.4.1 Get Permutations

Get Permutation module interacts directly with the generate rows module. The purpose of the module is to return the number of permutations given the number of filled blocks in a given row or column and the total number of white spaces that are left if cells are filled in following provided constraints. By permutations we mean possible setup breaks between consecutive runs, given the constraints assigned to a row. Here we present two possible "permutations" for a row that was assigned 9 as its constraint. We treat 1 as "filled and 0 as "empty", as per translator output

- 0111111111
- 1111111110

The number of possible breaks is at most 4 - in the worst case scenario, given rows is assigned 5 ones, so there are 4 breaks between consecutive 4 runs. In the worst case scenario, the length of the space between consecutive runs is 9 - in case a row is assigned to 1 as its single number in the constraint. Thus, we encode a single permutation using 16 bit number, each 4 bits corresponding to the length of the space between consecutive runs. Below we present an example representation of permutation for given states of rows (1 - filled cell, 0 -empty cell in state):

- state: 11111101111, permutation encoding 0000-0000-0000-0001
- state: 1110010001, permutation encoding 0000-0000-0010-0011 [first break of length 2, second break of length 3]

3.4.2 Create A Row

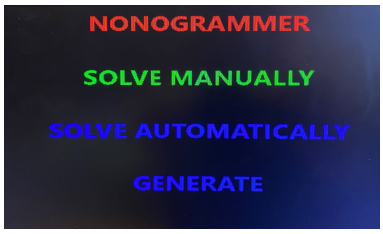
Create a Row module is a module that only directly interacts with generate rows module and it's responsible for returning encoding of a row/columns state of a nonogram given the number of

"runs", their length and then number of blocks that separate given runs. Module returns 20 bit encoding of a column/row, such that filled cell is equivalent "01" and empty cell is equivalent to "10"

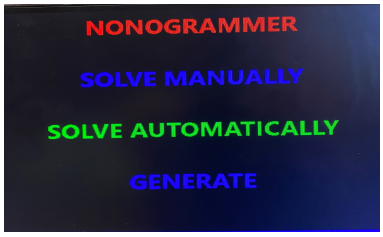
4 UI (Adrianna)

Return-UI module consists of a FSM of 4 states - IDLE, UI-SOLVER, UI-GENERATOR, UI-MANUAL. Module includes 4 different ROMs that provide access to 4 different coe files, abstracting 512 x 384 images that are then displayed on the screen

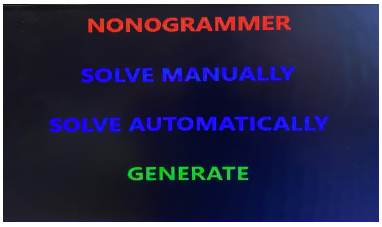
Based on the state that in provided by switches sw[2:1], output of one of 4 ROMs is chosen and then sent to the top level FSM that is the transferred to xvga module and displayed on the monitor. To limit memory use, we used only 4 colors on the UI display, which were encoded by 2 bits. We decide to remove the color mapping that is usually used, for example in the example display-module on the class website, and select the 12 bit pixel that is sent to the xvga module using a case statement that returns a correct color based on the 2-bit encoding saved in the coe file for each image. With this implementation we were able to save about 80 percent memory , with respect to using a regular color map



(a) Manual state



(b) Solver state



(c) Generate state

Figure 1: UI states

5 Nonogram generator (Adrianna)

5.1 Camera read

We based our camera read module on the module provided on the course website but we modified it such that the user is able to take a photo instead of just showing a video. If up button is not clicked, the camera is nor showing any output, but on button click, output of he camera is transferred to the frame buffer. Specifically, in our project we used OV7670 camera and a micro controller used to initialize it.Camera module has been modified with respect to the skeleton code provide on class website to include and additional state that initializes camera read on button press and which allows only writing to frame buffer once the user pressed the center button. Output pixel is the set to the top level FSM of the project. However, the process of storing the pixel outputted by the camera read is modified with respect to what was provided in the skeleton code from a class

website. Before RGB value is saved to the BRAM, we change the RGB value to either black. The following formula was used to perform black-scaling:

$$R * 4 + G * 2 + B * 2$$

If the value from the formula is greater than 5, we assigned a pixel to be white, otherwise we assigned it to black. The ise of the transformation of the RGB values comes from source (1) , but the value on the right hand side of the inequality comes from experimentation on the output of the camera

In addition to saving the RGB values in the BRAM, as per the skeleton code provide in class, we are also storing a digital abstraction of the whole photo in the (height - 240, width - 320) 2D array. If a pixel in a given coordinates was classified as black using the formula presented above, we stored 1 in a given bit in the 2D array. If the pixel was classified as white, we were storing 0 in the respective location. Dimensions of the BRAM that serves as the frame buffer (width - 16, height - 76800).

5.2 Image re-scaling and abstraction

After the image was stored in the frame buffer, the next step was rescaling the image from 320x240 to 40x30, to the sizes that are realistic for a regular nonogram. Rescaling was done using bilinear filtering in real time, i.e. while the new pixel was being saver to photo BRAM, new pixel was being saved to a 2D array (height - 30, width - 40) of registers, using the following formula $new - index = original - index/8$. That is, bilinear filtering was based on bit shifting current value of hcount and vcount by 3 and saving current value but black-scaled pixel to the 2D array.

(a) Original photo

(b) Rescaled photo

Figure 2: Rescaling process

5.3 Constraint generator

After rescaling an original photo to 40x30, we generate the constraints for a nonogram of the same dimensions whose solution corresponds to the photo that was taken by the user. Constraint generator is based on firstly going along each row and running a counter as long as a run of consecutive filled cell is present and then saving then number into the constraint abstraction. Each constrains is encoded by 120 bits, as in the worst case scenario we can have 40 consecutive blocks (6bits) and at most 20 numbers in the constraint (all 1's).

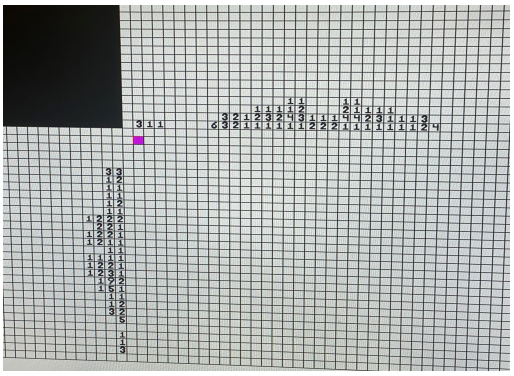


Figure 3: Generated puzzle

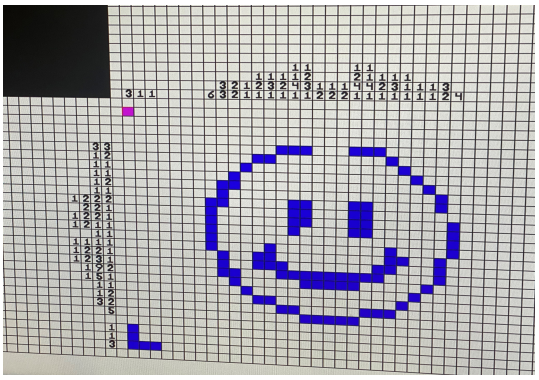


Figure 4: Generated puzzle with optional solution

Figure 5: Generator state

6 Solved Display (Joules)

The nonogram display system was built from the ground up in order to be as flexible and extensible as possible. This meant methods had to be developed for storing the state of the grid (i.e. which tiles are black or white), and the values of each constraint, as well as their position. This also meant methods to keep track of what part of the puzzle was being rendered had to be developed as well. For starters a framework was built up off of the hcount and vcount inputs. It was decided that 16x16 grid tiles would give us a good range of puzzle sizes, allowing 10x10 up to 30x40. Because of this if we bitshift our hcount and vcount by 4, effectively dividing by 16 and then rounding down. This gives us an effective indexing method, shown below that we can use to reference against arrays that we store our values in. An additional complexity arose with being able to access specific strings of bits from rows of our constraint arrays in order to access the values of each 4 bit constraint for our 10x10 puzzle. The elegant solution is to bitshift our index values back up by 4, allowing us to grab that index of the array, followed by the next 3 bits to get the 4 bits for that constraint. We have two pairs of indices to use for referencing our position in the display. X and Y, which start counting at the top left of the screen, which we use as the basis of addressing constraints in each row of our left constraints, and the constraints in each column of our top constraints. We also have x-shift and y-shift, which start at the top level of our grid. We use these to index our grid values as well as selecting which row or column constraint is being used.

In the above diagram, green squares represent our top constrains region, blue represents left constraints, and orange represents our display grid.
The information for our display is stored in 2 2d arrays, vals, and constraints, which hold the grid values and constraint values respectively. Vals is a 10x10 array, with 1s and 0s representing black and white tiles on the grid. Constraints is a 20x20 array, with each row representing a row of constraints for row constraints or a column for column constraints. Each row of constraints is comprised of 5 4 bit values representing each constraint. The layout of these arrays is shown below.

.5[h!]

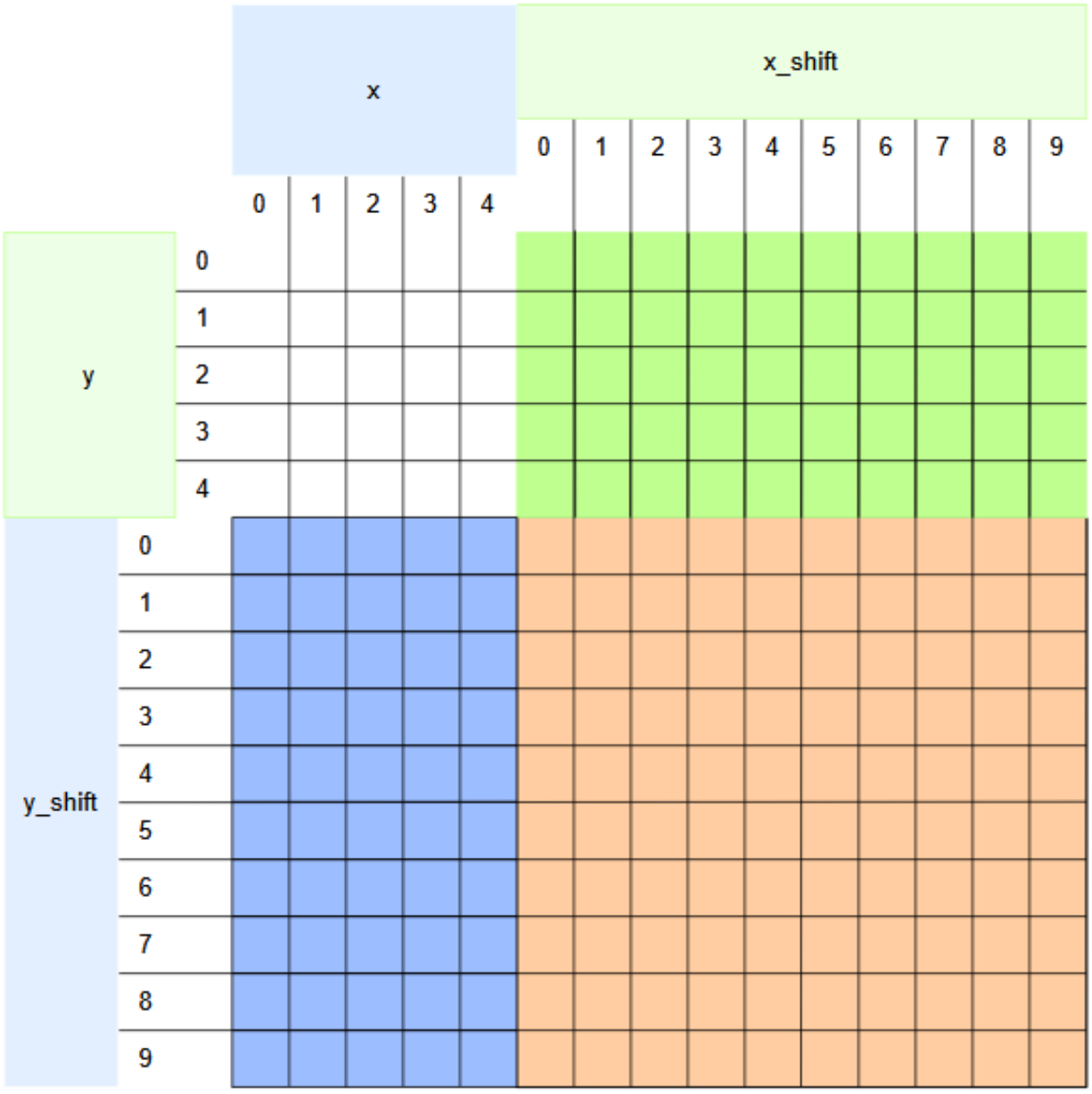


Figure 6: The grid addressing system used by all display modules

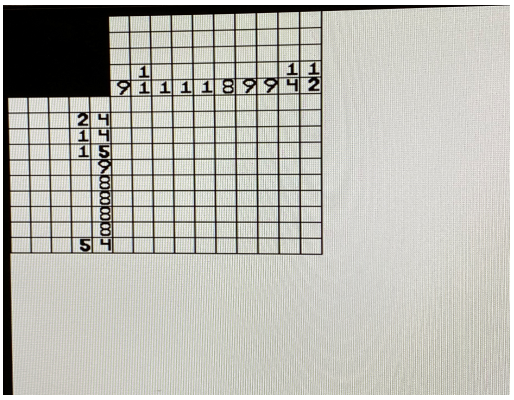


Figure 7: User requested constraints

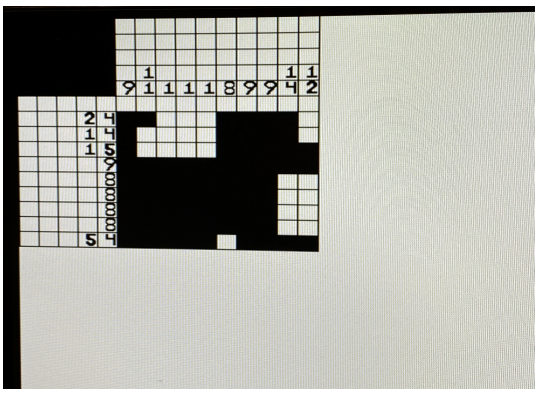
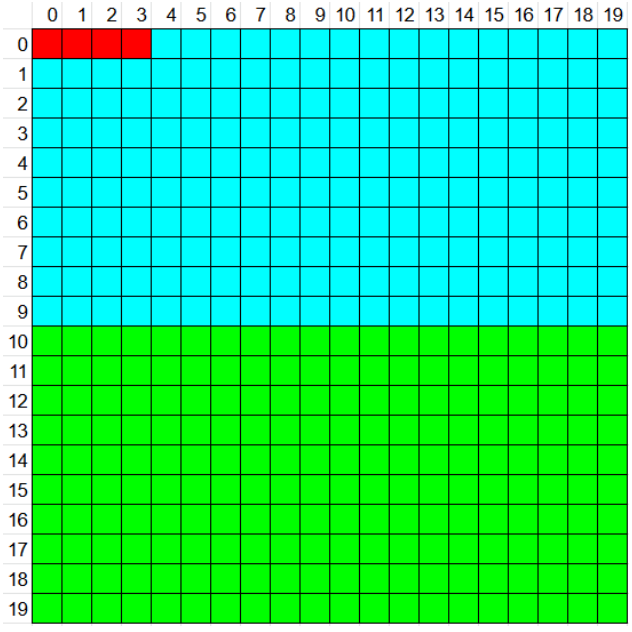


Figure 8: Solved nonogram [address 1]

Figure 9: Solving process with the solver



.5[h!]

Figure 10: the array used for storing our constraints,

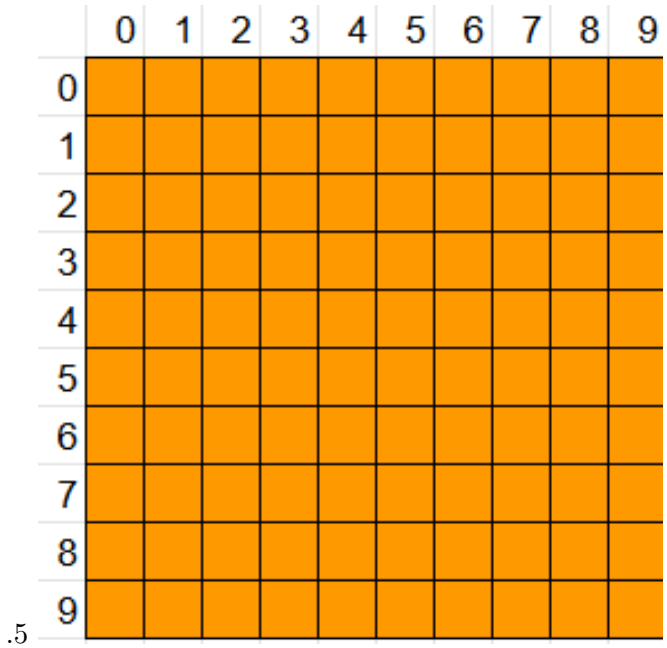


Figure 11: the indexing layout of our grid

One of the major elements of the display modules is handling the rendering of different components in a flexible manner. The display is handled by adding different pixel elements with black overlap (i.e. we have black pixels render over white pixels). We have pixel values held for the left constraints, top constraints, the grid overlay, and the tiles our solution is displayed upon, each with their own logic.

6.1 left-pixels

Left-pixels handles displaying the numerical value of our constraints on the left hand side of the nonogram. For this we assign a value to each area within one grid tile located in the left constraints region with our constraints array. We do this using x and y-shift, as these provide integer indexes for any point within those square regions, allowing us to reference which constraint value we should display in this tile. We have pixel outputs for each number we might need to display on hand, and using the value we pull from the constraints array we can select the correct pixel output in order to the display. However since our number blob modules require a position indicator, we need some way to provide a hcount and vcount to serve as the origin. Fortunately, we can bitshift our x and y values up by 4, effectively rounding down the position of our hcount and vcount to the nearest multiple of 16. Using these as our x and y coordinates for each number allows us to display numbers in their respective grid location.

6.2 top-pixels

The logic implemented for the top-pixels is nearly identical as that of the left pixels, with the only difference being that we use y and x-shift and we offset our x-shift index by 10 (i.e. constraints[x-shift+10][y+:3]). This is because we store all constraints in one 2d array, with left constraints stored first. (Due to the way constraints are delivered to our 10x10 modules, we actually use [x-shift-19] to index in reverse to display constraints correctly.

6.3 grid-overlay

The grid overlay is a set of pixels using ternary logic to render black pixels on any pixel who's hcount or vcount values are divisible by 16 (our gridsize). This is a simple yet important component of our display module as it gives the user a visual indication of the overall layout of the puzzle and constraints.

6.4 tile-pixels

Tile pixels handle displaying the solution of our nonogram. Like our constraints we use the x-shift and y-shift variables to index into our solution array, and using the value we get with that index, we set every pixel with those particular x-shift and y-shift values to black or white depending on the value of our values array.

All of these are combined together by inverting each value, adding them, and re-inverting them. In order to get the constraint and solution values, we send in two pulses, indicating when constraints and values are being sent in, and send in constraints on a row by row basis, and all values in on 10 10 long arrays. On reset we clear the values of each of these arrays so that we can have a clean start to import new values if we chose a different puzzle.

7 Manual Display 10x10 (Joules)

The manual 10x10 display is very similar to our solved display, with two main alterations: we have a cursor that we control and change values with, and the removal of an imported solution. With this module we simply import in a set of constraints similar to the solved display, and manually change values on the grid to try to solve it. The logic for our cursor is handled similarly to one of the tiles of our solution, except we can change the position of our cursor on the grid within all valid positions using directional buttons and using the x-shift and y-shift values of where our cursor is, we can use the center button to change the state of that particular grid position between white and black. We use ternary logic to restrict the movement of the cursor such that it can only move in valid directions.

8 Manual Display 30x40 (Joules)

The manual 30x40 display module is unique in that it is designed to allow the user to solve brand new nonograms imported from images. The manual 30x40 display is similar to our manual 10x10, with changed offsets and 2d array sizes to accommodate the increased size of the puzzle. Because we have more constraints our constraints array increases to a 120x70, with 6 bit encoding of constraint values, and a maximum of 20 constraints per row or column. We have the first 30 rows of this array stored as our row constraints, each 40 long, and the last 40 as our column constraints, each 30 long with 30 leading zeros. Due to the new unsymmetrical shape of our puzzle, x-shift and y-shift are now offset by 320 and 240 respectively, ensuring that we have our 0,0 located at the top left of the grid. Similar to our other display modules, we send in constraints 1 row at a time after receiving a start pulse. Similar to the automatic solver, we also send in the solution values, although due to the size of our solution values array, we send them in row by row. Because we want to verify the functionality of our nonogram generator, we store the scaled camera pixel values in a solution values array, which we can display in a similar method to our normal tile-pixels, allowing us to verify that the constraints and actual image align.

9 Process description/Design challenges

One of the main challenges in the implementation of the functionality was the number of moving parts that affected each other and had to work perfectly in sync in order to produce a correct result. One of the most frequent bugs we encountered which in the time was one of the most difficult mistakes to find, was being off by one clock cycle which was breaking the whole system. It was particularly challenging to debug since iterative solver module can be treated one big FSM that consist of 1000 lines of code. Running a testbench and analysing the waveform proved to be the most useful way to debug code, but, honestly, we spend hours staring at it to find that we forgot to subtract 1 in one place.

Another major roadblock in the implementation was developing a robust system by which we could display the constraints and solution to a nonogram. Initially it seemed like a fairly easy task, yet it was soon realized that the number of different elements that needed to be tracked, as well as different layers of pixels overlayed became a large source of bugs and issues with the way Verilog processes arrays and data. Because all of the display modules could not be tested using test benches, it required creating desired "test patterns" to run trial and error with in order to see if the logic was working as desired, and hopefully figure out where the errors were occurring. A common one would be to fill the constraints array with a sequence of repeating numbers, and the solution array with a specific pattern, and go about changing what was in those to find where there might be an incorrect addressing issue or possible faulty logic. Unfortunately this method did not always work, such as in the case of the 30x40 module which had the unfortunate issue of looping around constraints due to an issue with the addressing for constraints, causing it to read some constraints multiple times while skipping others. This meant that often times for debugging the display the best option was to incrementally change test patterns in different ways to find sources of errors.

A second challenge that arose in the display modules was the issue of submodularization. Because we are running a lot of wires between modules, by submodularizing we drastically increase the numbers of wires, as well as the number of potential errors. This had two main downsides, the first being that bitstream generation took longer with the submodularization of the display modules into several components (i.e. left, top, grid, tile) pixels were all separate modules. Additionally it left room for small bugs that would cause pixel outputs to become 1 bit to cause corruption of the displayed image. Although breaking things down into several modules made for cleaner code, the penalty of longer synthesis time and increased likelihood of bugs proved too strong a deterrent.

We cant highlight enough the importance of writing test-benches. At the final stage of development of our project, when the time of synthesis was very long, we started to write "fake modules" that would mimic the behavior of top level FSM but it wouldn't be using external elements or modules that we couldn't really simulate, for example debouncer or xvga. It helped us to find multiple bugs in out top level FSM implementation and saved us hours waiting for bitstream to be completed, Even though it might seem like a redundant work, since when writing "fake modules" we were not really developing anything new, it was better to spend 20 min writing the module and test-bench to discover the bug, instead of waiting hours to be able to upload new bit streams to the FPGA.

10 Summary

Overall this project was a very interesting and challenging problem. It required a lot of out of the box thinking as well as careful planning and testing to execute properly. Our group collaborated well in terms of effectively communicating what work was getting done, and in terms of managing getting things done when some aspects of the project hit roadblocks. We cleanly integrated different aspects of the project together, and tracked different versions of our code throughout testing using a shared github project. If we were to redo this project, there would be some changes we would

make. Mainly giving ourselves a slightly less complex set of main deliverable, as we quickly realized throughout the project that we had greatly underestimated the complexity of many aspects of the project, as well as different constraints we would encounter such as memory size and bitstream generation time. We would also likely work on better laying out how data transfer between modules worked before modules were written to save some time in the integration process.

11 Appendix: Source code link

`https://github.com/awojty/6.111_final_project`

12 Sources

https://www.rapidtables.com/convert/image/rgb-to-grayscale.html

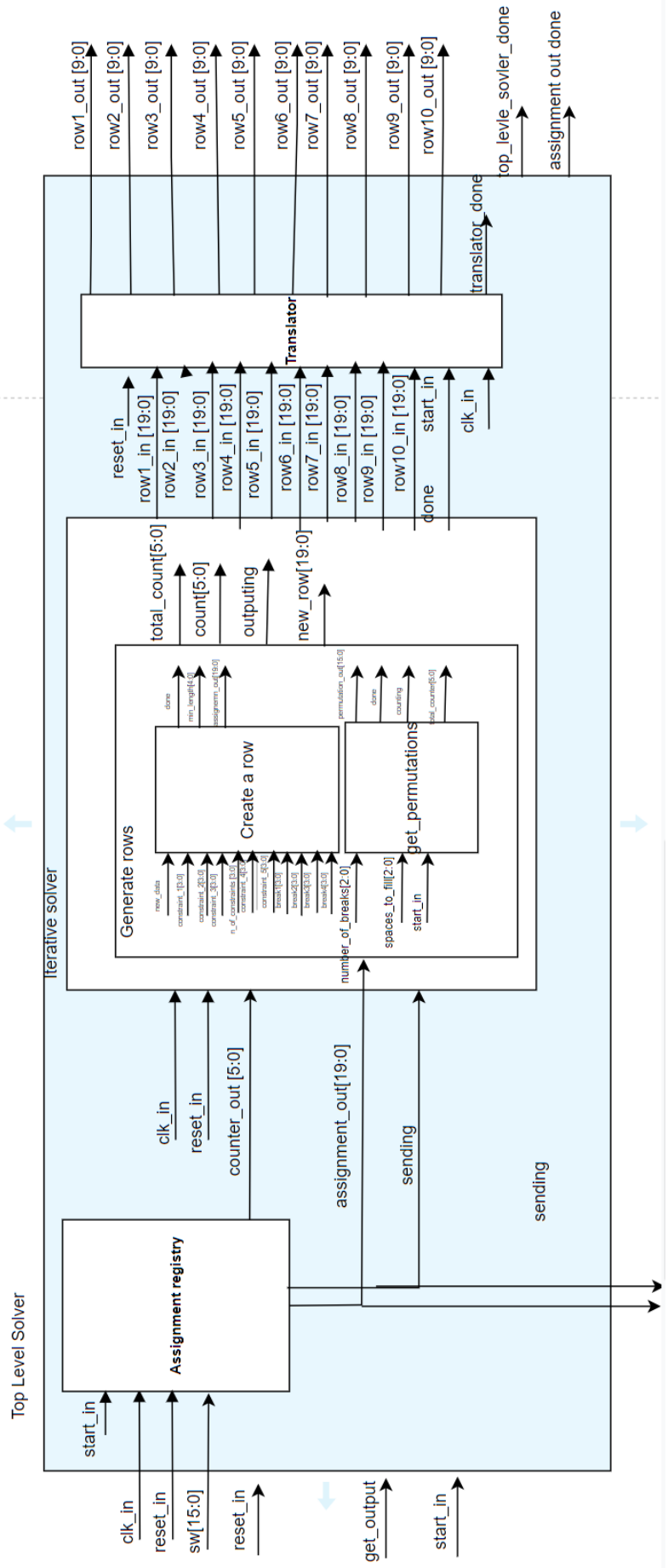


Figure 12: Top Level Solver diagram

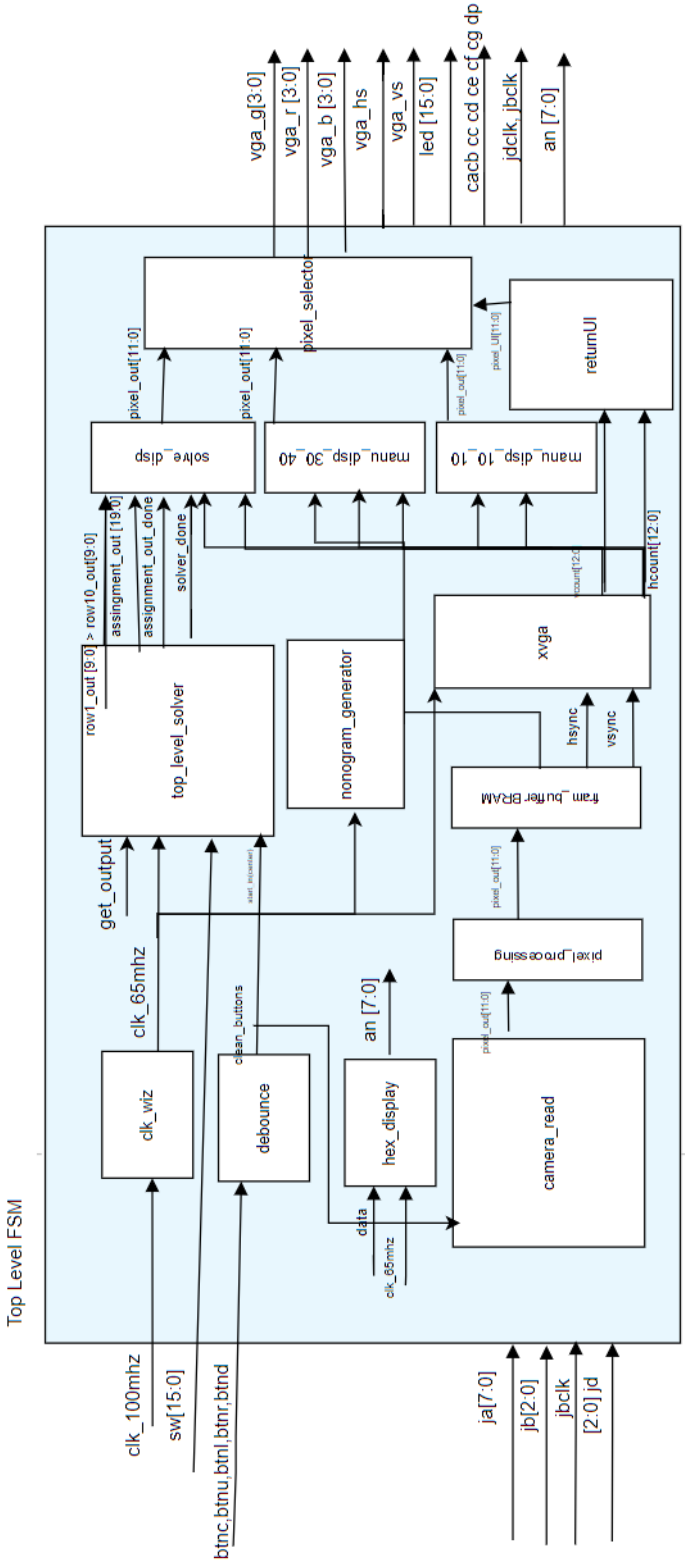


Figure 13: Top Level FSM diagram

Code

```
//-----------TESTBENCHES--------------


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
module assignment_registr_tb;
       // Inputs
       logic clk;
       logic confirm_in;
       logic rst_in;
       logic [7:0] row_number_in;
       logic [7:0] col_number_in;
       logic [15:0] address_in;

       //out
       logic [19:0] assignment_out;
       logic done;
       logic sending;

       // Instantiate the Unit Under Test (UUT)
       //one_hz_period changed to 4 cycles so simulations don't take forever.
        assignments_registry uut(
                    .clk_in(clk),
                    .start_in(confirm_in),
                    .reset_in(rst_in),


                    .address_in(address_in),
                   .assignment_out(assignment_out),
                   .done(done),
                   .sending(sending)
    );
        always #5 clk = !clk;

        initial begin
        // Initialize Inputs

        clk = 0;

       confirm_in=0;
       rst_in=0;
       row_number_in=0;
       col_number_in=0;
       address_in=0;

        #100;
        //get t_arm
        address_in=16'b0000_000_000_000_000;

        rst_in = 1;
        #10;
        rst_in=0;
        confirm_in=1;



      // Add stimulus here
   end

Endmodule

`timescale 1ns / 1ps
module constraint_generator_tb;
       // Inputs
        logic clk;
        logic reset_in;
        logic start_in;
        logic [39:0] image_in ;

        //output
        logic [119:0] constraints_out;
        logic done;

       // Instantiate the Unit Under Test (UUT)
       //one_hz_period changed to 4 cycles so simulations don't take forever.
        constraint_generator uut(
                    .clk_in(clk),
                    .reset_in(reset_in),
                    .start_in(start_in), //when asserte, start accumulating
                    .image_in(image_in),
                    .constraints_out(constraints_out),
                    .done(done) //320 by 240


    );
```

```
always #5 clk = !clk;

initial begin
// Initialize Inputs
clk = 0;
reset_in = 0;
start_in = 0;
image_in = 0;
image_in[1] = 0;
image_in[2] = 0;
image_in[3] = 0;
image_in[4] = 0;
image_in[5] = 0;
image_in[6] = 0;
image_in[7] = 0;
image_in[8] = 0;
image_in[9] = 0;
image_in[10] = 0;
image_in[11] = 0;
image_in[12] = 0;
image_in[13] = 0;
image_in[14] = 0;
image_in[15] = 0;
image_in[16] = 0;
image_in[17] = 0;
image_in[18] = 0;
image_in[19] = 0;
image_in[20] = 0;
image_in[21] = 0;
image_in[22] = 0;
image_in[23] = 0;
image_in[24] = 0;
image_in[25] = 0;
image_in[26] = 0;
image_in[27] = 0;
image_in[28] = 0;
image_in[29] = 0;
image_in[30] = 0;
image_in[31] = 0;
image_in[32] = 0;
image_in[33] = 0;
image_in[34] = 0;
image_in[35] = 0;
image_in[36] = 0;
image_in[37] = 0;
image_in[38] = 0;
image_in[39] = 0;
#10;
//get t_arm
reset_in = 1;

#10;
reset_in = 0;
start_in = 1;

image_in = 40'b0000000000000000000000000000000000000001;
#10;
start_in = 0;
image_in = 40'b0000000000000000000000000000000000000001;
#10;
image_in = 40'b0000000000000000000000000000000000000001;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000001;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
image_in = 40'b0000000000000000000000000000000000000000;
#10;
```

```
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b00000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;
        #10;
        image_in = 40'b0000000000000000000000000000000000000000;

         #10;
         start_in = 0;

//          #6000000;
//         reset_in = 0;
//         start_in = 1;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//start_in = 0;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
```

```
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;
//image_in = 40'b1111111111111111111111111111111111111111;
//#10;



//        image_in[0] = 40'b0000000000000000000000000000000000000000;
//        image_in[1] = 40'b0000000000000000000000000000000000000000;
//        image_in[2] = 40'b0000000000000001000000000110000000000000;
//        image_in[3] = 40'b0000000000001000000000000000010000000000;
//        image_in[4] = 40'b0000000000000000000000000000000100000000;
//        image_in[5] = 40'b0000000010000000000000000000000000000000;
//        image_in[6] = 40'b0000000000000000000000000000000000000000;
//        image_in[7] = 40'b0000000000000011000000000110000000010000;
//        image_in[8] = 40'b0000000000000111000000000111000000000000;
//        image_in[9] = 40'b0000100000001111100000001111100000000000;
//        image_in[10] = 40'b0000000000001111100000001111100000000100;
//        image_in[11] = 40'b0001000000001111100000001111100000000000;
//        image_in[12] = 40'b0000000000000111100000000111100000000010;
//        image_in[13] = 40'b0010000000000011000000000111000000000010;
//        image_in[14] = 40'b0010000000000000000000000000000000000000;
//        image_in[15] = 40'b0010000000000000000000000000000000000000;
//        image_in[16] = 40'b0010000000000000000000000000000000000000;
//        image_in[17] = 40'b0010000000000000000000000000000000000010;
//        image_in[18] = 40'b0000000001000000000000000000000110000010;
//        image_in[19] = 40'b0000000001100000000000000000000100000000;
//        image_in[20] = 40'b0001000000010000000000000000001000000100;
//        image_in[21] = 40'b0000100000001100000000000000110000000000;
//        image_in[22] = 40'b0000000000000111000000000011000000001000;
//        image_in[23] = 40'b0000000000000000111000011100000000010000;
//        image_in[24] = 40'b0000000000000000000000000000000000100000;
//        image_in[25] = 40'b0000000010000000000000000000000000000000;
//        image_in[26] = 40'b0000000001000000000000000000000100000000;
//        image_in[27] = 40'b0000000000010000000000000000010000000000;
//        image_in[28] = 40'b0000000000000011000000000110000000000000;
//        image_in[29] = 40'b0000000000000000000000000000000000000000;



   end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/27/2021 01:46:06 PM
// Design Name:
// Module Name: filter_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module filter_tb;
//input
logic clk;
logic reset_in;
logic start_in;
logic [319:0] photo_in;
```

```
logic done;
logic outputing;
logic [39:0] rescaled_out;
//output
    filter uut(
        .clk_in(clk),
         .reset_in(reset_in),
         .start_in(start_in), //when asserte, start accumulating
          .photo_in(photo_in)  ,
          .done(done), //320 by 240
          .outputing(outputing),
          .rescaled_out(rescaled_out)  //for the sake of testing // since thre are 10 fields and each field has two bits (3 states) > 10*2 = 20

        );


        always #5 clk = !clk;

        initial begin
        // Initialize Inputs

        clk = 0;

       reset_in=0;
       start_in=0;
       photo_in=0;
        #100;
        //get t_arm

        reset_in = 1;
        #10;
        reset_in = 0;
        start_in =1;
        photo_in <= 0;
         #10;
         start_in =0;
       photo_in <= 0;
        #10;
        photo_in <= 0;
         #10;
        photo_in <= 0;
         #10;
         photo_in <=0;
         #10;
         photo_in <=0;
        #10;
         photo_in <=0;
         #10;
         photo_in <=0;
         #10;
         photo_in <=0;
         #10;
         photo_in <=0;
         #10;
         photo_in <=0;
         #10;
         photo_in <=319'h11111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=319'h1111111;
         #10;
         photo_in <=311;
         #10;
         photo_in <=319'h1111111;
      // Add stimulus here
      end



endmodule

`timescale 1ns / 1ps
module generate_rows_tb;
```

```
        // Inputs
        logic clk;
        logic start_in;
        logic reset_in;
        logic [19:0] assignment;


        //out
        logic outputing;
        logic done;
        logic [19:0] new_row;
        logic [6:0] count; //totoal number of rows returned
        logic [6:0] total_count;

        // Instantiate the Unit Under Test (UUT)
        //one_hz_period changed to 4 cycles so simulations don't take forever.
         generate_rows uut(
                     .clk_in(clk),
                     .start_in(start_in),
                     .reset_in(reset_in),
                     .assignment(assignment),
                    .done(done),
                     .outputing(outputing), //asserted when the new_row is ready on the output (fully)
                     .new_row(new_row),
                     .count(count), //current index of the row in the set of that we are oging to return
                     .total_count(total_count) //returns the tola nbumber of optison returend for a given setging
                    );
         always #5 clk = !clk;

         initial begin
         // Initialize Inputs
          clk =0;
          start_in =0;
          reset_in = 0;
          assignment =0 ;

         #100;
          reset_in = 1;
          #10;
          reset_in = 0;
          assignment =20'b00000000000000110_0001; //24
          start_in =1;
          #10;
          start_in =0;

//          #10000;
//          assignment =20'b0000_0000_0000_0000_1000; //24
//          start_in =1;
//          #10;
//          start_in =0;

//          #10000;
//          assignment =20'b0000_0000_0000_0000_1000; //24
//          start_in =1;

//          #10;
//          start_in =0;
//          #10000;

//          assignment =20'b0000_0000_0000_0000_0000; //24
//          start_in =1;
          #10;
          start_in =0;


       // Add stimulus here
    end

endmodule

`timescale 1ns / 1ps
module get_permutations_tb;
        // Inputs
        logic clk;
        logic confirm_in;
        logic rst_in;

        logic [2:0] number_of_breaks;
        logic [2:0] space_to_fill_left;

        //out

        logic done;
        logic [11:0] permutation_out;
        logic [5:0] total_counter;

        // Instantiate the Unit Under Test (UUT)
        //one_hz_period changed to 4 cycles so simulations don't take forever.
         get_permutations uut(
                     .clk_in(clk),
```

```
                .start_in(confirm_in),
                .reset_in(rst_in),
                .number_of_breaks_in(number_of_breaks),
                .space_to_fill_in(space_to_fill_left),
                .permutation_out(permutation_out),
                .total_counter(total_counter),
               .done(done));
        always #5 clk = !clk;

        initial begin
        // Initialize Inputs
        clk = 0;
        confirm_in = 0;
        rst_in = 0;

        number_of_breaks = 0;
        space_to_fill_left =0;
        #100;
        //get t_arm
        rst_in = 1;
        #10;
        rst_in = 0;
        confirm_in=1; // free the button

        number_of_breaks = 3'd2;
        space_to_fill_left = 3'd5;

        #10;
        confirm_in=0;
        #200;



      // Add stimulus here
   end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
module indexing_tb;
       // Inputs
       logic clk;
       logic [4:0] index; //actual numebr of the nonogram selected by teh user
        logic btnc; //user confirms the slection of the nonogram

       logic rst_in;
       logic [10:0] array_in;


       //out

       logic done;
       logic  y_out;

       // Instantiate the Unit Under Test (UUT)
       //one_hz_period changed to 4 cycles so simulations don't take forever.
    fir31 uut(
          .clk_in(clk), .rst_in(rst_in),
          .index_in(index),
          .y_out(y_out),
          .array_in(array_in)
    );
        always #5 clk = !clk;

        initial begin
            // Initialize Inputs
            clk = 0;
            index = 0;
            btnc = 0;
            rst_in=0;
            array_in = 11'b0;

            #100;
            rst_in=1;
            #10;
            array_in = 10'b1010101011;
            rst_in=0;
            btnc = 1; //confirm
            index = 0; //nonogram at index 0
            #500;// 25 clocl cycles just inc ase - technically 23 shoudl be no
            btnc = 1; //confirm
            index = 1; //nonogram at index 0
            #10;
            btnc = 0; //free the button (as the user would)
            #500;//
```

```
      // Add stimulus here
   end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/16/2021 01:12:52 AM
// Module Name: iterative_solver_tb
//
//////////////////////////////////////////////////////////////////////////////////

module iterative_solver_chicken_tb;
       // Inputs
       logic clk;

       logic reset_in;
       logic [19:0] assignment_in;
       logic [5:0] index_in;
       logic [3:0] column_number_in;
       logic [3:0] row_number_in;
       logic start_sending_nonogram;


       //out
       logic solution_out;

       iterative_solver uut(
                    .clk_in(clk),

                    .reset_in(reset_in),
                    .index_in(index_in), //idnex of row/col beign send - max is 20 so 6 bits
                    .column_number_in(column_number_in), //grid size - max 10
                    .row_number_in(row_number_in), //grid size - max 10

                    .assignment_in(assignment_in), // array of cosntraitnrs in - max of 20 btis since 4 btis * 5 slots

                    .start_sending_nonogram(start_sending_nonogram), //if asserted to 1, im in the rpcoess of sendifg the puzzle
                     //only for the sake of testing wheterh we saved things correctly in the test bench
                    .solution_out(solution_out)
       );

       //one_hz_period changed to 4 cycles so simulations don't take forever.

        always #5 clk = !clk;

        initial begin
        // Initialize Inputs
       clk=0;

       reset_in=0;
       assignment_in=0;
       index_in=0;
       column_number_in=0;
       row_number_in=0;
       start_sending_nonogram=0;
       #100;
       reset_in = 1;
       #10;
       reset_in = 0;

       index_in<=0;

       start_sending_nonogram =1;
       column_number_in=10;
       row_number_in=10;

       assignment_in =20'b00000000000000000011;//
       index_in<=0;
       #10;


        assignment_in =20'b00000000000000010010;//
        index_in<=1;
       #10;

    assignment_in =20'b00000000000000100011;//
        index_in<=2;
       #10;

    assignment_in =20'b00000000000000100010;//
        index_in<=3;
       #10;

    assignment_in =20'b00000000000000000110;//
        index_in<=4;
       #10;
```

```
    assignment_in =20'b00000000000001010001;//
        index_in<=5;
       #10;

    assignment_in =20'b00000000000000000110;//
        index_in<=6;
       #10;

    assignment_in =20'b00000000000000000001;//
        index_in<=7;
       #10;

    assignment_in =20'b00000000000000000010;//
        index_in<=8;
       #10;

    assignment_in =20'b00000000000000000000;//
        index_in<=9;
       #10;

    assignment_in =20'b00000000000000100001;//
        index_in<=10;
       #10;

    assignment_in =20'b00000000000000010011;//
        index_in<=11;
       #10;

    assignment_in =20'b00000000000001010001;//
        index_in<=12;
       #10;

    assignment_in =20'b00000000000000010111;//
        index_in<=13;
       #10;

    assignment_in =20'b00000000000000000101;//
        index_in<=14;
       #10;

    assignment_in =20'b00000000000000000011;//
        index_in<=15;
       #10;

    assignment_in =20'b00000000000000000100;
        index_in<=16;
       #10;

    assignment_in =20'b00000000000000000011;
        index_in<=17;
       #10;

    assignment_in =20'b00000000000000000000;
        index_in<=18;
       #10;

    assignment_in =20'b00000000000000000000;
        index_in<=19;
       #10;


       #1000;


       end


endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Create Date: 11/16/2021 01:12:52 AM
// Module Name: iterative_solver_tb
//
//////////////////////////////////////////////////////////////////////////////////

module iterative_solver_tb;
       // Inputs
       logic clk;

       logic reset_in;
       logic [19:0] assignment_in;
       logic [5:0] index_in;
       logic [3:0] column_number_in;
       logic [3:0] row_number_in;
       logic start_sending_nonogram;


       //out
```

```
    logic solution_out;

    iterative_solver_wth_reset uut(
                .clk_in(clk),

                .reset_in(reset_in),
                .index_in(index_in), //idnex of row/col beign send - max is 20 so 6 bits
                .column_number_in(column_number_in), //grid size - max 10
                .row_number_in(row_number_in), //grid size - max 10

                .assignment_in(assignment_in), // array of cosntraitnrs in - max of 20 btis since 4 btis * 5 slots

                .start_sending_nonogram(start_sending_nonogram), //if asserted to 1, im in the rpcoess of sendifg the puzzle
                 //only for the sake of testing whetherh we saved things correctly in the test bench
                .solution_out(solution_out)
    );

    //one_hz_period changed to 4 cycles so simulations don't take forever.

     always #5 clk = !clk;

     initial begin
     // Initialize Inputs
    clk=0;

    reset_in=0;
    assignment_in=0;
    index_in=0;
    column_number_in=0;
    row_number_in=0;
    start_sending_nonogram=0;
    #100;
    reset_in = 1;
    #10;
    reset_in = 0;

    index_in<=0;

    start_sending_nonogram =1;
    column_number_in=10;
    row_number_in=10;



    //assignment_in =20'b00000000000000000011;//
    assignment_in =20'b0000000000000000_0000;
    index_in<=0;
    #10;


     //assignment_in =20'b00000000000000010010;//
     assignment_in =20'b0000000000000_0010_0100;//
     index_in<=1;
    #10;

assignment_in =20'b0000000000000_0001_0100;//
     index_in<=2;
    #10;

assignment_in =20'b0000000000000_0001_0101;//
     index_in<=3;
    #10;

assignment_in =20'b0000000000000000_1010;//
     index_in<=4;
    #10;

assignment_in =20'b0000000000000000_1000;//
     index_in<=5;
    #10;

assignment_in =20'b0000000000000000_1000;//
     index_in<=6;
    #10;

assignment_in =20'b0000_0000_0000_0000_1000;//
     index_in<=7;
    #10;

assignment_in =20'b0000000000000000_1000;//
     index_in<=8;
    #10;

assignment_in =20'b0000000000000_0101_0100;//
     index_in<=9;
    #10;

assignment_in =20'b0000000000000_0001_0010;//
     index_in<=10;
```

```
   #10;

assignment_in =20'b0000000000000_0001_0100;//

    index_in<=11;
   #10;

assignment_in =20'b00000000000000001001;//
    index_in<=12;
   #10;

assignment_in =20'b00000000000000001001;//
    index_in<=13;
   #10;

assignment_in =20'b00000000000000001000;//
    index_in<=14;
   #10;

assignment_in =20'b00000000000000000110;//
    index_in<=15;
   #10;

assignment_in =20'b00000000000000000110;
    index_in<=16;
   #10;

assignment_in =20'b0000_0000_0000_0000_0110;
    index_in<=17;
   #10;

assignment_in =20'b00000000000001100001;
    index_in<=18;
   #10;

assignment_in =20'b0000_0000_0000_0000_1001;
    index_in<=19;









    ////////////////////////////////////////////
   #40;
   start_sending_nonogram =0;


   #10000;

   //NEW_INPUT


   reset_in = 0;


   start_sending_nonogram =1;
   column_number_in=10;
   row_number_in=10;
   //assignment_in =20'b00000000000000000011;//
   assignment_in =20'b00000000000000000000;
   index_in<=0;
   #10;

    //assignment_in =20'b00000000000000010010;//
    assignment_in =20'b00000000000000001000;//
    index_in<=1;
   #10;

assignment_in =20'b00000000000000000001000;//
    index_in<=2;
   #10;

assignment_in =20'b00000000000000001000;//
    index_in<=3;
   #10;

assignment_in =20'b00000000000000001000;//
    index_in<=4;
   #10;

assignment_in =20'b00000000000000001000;//
    index_in<=5;
   #10;

assignment_in =20'b00000000000000001000;//
```

```
         index_in<=6;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=7;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=8;
        #10;

    assignment_in =20'b00000000000000000000;//
         index_in<=9;
        #10;

    assignment_in =20'b00000000000000000000;//
         index_in<=10;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=11;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=12;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=13;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=14;
        #10;

    assignment_in =20'b00000000000000001000;//
         index_in<=15;
        #10;

    assignment_in =20'b00000000000000001000;
         index_in<=16;
        #10;

    assignment_in =20'b00000000000000001000;
         index_in<=17;
        #10;

    assignment_in =20'b00000000000000001000;
         index_in<=18;
        #10;

    assignment_in =20'b00000000000000000000;
         index_in<=19;
        #40;
        start_sending_nonogram =0;



        end


endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////

module top_level_fresher_testing_tb;
            // Inputs
          logic clk;
          logic reset_in;

          logic [15:0] sw;
          logic btnc;
          logic btnr;
          logic btnl;
          logic btnd;
          logic btnu;
          logic [7:0] ja;
          logic [2:0] jb;
          logic [12:0] hcount;
          logic [12:0] vcount;
          //output
          logic [119:0] constraints_out;
          logic done;

        // Instantiate the Unit Under Test (UUT)
        //one_hz_period changed to 4 cycles so simulations don't take forever.
        top_level_fresher_testing uut(
                 .clk_100mhz(clk),
```

```
        .sw(sw),
        .btnc(btnc), .btnu(btnu), .btnl(btnl), .btnr(btnr), .btnd(btnd),
        .hcount(hcount), .vcount(vcount),
        .ja(ja), //pixel data from camera
        .jb(jb)); //other data from camera (including clock return)

always #5 clk = !clk;
initial begin

clk = 0;


sw = 0;
btnc = 0;
btnr = 0;
btnl = 0;
btnd  = 0;
btnu = 0 ;
#10;
sw[0] = 1;
#10;
sw[0] = 0;
#10;
sw[2:1] = 2'b11;
#10;
btnc <=1;
#10;
btnc <=0;
#10;
btnl <=1;
#10;
btnl <=0;

#50;
vcount = 240;
hcount=320;
#10;
hcount=336;
#10;
hcount=352;
#10;
hcount=368;
#10;
hcount=384;
#10;
hcount=400;
#10;
hcount=416;
#10;
hcount=432;
#10;
hcount=448;
#10;
hcount=464;
#10;
hcount=480;
#10;
hcount=496;
#10;
hcount=512;
#10;
hcount=528;
#10;
hcount=544;
#10;
hcount=560;
#10;
hcount=576;
#10;
hcount=592;
#10;
hcount=608;
#10;
hcount=624;
#10;
hcount=640;
#10
hcount=656;
#10
hcount=672;
#10;
hcount=688;
#10;
hcount=704;
#10;
hcount=720;
#10;
hcount=736;
#10;
hcount=752;
```

```
        #10;
        hcount=768;
        #10;
        hcount=784;
        #10;
        hcount=800;
        #10;
        hcount=816;
        #10;
        hcount=832;
        #10;
        hcount=848;
        #10;
        hcount=864;
        #10;
        hcount=880;
        #20;
        hcount=319;
        vcount=241;
        #10;
        vcount=257;
        #10;
        vcount=273;
        #10;
        vcount=289;
        #10;
        vcount=305;
        #10;
        vcount=321;
        #10;
        vcount=337;
        #10;
        vcount=353;
        #10;
        vcount=369;
        #10;
        vcount=385;

        #10;
        vcount=401;
        #10;
        vcount=417;
        #10;
        vcount=433;
        #10;
        vcount=449;
        #10;
        vcount=465;
        #10;
        vcount=481;
        #10;
        vcount=497;
        #10;
        vcount=513;
        #10;
        vcount=529;
        #10;
        vcount=545;

        #10;
        vcount=561;
        #10;
        vcount=577;
        #10;
        vcount=593;
        #10;
        vcount=609;
        #10;
        vcount=625;
        #10;
        vcount=641;
        #10;
        vcount=657;
        #10;
        vcount=673;
        #10;
        vcount=689;
        #10;
        vcount=705;
        end

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Create Date: 11/16/2021 01:12:52 AM
// Module Name: iterative_solver_tb
//
//////////////////////////////////////////////////////////////////////////////////
```

```
module top_level_nonogram_solver_tb;
	// Inputs
	logic clk;
	logic [15:0] sw;
	logic start_in;
	logic reset_in;
	//out
	logic [9:0] row1_out;
	logic [9:0] row2_out;
	logic [9:0] row3_out;
	logic [9:0] row4_out;
	logic [9:0] row5_out;
	 logic [9:0] row6_out;
	logic [9:0] row7_out;
	logic [9:0] row8_out;
	logic [9:0] row9_out;
	logic [9:0] row10_out;

	logic top_level_solver_done;

	top_level_solver uut(
			.clk_in(clk),
			.start_in(start_in), // assered when in the correct stata
			.sw(sw),
			.reset_in(reset_in),

			.row1_out(row1_out),
			.row2_out(row2_out),
			.row3_out(row3_out),
			.row4_out(row4_out),
			.row5_out(row5_out),
			.row6_out(row6_out),
			.row7_out(row7_out),
			.row8_out(row8_out),
			.row9_out(row9_out),
			.row10_out(row10_out),
			.top_level_solver_done(top_level_solver_done)
	);

	//one_hz_period changed to 4 cycles so simulations don't take forever.

	 always #5 clk = !clk;

	 initial begin
	 // Initialize Inputs
	clk=0;

	sw = 0;
	start_in = 0;
	reset_in=0;

	#100;
	reset_in = 1;
	#10;
	reset_in = 0;
	start_in  = 1;
	 sw = 0; // get nonogram at idnex 0
	 #10;


	end


endmodule



//-----------IMPLEMENTATION--------------
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Create Date: 11/16/2021 01:12:52 AM
//////////////////////////////////////////////////////////////////////////////////

module top_level_nonogram_solver_tb;
	// Inputs
	logic clk;
	logic [15:0] sw;
	logic start_in;
	logic reset_in;
	//out
	logic [9:0] row1_out;
	logic [9:0] row2_out;
	logic [9:0] row3_out;
	logic [9:0] row4_out;
	logic [9:0] row5_out;
	 logic [9:0] row6_out;
	logic [9:0] row7_out;
	logic [9:0] row8_out;
	logic [9:0] row9_out;
	logic [9:0] row10_out;
```

```
        logic top_level_solver_done;
        logic get_output;
        logic [19:0] assignment_out;
        logic assignment_out_done;
        logic sending_assignment;

        top_level_solver uut(
                    .clk_in(clk),
                    .start_in(start_in), // assered when in the correct stata
                    .sw(sw),
                    .reset_in(reset_in),
                    .get_output(get_output),
                    .assignment_out(assignment_out),
                    .assignment_out_done(assignment_out_done),
                    .sending_assignment(sending_assignment),

                    .row1_out(row1_out),
                    .row2_out(row2_out),
                    .row3_out(row3_out),
                    .row4_out(row4_out),
                    .row5_out(row5_out),
                    .row6_out(row6_out),
                    .row7_out(row7_out),
                    .row8_out(row8_out),
                    .row9_out(row9_out),
                    .row10_out(row10_out),
                    .top_level_solver_done(top_level_solver_done)
        );


//           input wire clk_in,
//    input wire start_in, // assered when in the correct stata
//    input wire reset_in,
//    input wire get_output,
//    input wire [15:0] sw,
//    output logic [19:0] assignment_out,
//    output logic [9:0] row1_out,
//    output logic [9:0] row2_out,
//    output logic [9:0] row3_out,
//    output logic [9:0] row4_out,
//    output logic [9:0] row5_out,
//    output logic [9:0] row6_out,
//    output logic [9:0] row7_out,
//    output logic [9:0] row8_out,
//    output logic [9:0] row9_out,
//    output logic [9:0] row10_out,
//    output logic top_level_solver_done,
//    output logic assignment_out_done

        //one_hz_period changed to 4 cycles so simulations don't take forever.

         always #5 clk = !clk;

         initial begin
         // Initialize Inputs
        clk=0;

        sw = 0;
        start_in = 0;
        reset_in=0;
        get_output = 0;

        #100;
        reset_in = 1;
        #10;
        reset_in = 0;
        //start_in  = 1;
        //get_output = 0;
         sw = 0; // get nonogram at idnex 0

         #20;
         start_in  = 0;
         sw[15:14]= 2'b10;

         #20;
         get_output = 1;

         #10;
         get_output = 0;
         #20;
         get_output = 0;

         #20000;

         start_in  = 1;

          #10;
          start_in  = 0;
```

```
         #40000;
         start_in  = 1;
         #10;
         start_in  = 0;

        //assignment_out_done


       end


endmodule

`timescale 1ns / 1ps
module translator_tb;
       // Inputs
        logic clk;
        logic reset_in;
        logic start_in;
        logic [19:0] row1;
        logic [19:0] row2;
        logic [19:0] row3;
        logic [19:0] row4;
        logic [19:0] row5;
        logic [19:0] row6;
        logic [19:0] row7;
        logic [19:0] row8;
        logic [19:0] row9;
        logic [19:0] row10;
        //output
        logic [9:0] row1_out;
        logic [9:0] row2_out;
        logic [9:0] row3_out;
        logic [9:0] row4_out;
        logic [9:0] row5_out;
        logic [9:0] row6_out;
        logic [9:0] row7_out;
        logic [9:0] row8_out;
        logic [9:0] row9_out;
        logic [9:0] row10_out;
        logic solution_out;

       // Instantiate the Unit Under Test (UUT)
       //one_hz_period changed to 4 cycles so simulations don't take forever.
        translator uut(
                    .clk_in(clk),
                    .reset_in(reset_in),
                    .start_in(start_in),
                    .row1(row1),
                    .row2(row2),
                    .row3(row3),
                    .row4(row4),
                    .row5(row5),
                    .row6(row6),
                    .row7(row7),
                    .row8(row8),
                    .row9(row9),
                    .row10(row10),
                    .row1_out(row1_out),
                    .row2_out(row2_out),
                    .row3_out(row3_out),
                    .row4_out(row4_out),
                    .row5_out(row5_out),
                    .row6_out(row6_out),
                    .row7_out(row7_out),
                    .row8_out(row8_out),
                    .row9_out(row9_out),
                    .row10_out(row10_out),

                    .solution_out(solution_out)

    );
        always #5 clk = !clk;

        initial begin
        // Initialize Inputs
        clk = 0;

        reset_in = 0;
        start_in = 0;
        row1 = 0;
        row2 = 0;
        row3 = 0;
        row4 = 0;
        row5 = 0;
        row6 = 0;
        row7 = 0;
        row8 = 0;
        row9 = 0;
        row10 = 0;
```

```
        #100;
        //get t_arm
        reset_in = 1;


        #10;
        reset_in = 0;
        start_in = 1;
        row1 = 20'b1010_1010_1010_1010_1010;
        row2 = 20'b1010_0101_0101_0101_1010;
        row3 = 20'b1010_0101_0101_0101_1010;
        row4 = 20'b1010_0101_0101_0101_1010;
        row5 = 20'b1010_0101_0101_0101_1010;
        row6 = 20'b1010_0101_0101_0101_1010;
        row7 = 20'b1010_0101_0101_0101_1010;
        row8 = 20'b1010_0101_0101_0101_1010;
        row9 = 20'b1010_0101_0101_0101_1010;
        row10 = 20'b1010_1010_1010_1010_101;
        #10;
        reset_in = 0;
        start_in = 0;
        #20000;
        //test if it outputs again the same thing
        start_in = 1;
        row1 = 20'b1010_1010_1010_1010_1010;
        row2 = 20'b1010_0101_0101_0101_1010;
        row3 = 20'b1010_0101_0101_0101_1010;
        row4 = 20'b1010_0101_0101_0101_1010;
        row5 = 20'b1010_0101_0101_0101_1010;
        row6 = 20'b1010_0101_0101_0101_1010;
        row7 = 20'b1010_0101_0101_0101_1010;
        row8 = 20'b1010_0101_0101_0101_1010;
        row9 = 20'b1010_0101_0101_0101_1010;
        row10 = 20'b1010_1010_1010_1010_101;

    end

endmodule


module elevens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_eleven_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

         end
    end
endmodule

module twelves_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twelve_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
```

```
            pixel_out <= 0;

        end
    end
endmodule

module thirteens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic  image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_thirteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule
module forteens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_forteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

      always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule

module fifteens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic  image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_fifteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule

module sixteens_pixels
```

```
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic  image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_sixteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module seventeens_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_seventeen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module eighteens_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_eighteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module nineteens_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
```

```
   logic  image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_nineteen_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module twenties_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_twenty_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module twentyones_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_twentyone_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module twentytwos_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic  image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_twentytwo_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin
```

```
            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule

module twentythrees_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentythree_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule

module twentyfours_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentyfour_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule

module twentyfives_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentyfive_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
    // use color map to create 4 bits R, 4 bits G, 4 bits B
       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
```

```
            pixel_out <= 0;

        end
    end
endmodule

module twentysixes_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentysix_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule

module twentysevens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentyseven_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule

module twentyeights_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_twentyeight_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));

       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule
```

```
module twentynines_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_twentynine_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module thirties_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirty_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module thirtyones_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtyone_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module thirtytwos_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
```

```
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtytwo_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module thirtythrees_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtythree_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module thirtyfours_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtyfour_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module thirtyfives_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic image_bits;
   // calculate rom address and read the location
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtyfive_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;
```

```
        end
    end
endmodule
module thirtysixes_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits;
    // calculate rom address and read the location
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_thirtysix_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule
module thirtysevens_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_thirtyseven_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule
module thirtyeights_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
    small_thirtyeight_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
       always_ff @ (posedge pixel_clk_in) begin
         if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

             if (image_bits == 1'b0) begin
                 pixel_out <= 12'hfff; // white
             end else begin
                 pixel_out <= 12'd0; // black
             end
        end else  begin
             pixel_out <= 0;

        end
    end
endmodule
module thirtynines_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);
    logic [15:0] image_addr;   // num of bits for 256*240 ROM
    logic image_bits;
    // calculate rom address and read the location
```

```
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_thirtynine_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
      always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module forties_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);
   logic [15:0] image_addr;   // num of bits for 256*240 ROM
   logic image_bits;
   assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   small_forty_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(image_bits));
     always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (image_bits == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule


`default_nettype none
module manual_disp_30_40(
                  input wire clock,
                  input wire reset,
                  input wire left,
                  input wire right,
                  input wire up,
                  input wire down,
                  input wire center,
                  input wire show_sol,
                  input wire start_sending_constraint,
                  input wire start_sending_photo,
                  input wire [39:0] photo_in,
                  input wire [119:0] constraint_vals,
                  input wire [12:0] hcount,
                  input wire [12:0] vcount,
                  input wire [15:0] switch,
                  output logic [11:0] pixel_out);

    logic receiving; //lets the system know if it is in receiving mode or not
    logic [6:0] count_num; //counts from 0-19 to keep track of indexing with clock cycles for the 80x80 constraints
    logic [6:0] count_num_photo;

    logic [10:0] address_x;
    logic [10:0] address_y;
    logic [10:0] address_x_shift; //coordinates of every 16 pixels,
    logic [10:0] address_y_shift;
    logic [9:0] x_in; //coordinates upscaled to give display coordinates of image blobs
    logic [9:0] y_in;
    logic [10:0] lower_x; //addressing for registers of values, use [lower +:3] to grab 4 bit val
    logic [10:0] lower_y;

    //initialize our different layers of pixels
    logic [11:0] left_pixels;
    logic [11:0] top_pixels;
    logic [11:0] grid_pixels;
    logic [11:0] tile_pixels;
    logic [5:0] top_val;
    logic [5:0] left_val;

    //initialize the pixel values of our numbers
    logic [11:0] one_pixels;
    logic [11:0] two_pixels;
    logic [11:0] three_pixels;
    logic [11:0] four_pixels;
    logic [11:0] five_pixels;
    logic [11:0] six_pixels;
```

```
    logic [11:0] seven_pixels;
    logic [11:0] eight_pixels;
    logic [11:0] nine_pixels;
    logic [11:0] nine_pixels;
    logic [11:0] ten_pixels;
    logic [11:0] eleven_pixels;
    logic [11:0] twelve_pixels;
    logic [11:0] thirteen_pixels;
    logic [11:0] fourteen_pixels;
    logic [11:0] fifteen_pixels;
    logic [11:0] sixteen_pixels;
    logic [11:0] seventeen_pixels;
    logic [11:0] eighteen_pixels;
    logic [11:0] nineteen_pixels;
    logic [11:0] twenty_pixels;
    logic [11:0] twentyone_pixels;
    logic [11:0] twentytwo_pixels;
    logic [11:0] twentythree_pixels;
    logic [11:0] twentyfour_pixels;
    logic [11:0] twentyfive_pixels;
    logic [11:0] twentysix_pixels;
    logic [11:0] twentyseven_pixels;
    logic [11:0] twentyeight_pixels;
    logic [11:0] twentynine_pixels;
    logic [11:0] thirty_pixels;
    logic [11:0] thirtyone_pixels;
    logic [11:0] thirtytwo_pixels;
    logic [11:0] thirtythree_pixels;
    logic [11:0] thirtyfour_pixels;
    logic [11:0] thirtyfive_pixels;
    logic [11:0] thirtysix_pixels;
    logic [11:0] thirtyseven_pixels;
    logic [11:0] thirtyeight_pixels;
    logic [11:0] thirtynine_pixels;
    logic [11:0] forty_pixels;


    logic [11:0] blank_pixels;

    //values we use to control our "cursor"
    reg [6:0] cursor_x;
    reg [6:0] cursor_y;
    logic [11:0] cursor_pixels;
    logic [11:0] cursor_color;
    logic center_old; //we change the value of a tile only on rising edge of center
    assign cursor_color = 12'hf0f;
    ones_pixels #(.WIDTH(16), .HEIGHT(16))
one(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(one_pixels));
    twos_pixels #(.WIDTH(16), .HEIGHT(16))
two(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(two_pixels));
    threes_pixels #(.WIDTH(16), .HEIGHT(16))
three(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(three_pixels));
    fours_pixels #(.WIDTH(16), .HEIGHT(16))
four(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(four_pixels));
    fives_pixels #(.WIDTH(16), .HEIGHT(16))
five(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(five_pixels));
    sixes_pixels #(.WIDTH(16), .HEIGHT(16))
six(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(six_pixels));
    sevens_pixels #(.WIDTH(16), .HEIGHT(16))
seven(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(seven_pixels));
    eights_pixels #(.WIDTH(16), .HEIGHT(16))
eight(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(eight_pixels));
    nines_pixels #(.WIDTH(16), .HEIGHT(16))
nine(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(nine_pixels));
    tens_pixels #(.WIDTH(16), .HEIGHT(16))
ten(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(ten_pixels));
    elevens_pixels #(.WIDTH(16), .HEIGHT(16))
eleven(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(eleven_pixels));
    twelves_pixels #(.WIDTH(16), .HEIGHT(16))
twelve(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twelve_pixels));
    thirteens_pixels #(.WIDTH(16), .HEIGHT(16))
thirteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirteen_pixels));
    forteens_pixels #(.WIDTH(16), .HEIGHT(16))
fourteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(fourteen_pixels));
    fifteens_pixels #(.WIDTH(16), .HEIGHT(16))
fifteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(fifteen_pixels));
    sixteens_pixels #(.WIDTH(16), .HEIGHT(16))
sixteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(sixteen_pixels));
    seventeens_pixels #(.WIDTH(16), .HEIGHT(16))
seventeen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(seventeen_pixels));
    eighteens_pixels #(.WIDTH(16), .HEIGHT(16))
eighteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(eighteen_pixels));
    nineteens_pixels #(.WIDTH(16), .HEIGHT(16))
nineteen(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(nineteen_pixels));
    twenties_pixels #(.WIDTH(16), .HEIGHT(16))
twenty(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twenty_pixels));
    twentyones_pixels #(.WIDTH(16), .HEIGHT(16))
twentyone(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentyone_pixels));
    twentytwos_pixels #(.WIDTH(16), .HEIGHT(16))
twentytwo(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentytwo_pixels));
```

```
    twentythrees_pixels #(.WIDTH(16), .HEIGHT(16))
twentythree(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentythree_pixels));
    twentyfours_pixels #(.WIDTH(16), .HEIGHT(16))
twentyfour(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentyfour_pixels));
    twentyfives_pixels #(.WIDTH(16), .HEIGHT(16))
twentyfive(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentyfive_pixels));
    twentysixes_pixels #(.WIDTH(16), .HEIGHT(16))
twentysix(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentysix_pixels));
    twentysevens_pixels #(.WIDTH(16), .HEIGHT(16))
twentyseven(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentyseven_pixels));
    twentyeights_pixels #(.WIDTH(16), .HEIGHT(16))
twentyeight(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentyeight_pixels));
    twentynines_pixels #(.WIDTH(16), .HEIGHT(16))
twentynine(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(twentynine_pixels));
    thirties_pixels #(.WIDTH(16), .HEIGHT(16))
thirty(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirty_pixels));
    thirtyones_pixels #(.WIDTH(16), .HEIGHT(16))
thirtyone(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtyone_pixels));
    thirtytwos_pixels #(.WIDTH(16), .HEIGHT(16))
thirtytwo(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtytwo_pixels));
    thirtythrees_pixels #(.WIDTH(16), .HEIGHT(16))
thirtythree(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtythree_pixels));
    thirtyfours_pixels #(.WIDTH(16), .HEIGHT(16))
thirtyfour(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtyfour_pixels));
    thirtyfives_pixels #(.WIDTH(16), .HEIGHT(16))
thirtyfive(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtyfive_pixels));
    thirtysixes_pixels #(.WIDTH(16), .HEIGHT(16))
thirtysix(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtysix_pixels));
    thirtysevens_pixels #(.WIDTH(16), .HEIGHT(16))
thirtyseven(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtyseven_pixels));
    thirtyeights_pixels #(.WIDTH(16), .HEIGHT(16))
thirtyeight(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtyeight_pixels));
    thirtynines_pixels #(.WIDTH(16), .HEIGHT(16))
thirtynine(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(thirtynine_pixels));
    forties_pixels #(.WIDTH(16), .HEIGHT(16))
forty(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(forty_pixels));

    assign blank_pixels = 12'hfff;
    logic  solvals [0:29][0:39];


     //30x40 array to store 1 and 0s of our grid for display
    logic  vals [0:29][0:39];// = {'{0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1},
//                              '{0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1},
//                              '{0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0},
//                              '{0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0},
//                              '{0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1},
//                              '{0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1},
//                              '{0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0},
//                              '{0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0},
//                              '{0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,1,1,1,1},
//                              '{0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1,0,0,0,1,1,1,1,0,0,1},
//                              '{0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,1},
//                              '{0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1,0,0,0,1,0,0,0,1,1,1},
//                              '{0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1,0,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1},
//                              '{1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0,1,1,1,1,0,0,0,1,1,0},
//                              '{0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0,0,0,0}};
                                                //1         2           3              4          5           6             7            8
  9            10          11            12             13       14             15          16
    logic  constraints [0:69][0:119];

    //create addressing system
    assign address_x = (hcount) >> 4;
    assign address_y = (vcount) >> 4;
    assign address_x_shift = (hcount -320) >>4;
    assign address_y_shift = (vcount - 240) >>4;
    assign lower_x = ((address_x<<2) + (address_x<<1));
    assign lower_y = ((address_y<<2) + (address_y<<1));

    assign x_in = address_x << 4;
    assign y_in = address_y << 4;

    //logic [0:39] solvals [0:29];
    logic [11:0] sol_pixels;
```

```
    assign top_val = {constraints[address_x_shift+30][lower_y +30],constraints[address_x_shift+30][lower_y +31],//30 added on top because we skip the
first 5 tiles (they dont exist)
                      constraints[address_x_shift+30][lower_y +32],constraints[address_x_shift+30][lower_y +33],
                      constraints[address_x_shift+30][lower_y +34],constraints[address_x_shift+30][lower_y+35]};

    assign left_val = {constraints[address_y_shift][lower_x+0],constraints[address_y_shift][lower_x+1],
                      constraints[address_y_shift][lower_x+2],constraints[address_y_shift][lower_x+3],
                       constraints[address_y_shift][lower_x+4],constraints[address_y_shift][lower_x+5]};

    assign top_pixels = (hcount > 320 & hcount < 960 & vcount <240) ? top_val==6'b000001 ? one_pixels : top_val==6'b000010 ?
                         two_pixels : top_val==6'b000011 ? three_pixels : top_val==6'b000100 ? four_pixels :
                         top_val==6'b000101 ? five_pixels : top_val==6'b000110 ? six_pixels : top_val==6'b000111 ?
                         seven_pixels : top_val==6'b001000 ? eight_pixels : top_val==6'b001001 ? nine_pixels :
                         top_val==6'b001010 ? ten_pixels : top_val==6'b001011 ? eleven_pixels : top_val==6'b001100 ?
                         twelve_pixels : top_val==6'b001101 ? thirteen_pixels : top_val==6'b001110 ? fourteen_pixels :
                         top_val==6'b001111 ? fifteen_pixels : top_val==6'b010000 ? sixteen_pixels : top_val==6'b010001 ?
                         seventeen_pixels : top_val==6'b010010 ? eighteen_pixels : top_val==6'b010011 ? nineteen_pixels :
                         top_val==6'b010100 ? twenty_pixels : top_val==6'b010101 ? twentyone_pixels : top_val==6'b010110 ?
                         twentytwo_pixels : top_val==6'b010111 ? twentythree_pixels : top_val==6'b011000 ? twentyfour_pixels :
                         top_val==6'b011001 ? twentyfive_pixels : top_val==6'b011010 ? twentysix_pixels : top_val==6'b011011 ?
                         twentyseven_pixels: top_val==6'b011100 ? twentyeight_pixels : top_val==6'b011101 ? twentynine_pixels :
                         top_val==6'b011110 ? thirty_pixels : top_val==6'b011111 ? thirtyone_pixels : top_val==6'b100000 ?
                         thirtytwo_pixels : top_val==6'b100001 ? thirtythree_pixels : top_val==6'b100010 ? thirtyfour_pixels :
                         top_val==6'b100011 ? thirtyfive_pixels : top_val==6'b100100 ? thirtysix_pixels : top_val==6'b100101 ?
                         thirtyseven_pixels : top_val==6'b100110 ? thirtyeight_pixels : top_val==6'b100111 ? thirtynine_pixels :
                         top_val==6'b101000 ? forty_pixels :12'hfff : 12'hfff;

    assign left_pixels =(vcount > 240 & vcount < 720 & hcount <320) ? left_val==6'b000001 ? one_pixels : left_val==6'b000010 ?
                         two_pixels : left_val==6'b000011 ? three_pixels : left_val==6'b000100 ? four_pixels :
                         left_val==6'b000101 ? five_pixels : left_val==6'b000110 ? six_pixels : left_val==6'b000111 ?
                         seven_pixels : left_val==6'b001000 ? eight_pixels : left_val==6'b001001 ? nine_pixels :
                         left_val==6'b001010 ? ten_pixels : left_val==6'b001011 ? eleven_pixels : left_val==6'b001100 ?
                         twelve_pixels : left_val==6'b001101 ? thirteen_pixels : left_val==6'b001110 ? fourteen_pixels :
                         left_val==6'b001111 ? fifteen_pixels : left_val==6'b010000 ? sixteen_pixels : left_val==6'b010001 ?
                         seventeen_pixels : left_val==6'b010010 ? eighteen_pixels : left_val==6'b010011 ? nineteen_pixels :
                         left_val==6'b010100 ? twenty_pixels : left_val==6'b010101 ? twentyone_pixels : left_val==6'b010110 ?
                         twentytwo_pixels : left_val==6'b010111 ? twentythree_pixels : left_val==6'b011000 ? twentyfour_pixels :
                         left_val==6'b011001 ? twentyfive_pixels : left_val==6'b011010 ? twentysix_pixels : left_val==6'b011011 ?
                         twentyseven_pixels: left_val==6'b011100 ? twentyeight_pixels : left_val==6'b011101 ? twentynine_pixels :
                         left_val==6'b011110 ? thirty_pixels : left_val==6'b011111 ? thirtyone_pixels : left_val==6'b100000 ?
                         thirtytwo_pixels : left_val==6'b100001 ? thirtythree_pixels : left_val==6'b100010 ? thirtyfour_pixels :
                         left_val==6'b100011 ? thirtyfive_pixels : left_val==6'b100100 ? thirtysix_pixels : left_val==6'b100101 ?
                         thirtyseven_pixels : left_val==6'b100110 ? thirtyeight_pixels : left_val==6'b100111 ? thirtynine_pixels :
                         left_val==6'b101000 ? forty_pixels :12'hfff : 12'hfff;


    assign grid_pixels = (hcount > 320 | vcount > 240) ? ((hcount % 16 ==0) | (vcount % 16 ==0)) ? 12'h000 : 12'hfff : 12'h000;
    assign tile_pixels = (hcount > 320 & vcount > 240 & hcount < 960 & vcount < 720) ? vals[address_y_shift][address_x_shift] ? 12'h000 : 12'hfff :
12'hfff;
    assign sol_pixels = show_sol ? (hcount > 320 & vcount > 240 & hcount < 960 & vcount < 720) ? solvals[address_y_shift][address_x_shift] ? 12'h00f :
12'hfff : 12'hfff : 12'hfff;
    assign cursor_pixels = (hcount > 320 & vcount > 240 & hcount < 960 & vcount < 720) ? (address_x_shift == cursor_x & address_y_shift ==cursor_y) ?
cursor_color : 12'hfff : 12'hfff;

    assign pixel_out = (hcount < 961 & vcount < 721) ? ~(~top_pixels + ~left_pixels + ~grid_pixels + ~tile_pixels + ~cursor_pixels + ~sol_pixels) :
12'hfff  ;

    //logic to generate number grid values
    logic [4:0] count2;

   //keeping track of previous states of buttons so we can assign on rising edge of buttons
   logic left_old,right_old,up_old,down_old,center_old;
   logic movel,mover,moveu,moved,select;
   assign movel = (left_old != left & left==1) ? 1 : 0;
   assign mover = (right_old != right & right==1) ? 1 : 0;
   assign moved = (down_old != down & down==1) ? 1 : 0;
   assign moveu = (up_old != up & up==1) ? 1 : 0;
   assign select = (center_old != center & center==1) ? 1 : 0;
    always_ff @(posedge clock) begin
        //logic to control position of cursor
        if (reset) begin
            constraints <= '{default:79'd0};
            vals <= '{default:1'b0};
            count_num <= 7'b0;
            center_old<=1'b1;
            receiving <= 1'b0;
            cursor_x <= 7'd0;
            cursor_y <= 7'd0;
            count_num <= 5'b0;
            count_num_photo <= 0;
            receiving <= 1'b0;

        end else if (start_sending_constraint && (count_num < 70)) begin
            constraints[count_num][0] <= constraint_vals[119];
            constraints[count_num][1] <= constraint_vals[118];
            constraints[count_num][2] <= constraint_vals[117];
            constraints[count_num][3] <= constraint_vals[116];
            constraints[count_num][4] <= constraint_vals[115];
            constraints[count_num][5] <= constraint_vals[114];
```

```
constraints[count_num][6] <= constraint_vals[113];
constraints[count_num][7] <= constraint_vals[112];
constraints[count_num][8] <= constraint_vals[111];
constraints[count_num][9] <= constraint_vals[110];
constraints[count_num][10] <= constraint_vals[109];
constraints[count_num][11] <= constraint_vals[108];
constraints[count_num][12] <= constraint_vals[107];
constraints[count_num][13] <= constraint_vals[106];
constraints[count_num][14] <= constraint_vals[105];
constraints[count_num][15] <= constraint_vals[104];
constraints[count_num][16] <= constraint_vals[103];
constraints[count_num][17] <= constraint_vals[102];
constraints[count_num][18] <= constraint_vals[101];
constraints[count_num][19] <= constraint_vals[100];
constraints[count_num][20] <= constraint_vals[99];
constraints[count_num][21] <= constraint_vals[98];
constraints[count_num][22] <= constraint_vals[97];
constraints[count_num][23] <= constraint_vals[96];
constraints[count_num][24] <= constraint_vals[95];
constraints[count_num][25] <= constraint_vals[94];
constraints[count_num][26] <= constraint_vals[93];
constraints[count_num][27] <= constraint_vals[92];
constraints[count_num][28] <= constraint_vals[91];
constraints[count_num][29] <= constraint_vals[90];
constraints[count_num][30] <= constraint_vals[89];
constraints[count_num][31] <= constraint_vals[88];
constraints[count_num][32] <= constraint_vals[87];
constraints[count_num][33] <= constraint_vals[86];
constraints[count_num][34] <= constraint_vals[85];
constraints[count_num][35] <= constraint_vals[84];
constraints[count_num][36] <= constraint_vals[83];
constraints[count_num][37] <= constraint_vals[82];
constraints[count_num][38] <= constraint_vals[81];
constraints[count_num][39] <= constraint_vals[80];
constraints[count_num][40] <= constraint_vals[79];
constraints[count_num][41] <= constraint_vals[78];
constraints[count_num][42] <= constraint_vals[77];
constraints[count_num][43] <= constraint_vals[76];
constraints[count_num][44] <= constraint_vals[75];
constraints[count_num][45] <= constraint_vals[74];
constraints[count_num][46] <= constraint_vals[73];
constraints[count_num][47] <= constraint_vals[72];
constraints[count_num][48] <= constraint_vals[71];
constraints[count_num][49] <= constraint_vals[70];
constraints[count_num][50] <= constraint_vals[69];
constraints[count_num][51] <= constraint_vals[68];
constraints[count_num][52] <= constraint_vals[67];
constraints[count_num][53] <= constraint_vals[66];
constraints[count_num][54] <= constraint_vals[65];
constraints[count_num][55] <= constraint_vals[64];
constraints[count_num][56] <= constraint_vals[63];
constraints[count_num][57] <= constraint_vals[62];
constraints[count_num][58] <= constraint_vals[61];
constraints[count_num][59] <= constraint_vals[60];
constraints[count_num][60] <= constraint_vals[59];
constraints[count_num][61] <= constraint_vals[58];
constraints[count_num][62] <= constraint_vals[57];
constraints[count_num][63] <= constraint_vals[56];
constraints[count_num][64] <= constraint_vals[55];
constraints[count_num][65] <= constraint_vals[54];
constraints[count_num][66] <= constraint_vals[53];
constraints[count_num][67] <= constraint_vals[52];
constraints[count_num][68] <= constraint_vals[51];
constraints[count_num][69] <= constraint_vals[50];
constraints[count_num][70] <= constraint_vals[49];
constraints[count_num][71] <= constraint_vals[48];
constraints[count_num][72] <= constraint_vals[47];
constraints[count_num][73] <= constraint_vals[46];
constraints[count_num][74] <= constraint_vals[45];
constraints[count_num][75] <= constraint_vals[44];
constraints[count_num][76] <= constraint_vals[43];
constraints[count_num][77] <= constraint_vals[42];
constraints[count_num][78] <= constraint_vals[41];
constraints[count_num][79] <= constraint_vals[40];
constraints[count_num][80] <= constraint_vals[39];
constraints[count_num][81] <= constraint_vals[38];
constraints[count_num][82] <= constraint_vals[37];
constraints[count_num][83] <= constraint_vals[36];
constraints[count_num][84] <= constraint_vals[35];
constraints[count_num][85] <= constraint_vals[34];
constraints[count_num][86] <= constraint_vals[33];
constraints[count_num][87] <= constraint_vals[32];
constraints[count_num][88] <= constraint_vals[31];
constraints[count_num][89] <= constraint_vals[30];
constraints[count_num][90] <= constraint_vals[29];
constraints[count_num][91] <= constraint_vals[28];
constraints[count_num][92] <= constraint_vals[27];
constraints[count_num][93] <= constraint_vals[26];
constraints[count_num][94] <= constraint_vals[25];
constraints[count_num][95] <= constraint_vals[24];
```

```
        constraints[count_num][96] <= constraint_vals[23];
        constraints[count_num][97] <= constraint_vals[22];
        constraints[count_num][98] <= constraint_vals[21];
        constraints[count_num][99] <= constraint_vals[20];
        constraints[count_num][100] <= constraint_vals[19];
        constraints[count_num][101] <= constraint_vals[18];
        constraints[count_num][102] <= constraint_vals[17];
        constraints[count_num][103] <= constraint_vals[16];
        constraints[count_num][104] <= constraint_vals[15];
        constraints[count_num][105] <= constraint_vals[14];
        constraints[count_num][106] <= constraint_vals[13];
        constraints[count_num][107] <= constraint_vals[12];
        constraints[count_num][108] <= constraint_vals[11];
        constraints[count_num][109] <= constraint_vals[10];
        constraints[count_num][110] <= constraint_vals[9];
        constraints[count_num][111] <= constraint_vals[8];
        constraints[count_num][112] <= constraint_vals[7];
        constraints[count_num][113] <= constraint_vals[6];
        constraints[count_num][114] <= constraint_vals[5];
        constraints[count_num][115] <= constraint_vals[4];
        constraints[count_num][116] <= constraint_vals[3];
        constraints[count_num][117] <= constraint_vals[2];
        constraints[count_num][118] <= constraint_vals[1];
        constraints[count_num][119] <= constraint_vals[0];
        count_num <= count_num + 1;

        if (count_num == 69) begin
            receiving <= 1'b0;
            count_num<=0;

        end



    end else if(start_sending_photo && (count_num_photo < 30)) begin

        solvals[count_num_photo][0] <= photo_in[0];
        solvals[count_num_photo][1] <= photo_in[1];
        solvals[count_num_photo][2] <= photo_in[2];
        solvals[count_num_photo][3] <= photo_in[3];
        solvals[count_num_photo][4] <= photo_in[4];
        solvals[count_num_photo][5] <= photo_in[5];
        solvals[count_num_photo][6] <= photo_in[6];
        solvals[count_num_photo][7] <= photo_in[7];
        solvals[count_num_photo][8] <= photo_in[8];
        solvals[count_num_photo][9] <= photo_in[9];
        solvals[count_num_photo][10] <= photo_in[10];
        solvals[count_num_photo][11] <= photo_in[11];
        solvals[count_num_photo][12] <= photo_in[12];
        solvals[count_num_photo][13] <= photo_in[13];
        solvals[count_num_photo][14] <= photo_in[14];
        solvals[count_num_photo][15] <= photo_in[15];
        solvals[count_num_photo][16] <= photo_in[16];
        solvals[count_num_photo][17] <= photo_in[17];
        solvals[count_num_photo][18] <= photo_in[18];
        solvals[count_num_photo][19] <= photo_in[19];
        solvals[count_num_photo][20] <= photo_in[20];
        solvals[count_num_photo][21] <= photo_in[21];
        solvals[count_num_photo][22] <= photo_in[22];
        solvals[count_num_photo][23] <= photo_in[23];
        solvals[count_num_photo][24] <= photo_in[24];
        solvals[count_num_photo][25] <= photo_in[25];
        solvals[count_num_photo][26] <= photo_in[26];
        solvals[count_num_photo][27] <= photo_in[27];
        solvals[count_num_photo][28] <= photo_in[28];
        solvals[count_num_photo][29] <= photo_in[29];
        solvals[count_num_photo][30] <= photo_in[30];
        solvals[count_num_photo][31] <= photo_in[31];
        solvals[count_num_photo][32] <= photo_in[32];
        solvals[count_num_photo][33] <= photo_in[33];
        solvals[count_num_photo][34] <= photo_in[34];
        solvals[count_num_photo][35] <= photo_in[35];
        solvals[count_num_photo][36] <= photo_in[36];
        solvals[count_num_photo][37] <= photo_in[37];
        solvals[count_num_photo][38] <= photo_in[38];
        solvals[count_num_photo][39] <= photo_in[39];


        count_num_photo <= count_num_photo +1;

        if (count_num_photo == 29) begin
            count_num_photo <=0;

        end


end else begin
        if (select) begin
            vals[cursor_y][cursor_x] <= ~vals[cursor_y][cursor_x];
```

```
            end
            //logic to control position of cursor
            if (moved && (cursor_y==29)) cursor_y <=cursor_y;
            else if (moved) cursor_y <= cursor_y + 6'd1;
            else if (movel && (cursor_x ==0)) cursor_x <= cursor_x;
            else if (movel) cursor_x <= cursor_x - 6'd1;
            else if (moveu && (cursor_y ==0))cursor_y<= cursor_y;
            else if (moveu) cursor_y <= cursor_y - 6'd1;
            else if (mover && (cursor_x==39)) cursor_x<=cursor_x;
            else if (mover) cursor_x <= cursor_x + 6'd1;
            else begin
                cursor_y <= cursor_y;
                cursor_x <= cursor_x;

            end


        end
        center_old <= center;
        up_old <= up;
        down_old <= down;
        left_old <= left;
        right_old <= right;

    end
endmodule
`default_nettype wire


module eights_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);

         logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_eight_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule

module sixes_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);

         logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_six_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module fives_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
```

```
    output logic [11:0] pixel_out);

         logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_five_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module fours_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);


         logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_four_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module tens_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);


    logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_ten_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule


module nines_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);


    logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
```

```
   logic red_mapped;
   small_nine_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module ones_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);


        logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_one_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module sevens_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);

     logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;
   logic red_mapped;
   small_seven_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
       end else  begin
            pixel_out <= 0;

        end
   end
endmodule
module twelve_pixels
   #(parameter WIDTH = 16,     // default picture width
               HEIGHT = 16)    // default picture height
   (input wire pixel_clk_in,
    input wire [10:0] x_in,hcount_in,
    input wire [9:0] y_in,vcount_in,
    output logic [11:0] pixel_out);


     logic [15:0] image_addr;
    assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

   logic red_mapped;
   small_twelve_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
   always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
```

```
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule
module threes_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);



     logic [15:0] image_addr;
     assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

    logic red_mapped;
    small_three_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
    always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule
module twos_pixels
    #(parameter WIDTH = 16,     // default picture width
                HEIGHT = 16)    // default picture height
    (input wire pixel_clk_in,
     input wire [10:0] x_in,hcount_in,
     input wire [9:0] y_in,vcount_in,
     output logic [11:0] pixel_out);

        logic [15:0] image_addr;
     assign image_addr = (hcount_in-x_in) + (vcount_in-y_in) * WIDTH;

    logic red_mapped;
    small_two_rom  rom1(.clka(pixel_clk_in), .addra(image_addr), .douta(red_mapped));
    always_ff @ (posedge pixel_clk_in) begin
        if ((hcount_in >= x_in && hcount_in < (x_in+WIDTH)) && (vcount_in >= y_in && vcount_in < (y_in+HEIGHT))) begin

            if (red_mapped == 1'b0) begin
                pixel_out <= 12'hfff; // white
            end else begin
                pixel_out <= 12'd0; // black
            end
        end else  begin
            pixel_out <= 0;

        end
    end
endmodule

`timescale 1ns / 1ps

module return_UI(
                    input wire clk_in,
                    input wire reset_in,
                    input wire [3:0] state,
                    input wire [12:0] hcount,
                    input wire [12:0] vcount,
                    input wire [15:0] switch,
                    output logic [11:0] pixel_out);
    parameter [10:0] WIDTH = 512;
    parameter [10:0] HEIGHT = 384;
    logic [1:0] solver_image_bits;
    logic [1:0] manual_image_bits;
    logic [1:0] generate_image_bits;
    logic [1:0] idle_bits;
    logic [15:0] solver_image_addr;
    logic [15:0] manual_image_addr;
    logic [15:0] generate_image_addr;
    logic [20:0] image_addr;   // num of bits for 256*240 ROM
    logic [7:0] image_bits, red_mapped, green_mapped, blue_mapped;

    assign image_addr = (hcount) + (vcount) * WIDTH;
```

```
    solve_manually_rom  rom2(.clka(clk_in), .addra(image_addr), .douta(manual_image_bits));
    solve_automatically_rom  rom1(.clka(clk_in), .addra(image_addr), .douta(solver_image_bits));
    generate_rom  rom4(.clka(clk_in), .addra(image_addr), .douta(generate_image_bits));
//00 - black
//01 -red
//10 - blue
//11 green
    always_ff @(posedge clk_in) begin

        if ((hcount < (WIDTH)) && (vcount < (HEIGHT))) begin
            // IDLE
            if(state ==4'b0000) begin
                if (manual_image_bits == 2'b00) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end else if (manual_image_bits == 2'b01) begin
                    pixel_out <= {4'b0000, 4'b1111,4'b0000};
                end else if (manual_image_bits == 2'b10) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b1111};
                end else if (manual_image_bits == 2'b11) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b1111};
                end else begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end
            //manual solving
            end else if (state ==4'b0001) begin
               if (manual_image_bits == 2'b00) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end else if (manual_image_bits == 2'b01) begin
                    pixel_out <= {4'b1111, 4'b0000,4'b0000};
                end else if (manual_image_bits == 2'b10) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b1111};
                end else if (manual_image_bits == 2'b11) begin
                    pixel_out <= {4'b0000, 4'b1111,4'b0000};
                end else begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end

            //automatic solving
            end else if (state ==4'b0010) begin
                if (solver_image_bits == 2'b00) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};

                end else if (solver_image_bits == 2'b01) begin
                    pixel_out <= {4'b1111, 4'b0000,4'b0000};
                end else if (solver_image_bits == 2'b10) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b1111};
                end else if (solver_image_bits == 2'b11) begin
                    pixel_out <= {4'b0000, 4'b1111,4'b0000};
                end else begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end
            //generate
            end else if(state ==4'b0011) begin
                if (generate_image_bits == 2'b00) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end else if (generate_image_bits == 2'b01) begin
                    pixel_out <= {4'b1111, 4'b0000,4'b0000};
                end else if (generate_image_bits == 2'b10) begin
                    pixel_out <= {4'b0000, 4'b0000,4'b1111};
                end else if (generate_image_bits == 2'b11) begin
                    pixel_out <= {4'b0000, 4'b1111,4'b0000};
                end else begin
                    pixel_out <= {4'b0000, 4'b0000,4'b0000};
                end
            end else begin

                    if (manual_image_bits == 2'b00) begin
                        pixel_out <= {4'b0000, 4'b0000,4'b0000};

                    end else if (manual_image_bits == 2'b01) begin
                        pixel_out <= {4'b0000, 4'b1111,4'b0000};

                    end else if (manual_image_bits == 2'b10) begin
                        pixel_out <= {4'b0000, 4'b0000,4'b1111};

                    end else if (manual_image_bits == 2'b11) begin
                        pixel_out <= {4'b0000, 4'b0000,4'b1111};

                    end else begin
                        pixel_out <= {4'b0000, 4'b0000,4'b0000};

                    end
              end

        end else begin
            pixel_out <= 0;
        end


    end
```

```
endmodule

`default_nettype none
module solved_disp(
                    input wire clock,
                    input wire reset,
                    input wire solver_done,
                    input wire memory_read_start,
                    input wire [19:0] constraint_vals,
                    input wire [9:0] grid_vals1,
                    input wire [9:0] grid_vals2,
                    input wire [9:0] grid_vals3,
                    input wire [9:0] grid_vals4,
                    input wire [9:0] grid_vals5,
                    input wire [9:0] grid_vals6,
                    input wire [9:0] grid_vals7,
                    input wire [9:0] grid_vals8,
                    input wire [9:0] grid_vals9,
                    input wire [9:0] grid_vals10,
                    input wire [12:0] hcount,
                    input wire [12:0] vcount,
                    input wire [15:0] switch,
                    output logic [11:0] pixel_out);

    logic done_old; //we use this to keep track of whether or not the done value switches states
    logic receiving; //lets the system know if it is in receiving mode or not
    logic [3:0] count_grid; //counts from 0-9 to keep track of the indexing with clock cycles for the 10x10
    logic [4:0] count_num; //counts from 0-19 to keep track of indexing with clock cycles for the 20x20 constraints
    logic [10:0] address_x;
    logic [10:0] address_y;
    logic [10:0] address_x_shift; //coordinates of every 16 pixels,
    logic [10:0] address_y_shift;
    logic [9:0] x_in; //coordinates upscaled to give display coordinates of image blobs
    logic [9:0] y_in;
    logic [10:0] lower_x; //addressing for registers of values, use [lower +:3] to grab 4 bit val
    logic [10:0] lower_y;

    //initialize our different layers of pixels
    logic [11:0] left_pixels;
    logic [11:0] top_pixels;
    logic [11:0] grid_pixels;
    logic [11:0] tile_pixels;
    logic [3:0] top_val;
    logic [3:0] left_val;

    //initialize the pixel values of our numbers
    logic [11:0] one_pixels;
    logic [11:0] two_pixels;
    logic [11:0] three_pixels;
    logic [11:0] four_pixels;
    logic [11:0] five_pixels;
    logic [11:0] six_pixels;
    logic [11:0] seven_pixels;
    logic [11:0] eight_pixels;
    logic [11:0] nine_pixels;
    logic [11:0] ten_pixels;
    logic [11:0] blank_pixels;



    ones_pixels #(.WIDTH(16), .HEIGHT(16))
one(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(one_pixels));
    twos_pixels #(.WIDTH(16), .HEIGHT(16))
two(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(two_pixels));
    threes_pixels #(.WIDTH(16), .HEIGHT(16))
three(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(three_pixels));
    fours_pixels #(.WIDTH(16), .HEIGHT(16))
four(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(four_pixels));
    fives_pixels #(.WIDTH(16), .HEIGHT(16))
five(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(five_pixels));
    sixes_pixels #(.WIDTH(16), .HEIGHT(16))
six(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(six_pixels));
    sevens_pixels #(.WIDTH(16), .HEIGHT(16))
seven(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(seven_pixels));
    eights_pixels #(.WIDTH(16), .HEIGHT(16))
eight(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(eight_pixels));
    nines_pixels #(.WIDTH(16), .HEIGHT(16))
nine(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(nine_pixels));
    tens_pixels #(.WIDTH(16), .HEIGHT(16))
ten(.pixel_clk_in(clock), .x_in(x_in), .y_in(y_in), .hcount_in(hcount), .vcount_in(vcount), .pixel_out(ten_pixels));
    assign blank_pixels = 12'hfff;

   //unpacking grid values
   logic grid1 [9:0];
   assign grid1=  '{grid_vals1[9],grid_vals1[8],grid_vals1[7],grid_vals1[6],grid_vals1[5],
                    grid_vals1[4],grid_vals1[3],grid_vals1[2],grid_vals1[1],grid_vals1[0]};
   logic grid2 [9:0];
   assign grid2 = '{grid_vals2[9],grid_vals2[8],grid_vals2[7],grid_vals2[6],grid_vals2[5],
                    grid_vals2[4],grid_vals2[3],grid_vals2[2],grid_vals2[1],grid_vals2[0]};
   logic grid3 [9:0];
```

```
    assign grid3 = '{grid_vals3[9],grid_vals3[8],grid_vals3[7],grid_vals3[6],grid_vals3[5],
                     grid_vals3[4],grid_vals3[3],grid_vals3[2],grid_vals3[1],grid_vals3[0]};
    logic grid4 [9:0];
    assign grid4= '{grid_vals4[9],grid_vals4[8],grid_vals4[7],grid_vals4[6],grid_vals4[5],
                    grid_vals4[4],grid_vals4[3],grid_vals4[2],grid_vals4[1],grid_vals4[0]};
    logic grid5 [9:0];
    assign grid5 = '{grid_vals5[9],grid_vals5[8],grid_vals5[7],grid_vals5[6],grid_vals5[5],
                    grid_vals5[4],grid_vals5[3],grid_vals5[2],grid_vals5[1],grid_vals5[0]};
    logic grid6 [9:0];
    assign grid6= '{grid_vals6[9],grid_vals6[8],grid_vals6[7],grid_vals6[6],grid_vals6[5],
                    grid_vals6[4],grid_vals6[3],grid_vals6[2],grid_vals6[1],grid_vals6[0]};
    logic grid7 [9:0];
    assign grid7= '{grid_vals7[9],grid_vals7[8],grid_vals7[7],grid_vals7[6],grid_vals7[5],
                    grid_vals7[4],grid_vals7[3],grid_vals7[2],grid_vals7[1],grid_vals7[0]};
    logic grid8 [9:0];
    assign grid8= '{grid_vals8[9],grid_vals8[8],grid_vals8[7],grid_vals8[6],grid_vals8[5],
                    grid_vals8[4],grid_vals8[3],grid_vals8[2],grid_vals8[1],grid_vals8[0]};
    logic grid9 [9:0];
    assign grid9 = '{grid_vals9[9],grid_vals9[8],grid_vals9[7],grid_vals9[6],grid_vals9[5],
                    grid_vals9[4],grid_vals9[3],grid_vals9[2],grid_vals9[1],grid_vals9[0]};
    logic grid10 [9:0];
    assign grid10= '{grid_vals10[9],grid_vals10[1],grid_vals10[8],grid_vals10[7],grid_vals10[6],
                    grid_vals10[4],grid_vals10[3],grid_vals10[2],grid_vals10[1],grid_vals10[0]};
     //10x10 array to store 1 and 0s of our grid for display
     logic  vals [0:9][0:9];

     logic  constraints [0:19][0:19];
     //create addressing system
     assign address_x = (hcount) >> 4;
     assign address_y = (vcount) >> 4;
     assign address_x_shift = (hcount -80) >>4;
     assign address_y_shift = (vcount - 80) >>4;
     assign lower_x = address_x<<2;
     assign lower_y = address_y<<2;

     assign x_in = address_x << 4;
     assign y_in = address_y << 4;


     assign top_val = {constraints[19-address_x_shift][lower_y +0],constraints[19-address_x_shift][lower_y +1],
                       constraints[19-address_x_shift][lower_y +2],constraints[19-address_x_shift][lower_y +3]};

     assign left_val = {constraints[address_y_shift][lower_x+0],constraints[address_y_shift][lower_x+1],
                        constraints[address_y_shift][lower_x+2],constraints[address_y_shift][lower_x+3]};

     assign top_pixels = (hcount > 80 & hcount < 240 & vcount <80) ? top_val==4'b0001 ? one_pixels : top_val==4'b0010 ?
                          two_pixels : top_val==4'b0011 ? three_pixels : top_val==4'b0100 ? four_pixels :
                          top_val==4'b0101 ? five_pixels : top_val==4'b0110 ? six_pixels : top_val==4'b0111 ?
                          seven_pixels : top_val==4'b1000 ? eight_pixels : top_val==4'b1001 ? nine_pixels :
                          top_val==4'b1010 ? ten_pixels : 12'hfff : 12'hfff;

     assign left_pixels = (vcount > 80 & vcount < 240 & hcount <80) ? left_val==4'b0001 ? one_pixels : left_val==4'b0010 ?
                          two_pixels : left_val==4'b0011 ? three_pixels : left_val==4'b0100 ? four_pixels :
                          left_val==4'b0101 ? five_pixels : left_val==4'b0110 ? six_pixels : left_val==4'b0111 ?
                          seven_pixels : left_val==4'b1000 ? eight_pixels : left_val==4'b1001 ? nine_pixels :
                          left_val==4'b1010 ? ten_pixels : 12'hfff : 12'hfff;

     assign grid_pixels = (hcount > 80 | vcount > 80) ? ((hcount % 16 ==0) | (vcount % 16 ==0)) ? 12'h000 : 12'hfff : 12'h000;
     assign tile_pixels = (hcount > 80 & vcount > 80 & hcount < 240 & vcount < 240) ? vals[address_y_shift][address_x_shift] ? 12'h000 : 12'hfff :
12'hfff;
     assign pixel_out = (hcount < 241 & vcount < 241) ? ~(~top_pixels + ~left_pixels + ~grid_pixels + ~tile_pixels) : 12'hfff  ;

     //logic to generate number grid values
     logic [4:0] count2;
     logic old_solved;
     always_ff @(posedge clock) begin

         if (reset) begin
             constraints <= '{default:20'b00000000000000000000};
             vals <= '{default:1'b0};
             count_grid <= 4'b0;
             count_num <= 5'b0;
             done_old<=1'b0;
             receiving <= 1'b0;
             old_solved <=0;
         end else if (memory_read_start & count_num < 10) begin
             constraints[count_num][19] <= constraint_vals[0];
             constraints[count_num][18] <= constraint_vals[1];
             constraints[count_num][17] <= constraint_vals[2];
             constraints[count_num][16] <= constraint_vals[3];
             constraints[count_num][15] <= constraint_vals[4];
             constraints[count_num][14] <= constraint_vals[5];
             constraints[count_num][13] <= constraint_vals[6];
             constraints[count_num][12] <= constraint_vals[7];
             constraints[count_num][11] <= constraint_vals[8];
             constraints[count_num][10] <= constraint_vals[9];
             constraints[count_num][9] <= constraint_vals[10];
             constraints[count_num][8] <= constraint_vals[11];
             constraints[count_num][7] <= constraint_vals[12];
             constraints[count_num][6] <= constraint_vals[13];
```

```
            constraints[count_num][5] <= constraint_vals[14];
            constraints[count_num][4] <= constraint_vals[15];
            constraints[count_num][3] <= constraint_vals[16];
            constraints[count_num][2] <= constraint_vals[17];
            constraints[count_num][1] <= constraint_vals[18];
            constraints[count_num][0] <= constraint_vals[19];
            count_num <= count_num + 5'b1;
        end else if (memory_read_start & count_num > 9) begin
            constraints[count_num][19] <= constraint_vals[0];
            constraints[count_num][18] <= constraint_vals[1];
            constraints[count_num][17] <= constraint_vals[2];
            constraints[count_num][16] <= constraint_vals[3];
            constraints[count_num][15] <= constraint_vals[4];
            constraints[count_num][14] <= constraint_vals[5];
            constraints[count_num][13] <= constraint_vals[6];
            constraints[count_num][12] <= constraint_vals[7];
            constraints[count_num][11] <= constraint_vals[8];
            constraints[count_num][10] <= constraint_vals[9];
            constraints[count_num][9] <= constraint_vals[10];
            constraints[count_num][8] <= constraint_vals[11];
            constraints[count_num][7] <= constraint_vals[12];
            constraints[count_num][6] <= constraint_vals[13];
            constraints[count_num][5] <= constraint_vals[14];
            constraints[count_num][4] <= constraint_vals[15];
            constraints[count_num][3] <= constraint_vals[16];
            constraints[count_num][2] <= constraint_vals[17];
            constraints[count_num][1] <= constraint_vals[18];
            constraints[count_num][0] <= constraint_vals[19];
            count_num <= count_num + 5'b1;

            if (count_num > 19) begin
                receiving <= 1'b0;
                count_num <= 5'b0;

            end

        end else if ((!old_solved) && (old_solved != solver_done)) begin
            vals[0][0:9] <= grid1;
            vals[1][0:9] <= grid2;
            vals[2][0:9] <= grid3;
            vals[3][0:9] <= grid4;
            vals[4][0:9] <= grid5;
            vals[5][0:9] <= grid6;
            vals[6][0:9] <= grid7;
            vals[7][0:9] <= grid8;
            vals[8][0:9] <= grid9;
            vals[9][0:9] <= grid10;
        end

       old_solved <=solver_done;

    end
endmodule
`default_nettype wire



`default_nettype none
module assignments_registry(
                    input wire clk_in,
                    input wire start_in,
                    input wire reset_in,
                    input wire [15:0] address_in,
                    output logic [19:0] assignment_out,
                    output logic [5:0] counter_out,
                    output logic done,
                    output logic sending
    );


    parameter ADDRESS_SIZE = 31;
    parameter SAMPLE_COUNT = 2082;//gets approximately (will generate audio at approx 48 kHz sample rate.
    logic [ADDRESS_SIZE:0] address;
    logic [15:0] dimensions;
    logic [31:0] address_input;







    logic started;
    logic [8:0] limit; // limit 200+200 = 400 > 9 bits
    //width of 20 height if at least 20 since at least for one nonogram
    logic [19:0] assignment_out1;
    logic [19:0] assignment_buffer;
    logic [19:0] assignment_buffer1;
    logic just_started;
    logic hello;
```

```
    logic [1:0] lol;
    logic acquired_address;

    assignments_rom  my_assignment_rom(.clka(clk_in), .addra(address_input), .douta(assignment_out1));
    always_ff @(posedge clk_in) begin
        if(reset_in) begin
            hello <=0;
            counter_out <=0;
            started<=0;
            sending<=0;
            done<=0;
            address_input<=0;
            assignment_out<=0;
            limit <=0;
            just_started <=0;
           // assignment_out1 <=0;
           acquired_address <=0;
           assignment_buffer1 <=0;

        end else begin
            if(start_in && ~acquired_address  && ~started) begin
                //address_input <= 21;

                lol <=address_in[15:14];
                address_input <= address_in[15:14] * 5'd20; //switch 15 adn 14 serve as adressing to select nonogram - so 4 in total to select from
                acquired_address <=1;
                limit <= 20;


            end else if (acquired_address) begin
                address_input <=address_input+1;
                started <=1;
                counter_out <= 0;

                acquired_address <=0;
                just_started <=1;


            end else if (started) begin
                acquired_address <=0;
                if (just_started) begin
                    just_started <=0;

                    address_input <=address_input+1;
                    counter_out <=  counter_out;
                    assignment_buffer <= assignment_out1;
                    assignment_buffer1 <= assignment_buffer;
                    assignment_out <= assignment_buffer1;
                    hello <=1;

                end else begin
                    if(counter_out == limit-1) begin
                        done <=1;
                        started<=0;
                        sending<=0;
                        just_started <=0;
                    end else begin
                        sending<=1;
                        address_input <=address_input+1;
                        if(hello) begin
                            hello <=0;
                            counter_out <=  0;
                        end else begin
                            counter_out <=  counter_out+1;
                        end

                        assignment_buffer <= assignment_out1;
                        assignment_buffer1 <= assignment_buffer;
                        assignment_out <= assignment_out1;
                    end


                end



            end

        end

    end




endmodule
`default_nettype wire
```

```
module create_a_row(
    input wire clk_in,
    input wire reset_in,
    input wire new_data, // assertes if sth new appears on the input (new cosntrins)
    input wire [3:0] constrain1,
    input wire [3:0] constrain2,
    input wire [3:0] constrain3,
    input wire [3:0] constrain4,
    input wire [3:0] constrain5,
    input wire [3:0] number_of_constraints,
    input wire [3:0] break1,
    input wire [3:0] break2,
    input wire [3:0] break3,
    input wire [3:0] break4,
    output logic [19:0] assignment_out, // 2- bit row output sic eeach cell encode by 2 bits
    output logic done,
    output logic [4:0] min_length
);
//tested - on new input
logic [4:0] i; //acts as afor loop counter
logic [5:0] running_sum_1; //5 bits sinnce max numebr is 20
logic [5:0] running_sum_2;
logic [5:0] running_sum_3;
logic [5:0] running_sum_4;
logic [5:0] running_sum_5;
logic [5:0] running_sum_6;
logic [5:0] running_sum_7;
logic [5:0] running_sum_8;
logic [19:0] new_row; //saves progrss
parameter limit = 20;
parameter original = 20'b10101010101010101010;
logic done_counting;
logic start_counting;
always_ff @(posedge clk_in) begin

    if(reset_in) begin
        i<=0;
        done_counting <=0;
        new_row <=0;
        start_counting <=0;
        assignment_out<=0;
        done<=0;
        min_length<=0;
        running_sum_1<=0; //5 bits sinnce max numebr is 20
        running_sum_2<=0;
        running_sum_3 <=0;
        running_sum_4<=0;
        running_sum_5<=0;
        running_sum_6<=0;
        running_sum_7<=0;
        running_sum_8<=0;

    end else if (new_data && ~start_counting) begin



        if(number_of_constraints == 2) begin

            running_sum_1 <= constrain1 + constrain1 + break1+ break1 +2; // always add tiwce sicne we migrate from 10 to 20
            running_sum_2 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 +2;
            running_sum_3 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 +2'd2;
            running_sum_4 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 +2'd2;
            running_sum_5 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 +2'd2;
            running_sum_6 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 +2'd2;
            running_sum_7 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 +2'd2;
            running_sum_8 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 + constrain5 +constrain5 + 2'd2;
        end else if(number_of_constraints == 3) begin
            running_sum_1 <= constrain1 + constrain1 + break1+ break1 +2; // always add tiwce sicne we migrate from 10 to 20
            running_sum_2 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 +2;
            running_sum_3 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 +4;
            running_sum_4 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 +4;
            running_sum_5 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 +4;
            running_sum_6 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 +4;
            running_sum_7 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 +4;
            running_sum_8 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 + constrain5 +constrain5 +4;
        end else if(number_of_constraints == 4) begin
            running_sum_1 <= constrain1 + constrain1 + break1+ break1 +2; // always add tiwce sicne we migrate from 10 to 20
            running_sum_2 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 +2 ;
            running_sum_3 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 +4;
            running_sum_4 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 +4;
            running_sum_5 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
```

```
break3 +6;
            running_sum_6 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 +6;
            running_sum_7 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 +6;
            running_sum_8 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 + constrain5 +constrain5 +6;

        end  else if(number_of_constraints == 5) begin

            running_sum_1 <= constrain1 + constrain1 + break1+ break1 +2; // always add tiwce sicne we migrate from 10 to 20
            running_sum_2 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 +2 ;
            running_sum_3 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 +4;
            running_sum_4 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 +4;
            running_sum_5 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 +6;
            running_sum_6 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 +6;
            running_sum_7 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 +8;
            running_sum_8 <= constrain1 + constrain1 + break1+ break1 + constrain2 + constrain2 + break2 + break2 + constrain3 + constrain3 + break3 +
break3 + constrain4 + constrain4 + break4 + break4 + constrain5 +constrain5 +8;
        end


        //reset values on new input
        new_row <=original;
        done_counting <=0;
        start_counting <=1;
        done <=0;
        i<=0;
        assignment_out<=0;

        min_length<=0;


    end else if (start_counting) begin
        if( done_counting) begin
            done <=1;
            assignment_out <= new_row;
            start_counting <=0;
            done_counting<=0;
            i<=0;

        end else begin
            min_length <=running_sum_8;
            i<=i+2;
            if(i < (constrain1 + constrain1)) begin
                //mark colored with 1
                new_row[i] <= 1;
                new_row[i+1] <= 0;
            end else if((i >=(constrain1 + constrain1)) && (i <running_sum_1)) begin
                new_row[i] <= 0;
                new_row[i+1] <= 1;
            end else if((i >=running_sum_1) && (i <running_sum_2)) begin
                new_row[i] <= 1;
                new_row[i+1] <= 0;
            end else if((i>=running_sum_2) && (i <running_sum_3)) begin
                new_row[i] <= 0;
                new_row[i+1] <= 1;
            end else if((i>=running_sum_3) && (i <running_sum_4)) begin
                new_row[i] <= 1;
                new_row[i+1] <= 0;
            end else if((i>=running_sum_4) && (i <running_sum_5)) begin
                new_row[i] <= 0;
                new_row[i+1] <= 1;
            end else if((i>=running_sum_5) && (i <running_sum_6)) begin
                new_row[i] <= 1;
                new_row[i+1] <= 0;
            end else if((i>=running_sum_6) && (i <running_sum_7)) begin
                new_row[i] <= 0;
                new_row[i+1] <= 1;
            end else if((i>=running_sum_7) && (i <running_sum_8)) begin
                new_row[i] <= 1;
                new_row[i+1] <= 0;

            end

            if(i >=running_sum_8) begin
                //marked the rest as unmarked  10.01.01.01.10.10.10.10.01.01
                done_counting<=1;

            end


        end
    end

end
Endmodule
```

```
`default_nettype none
module generate_rows(
                    input wire clk_in,
                    input wire start_in,
                    input wire reset_in,
                    input wire [19:0] assignment, // at most 5 sontrains of length 4
                    output logic done,
                    output logic outputing, //asserted when the new_row is ready on the output (fully)
                    output logic [19:0] new_row,
                    output logic [6:0] count, //current index of the row in the set of that we are oging to return
                    output logic [6:0] total_count //returns the tola nbumber of optison returend for a given setging
);


    parameter original = 20'b10101010101010101010;
    logic [5:0] i; //counter in for llop for 20 row genration
    logic  [2:0] number_of_breaks;
    logic  [2:0] space_to_fill_left;
    logic [11:0] breaks;
    logic start_generator;

    logic data_collected;
    logic permutation_started;

    logic started_shifting;
    logic [19:0] new_row1;

    logic [3:0] constrain1;
    logic [3:0] constrain2;
    logic [3:0] constrain3;
    logic [3:0] constrain4;
    logic [3:0] constrain5;
    logic [4:0] permutation_min_length;
    logic [3:0] number_of_numbers; // 5 is the max numer
    logic [2:0] number_of_breaks;
    logic [2:0] space_to_fill_left;
    logic returned_all_permutations;
    logic [15:0] permutation; // 6 bits encoding of breaks frm permutaiont module
    logic [11:0] permutation_out;
    logic [5:0] permutation_count;
    logic [5:0] counter; //used to coutn permutations when they are being returned

    logic [4:0] permutations_min_length_list [40:0]; //61 arryas of 5 bits ?
    logic [16:0] permutations_list [40:0]; //stroes numebrs -at most 30 permuations for a agiven set of constraints

    logic [19:0] basic_row_storage [40:0]; //stores actual rows - at most 60 row states
    logic [19:0] all_row_storage [40:0]; //stores actual rows - at most 60 row states



    logic create_a_row_from_permutations;

    logic [6:0] permutation_counter;


    logic in_progress;
    logic new_data; //assereted when new input to create_a_row is given

    logic done_generation;
    logic generating;
    logic generate_rows_from_basic;

    logic [6:0] shifts_limit;
    logic [6:0] shifts;
    logic [6:0] min_length;
    logic generate_states_from_permutations;
    logic len_1_started;

    logic [6:0] all_rows_counter;
    logic [4:0] running_sum_1; //5 bits sinnce max numebr is 20
    logic [4:0] running_sum_2;
    logic [4:0] running_sum_3;
    logic [4:0] running_sum_4;
    logic [4:0] running_sum_5;
    logic [4:0] running_sum_6;
    logic [4:0] running_sum_7;
    logic [4:0] running_sum_8;
    logic [19:0] new_row1;
    logic [19:0] new_row_from_create_a_row;
    logic [5:0] total_permutation_count;
    create_a_row my_create_a_row (
        .clk_in(clk_in),
        .reset_in(reset_in),
        .new_data(start_generator),

        .constrain1(constrain1),
        .constrain2(constrain2),
        .constrain3(constrain3),
```

```
    .constrain4(constrain4),
    .constrain5(constrain5),
    .number_of_constraints(number_of_numbers),
    .break1(permutation[3:0]),
    .break2(permutation[7:4]),
    .break3(permutation[11:8]),
    .break4(permutation[15:12]),
    .assignment_out(new_row_from_create_a_row),
    .done(done_generation),
    .min_length(permutation_min_length) //retusn the min length from filled to filled that covers all filled
);
get_permutations my_get_permutations(
                .clk_in(clk_in),
                .reset_in(reset_in),
                .start_in(starter), // asserted when we want to start generating permutations
                .number_of_breaks_in(number_of_breaks),//at most 4 breaks
                .space_to_fill_in(space_to_fill_left), // at most 5 space left (exclude teh compuslory break on the left)
                .permutation_out(permutation_out), //  mak of 4 breaks, eahc max encoded by 3 bits 4*3 ==12
                .done(returned_all_permutations),
                .counting(started_outputing_permutations),
                .total_counter(permutation_count) //returns the tola nbumber of optison returend for a given setging
);
logic started_outputing_permutations;
logic [6:0] create_a_row_counter;
logic wait_clock;
logic [19:0] old_version;
logic [6:0]  return_counter;
logic start_returning;
logic finished_returning;
logic starter;
logic [5:0] min_length;
logic [19:0] new_row2;
logic [19:0] assignment_stored;
logic returned_all_permutations_internal;
logic outputing_done;

always_ff @(posedge clk_in) begin
    if(reset_in || finished_returning) begin
        wait_clock<=0;
        outputing_done <=0;
        constrain1<=0;
        constrain2<=0;
        constrain3<=0;
        constrain4<=0;
        constrain5<=0;
        new_row1<=0;
        //permutation_min_length<=0;
        number_of_numbers<=0; // 5 is the max numer
        number_of_breaks<=0;
        space_to_fill_left<=0;
        //counters
        shifts <=0;
        number_of_breaks<=0;
        space_to_fill_left<=0;
        shifts_limit<=0;
        counter<=0;
        permutation_counter <=0;
        i<=0;
        create_a_row_counter <=0;

        number_of_numbers <=0;
        total_permutation_count<=0;
        all_rows_counter<=0;
        returned_all_permutations_internal<=0;
        //FSM
        in_progress <=0;
        len_1_started<=0;
        create_a_row_from_permutations<=0;
        //done_generation<=0;
        generating<=0;
        generate_rows_from_basic<=0;
        permutation_started<=0;
        data_collected<=0;
        generate_states_from_permutations <=0;
        start_generator <=0;
        //returned_all_permutations<=0;
        //outputs
        outputing<=0;
        count<=0;
        done <=0;
        total_count <=0;
        new_row<=20'b0;
        new_row1<=20'b0;
        //new_row_from_create_a_row<=20'b0;

        //regs
        //permutation_out <=0;
        running_sum_1<=0; //5 bits sinnce max numebr is 20
        running_sum_2<=0;
        running_sum_3 <=0;
```

```
        running_sum_4<=0;
        running_sum_5<=0;
        running_sum_6<=0;
        running_sum_7<=0;
        running_sum_8<=0;
        old_version <=0;
        return_counter <=0;
        start_returning <=0;
        finished_returning <=0;
        min_length<=0;
        new_row2 <=0;
        started_shifting<=0;
        assignment_stored<=0;

    end else if (start_returning) begin
        if (return_counter <=  (all_rows_counter-1)) begin
            return_counter <= return_counter +1;
            outputing <=1;
            new_row <=all_row_storage[return_counter];

        end else begin
            //on done reset the whole state machine
            //HERE
            finished_returning <=1;
            done <=1;
            outputing <=0;
            return_counter <=0;
            total_count<= all_rows_counter;
            //reset the state machine
            wait_clock<=0;
            permutation<=0;
            //counters
            shifts <=0;
            number_of_breaks<=0;
            space_to_fill_left<=0;
            shifts_limit<=0;
            counter<=0;
            permutation_counter <=0;
            i<=0;
            create_a_row_counter <=0;
            total_permutation_count<=0;
            min_length <=0;
            //FSM
            in_progress <=0;
            len_1_started<=0;
            create_a_row_from_permutations<=0;
            //done_generation<=0;
            generating<=0;
            generate_rows_from_basic<=0;
            permutation_started<=0;
            data_collected<=0;
            generate_states_from_permutations <=0;
            start_generator <=0;
            //returned_all_permutations<=0;
            returned_all_permutations_internal <=0;
            //outputs
            outputing<=0;
            count<=0;


            new_row<=20'b0;
            new_row1<=20'b0;
            new_row2<=20'b0;
            //new_row_from_create_a_row<=20'b0;

            //regs
            //permutation_out <=0;
            running_sum_1<=0; //5 bits sinnce max numebr is 20
            running_sum_2<=0;
            running_sum_3 <=0;
            running_sum_4<=0;
            running_sum_5<=0;
            running_sum_6<=0;
            running_sum_7<=0;
            running_sum_8<=0;
            old_version <=0;
            return_counter <=0;
            start_returning <=0;
            starter <=0;
            started_shifting<=0;
            assignment_stored<=0;
            new_data <=0;
        end
    end else if (create_a_row_from_permutations) begin
        // create BASIC rows
        if(~wait_clock) begin
            wait_clock <=1;

        end else if(~generating && wait_clock) begin
```

```
                generating<=1;
                start_generator <=1;
                 //used in create a row to mark incommingn new data
                permutation <= permutations_list[create_a_row_counter]; // input to create_a_row module - it only retusn ONE row per permutaiont
numerbs - the basic one, shifted to the left

            end else if (generating && start_generator) begin
                start_generator <=0;

            end else if (done_generation && generating) begin
                //save returned row
                old_version <=new_row_from_create_a_row;
                if (old_version != new_row_from_create_a_row) begin
                    basic_row_storage[create_a_row_counter] <= new_row_from_create_a_row;
                    permutations_min_length_list[create_a_row_counter] <= permutation_min_length;
                    create_a_row_counter <=create_a_row_counter +1;
                end


                generating<=0;
                wait_clock <=0;
                start_generator<=0;
            end
            if(create_a_row_counter >= total_permutation_count) begin
                //i have created and output all the rows for given cosntrinats
                create_a_row_from_permutations <=0;
                permutation_started<=0;

                in_progress <=0;
                generate_rows_from_basic <=1;
                counter <=0;
                create_a_row_counter <=0;
                new_data <=0;
                wait_clock<=0;
                generating <=0;

            end

        end else if (permutation_started) begin
            starter <=0;
            if(started_outputing_permutations) begin
                permutations_list[counter] <= permutation_out;

                counter <=counter +1;

            end else if(returned_all_permutations) begin
                returned_all_permutations_internal <=1;
                create_a_row_from_permutations <=1;
                permutation_started<=0;
                counter<=0;
                permutation_counter<=0; // used i nthe next state
                permutation_count <=counter;
                total_permutation_count <=counter;
                all_rows_counter<=0;


            end
        end else if (generate_rows_from_basic) begin
            //we are done with gerneating all the "basic" states for a given csontrin set > shift each basic set to left

                if(~started_shifting) begin
                    //we either just entered this state or we are done with shofting for  agiven basci state > introduce a enw basic state to new_row
                    new_row2 <=basic_row_storage[permutation_counter];
                    all_row_storage[all_rows_counter] <= basic_row_storage[permutation_counter];
                    all_rows_counter <=all_rows_counter+1;
                    // outputing <=1;
                    // count <= count +1;
                    permutation_counter <= permutation_counter +1;
                    shifts <=0;
                    min_length <=permutations_min_length_list[permutation_counter];
                    started_shifting <=1;


                end else begin
                    //save this whole thing to bram wbc clickcyclses suckkkk instead of returning avery clock cycle
                    //we are in the rpocess of shofting a given state
                    if(min_length < 5'd20) begin
                    //we are still allowed to shift
                        new_row2 <= {new_row2, 2'b10};
                        min_length <=min_length+ 2'b10;
                        all_row_storage[all_rows_counter] <= {new_row2, 2'b10};
                        all_rows_counter <=all_rows_counter+1;

                    end else begin
                        started_shifting <=0;
                        outputing <=0; // why ?

                        if(permutation_counter == total_permutation_count) begin
                            start_returning <=1;
                            //done<=1; // finish the whole genratE_row
```

```
                    generate_states_from_permutations <=0; // [otenailly need  ozero all the staes here ust to make sure
            end

            end



        end

end else if (data_collected && in_progress) begin
    //we have collected all the permutatiosn but now we need to shift them
    if (assignment_stored == 20'b0) begin
        count <= count+1;
        if(count == 11) begin
            done <=1;
            outputing <=0;
            finished_returning <=1;
            total_count<= 11;
        end else begin
            new_row <= original;
            outputing <=1;
            done <=0;
        end



    end else if(min_length == 10) begin
        //if it's the "best case scenario" - we have onyl one posible way to create a row - return it right away
        i<=i+2;

        if(outputing_done) begin
            finished_returning <=1;
            outputing<=0;
            outputing_done <=0;

        end
        if(i == 20) begin
            done<=1;
            count<=1;
            total_count <=1;
            i<=0;
            outputing<=1;
            new_row <=new_row1;
            outputing_done <=1;

        end else if(i <running_sum_1) begin
                    new_row1[i] <= 1'b1;
                    new_row1[i+1] <= 1'b0;
        end else if( i == running_sum_1) begin
                    new_row1[i] <= 1'b0;
                    new_row1[i+1] <= 1'b1;
        end else if(i < running_sum_2 && i > running_sum_1) begin
                    new_row1[i] <= 1'b1;
                    new_row1[i+1] <= 1'b0;
        end else if( i ==  running_sum_2) begin
                    new_row1[i] <= 1'b0;
                    new_row1[i+1] <= 1'b1;
        end else if(i < running_sum_3 && i >  running_sum_2) begin
                    new_row1[i] <= 1'b1;
                    new_row1[i+1] <= 1'b0;
        end else if( i ==  running_sum_3) begin
                    new_row1[i] <= 0;
                    new_row1[i+1] <= 1;
        end else if(i < running_sum_4 && i >  running_sum_3 ) begin
                    new_row1[i] <= 1'b1;
                    new_row1[i+1] <= 1'b0;
        end else if( i ==  running_sum_4) begin
                    new_row1[i] <= 1'b0;
                    new_row1[i+1] <= 1'b1;
        end else if( i <  running_sum_5 && i >  running_sum_4) begin
                    new_row1[i] <= 1'b1;
                    new_row1[i+1] <= 1'b0;
        end



    end else if(number_of_numbers == 1) begin
        //we have a single constrain
        if(len_1_started) begin

            count <= count+1; //increment the "idnex" of the returend row
            new_row <= {new_row, 2'b10}; //shoft one to the left
            if(count == total_count-1) begin
                //TODO - add one more FSM state to have one clock cycel delay between outpuing and done  ?
                //HERE - done up, out down
                done <=1;
                data_collected<=0;

                outputing<=0;
```

```
                len_1_started <=0;
                in_progress <=0;
            end
        end else begin
            //start internal fsm to shift
            total_count <= 7'd10 - assignment[3:0] +1'd1; // total number of rows toreturn

            i<=i+2;

            if( i > 18) begin
                len_1_started<=1;
                outputing<=1;
                i<=0;
                new_row<=new_row1;
                //count <=count+1;

            end else if(i <(assignment[3:0] + assignment[3:0])) begin
                    new_row1[i] = 1;
                    new_row1[i+1] = 0;
            end else begin
                    new_row1[i] = 0;
                    new_row1[i+1] = 1;
            end

        end

    end else begin
        //we can genreates statees "logically" - f
        permutation_started <=1;
        starter <=1;
        data_collected<=0;
        count<=0;

    end


end else if (start_in && (~in_progress)) begin
    //we have provided new data in assingemtne > syart the whole fsm again
    data_collected<=1;
    in_progress <=1; //start generation
    //zero all the values that might have been assignesd to sth before
    total_count <=0;
    count <=0;
    done <=0;
    outputing <=0;

    i<=0;
    assignment_stored <=assignment;
    constrain1<=assignment[3:0];
    constrain2<=assignment[7:4];
    constrain3<=assignment[11:8];
    constrain4<=assignment[15:12];
    constrain5<=assignment[19:16];
    running_sum_1 <= assignment[3:0] + assignment[3:0]; // always add tiwce sicne we migrate from 10 to 20
    running_sum_2 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] ;
    running_sum_3 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] + 2 + assignment[11:8] + assignment[11:8];
    running_sum_4 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] + 2 + assignment[11:8] + assignment[11:8] + 2 +
assignment[15:12] + assignment[15:12];
    running_sum_5 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] + 2 + assignment[11:8] + assignment[11:8] + 2 +
assignment[15:12] + assignment[15:12] + 2 + assignment[19:16] + assignment[19:16];

    //gernate numebr of breaks and spaces lef based on the passed cosntraints
    if (assignment[3:0] > 4'b0000 && assignment[7:4]<=4'b0000) begin
        min_length <= assignment[3:0];
        number_of_breaks <= 3'd0;
        number_of_numbers <= 3'd1;
        space_to_fill_left <= 10 - assignment[3:0];

        running_sum_1 <= assignment[3:0] + assignment[3:0]; // always add tiwce sicne we migrate from 10 to 20
        running_sum_2 <= assignment[3:0] + assignment[3:0];
        running_sum_3 <= assignment[3:0] + assignment[3:0];
        running_sum_4 <= assignment[3:0] + assignment[3:0];
        running_sum_5 <= assignment[3:0] + assignment[3:0];


    end else if (assignment[7:4]>4'b0000 && assignment[11:8]<=4'b0000) begin
        number_of_breaks<=1;
        number_of_numbers<=2;
        min_length <= assignment[3:0] + assignment[7:4] +1;
        space_to_fill_left <= 10 - assignment[3:0] - assignment[7:4] - 1;
        running_sum_1 <= assignment[3:0] + assignment[3:0]; // always add tiwce sicne we migrate from 10 to 20
        running_sum_2 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] ;
        running_sum_3 <=5'b11111;
        running_sum_4 <= 5'b11111;
        running_sum_5 <= 5'b11111;

    end else if ((assignment[11:8]>4'b0000) && assignment[15:12]<=1'b0) begin
        number_of_breaks<=2;
        number_of_numbers<=3;
        space_to_fill_left <= 10 - assignment[3:0] - assignment[7:4] - assignment[11:8]  - 2'd2;
```

```
                min_length <= assignment[3:0] + assignment[7:4] + assignment[11:8] + 2; // min lenght of the run
                running_sum_1 <= assignment[3:0] + assignment[3:0]; // always add tiwce sicne we migrate from 10 to 20
                running_sum_2 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] ;
                running_sum_3 <= assignment[3:0] + assignment[3:0] + 2 + assignment[7:4] + assignment[7:4] + 2 + assignment[11:8] + assignment[11:8];
                running_sum_4 <= 5'b11111;
                running_sum_5 <= 5'b11111;

            end else if (assignment[15:12]>1'b0 && assignment[19:16]<=1'b0) begin
                number_of_breaks<=3;
                number_of_numbers<=4;
                space_to_fill_left <= 10 - assignment[3:0] - assignment[7:4] - assignment[11:8] - assignment[15:12]  - 3;
                min_length <= assignment[3:0] + assignment[7:4] + assignment[11:8] + assignment[15:12] + 3; // min lenght of the run

            end else if (assignment[19:16]>1'b0) begin
                number_of_breaks<=4;
                number_of_numbers<=5;
                min_length <= assignment[3:0] + assignment[7:4] + assignment[11:8] + assignment[15:12] + assignment[19:16] + 4; // min lenght of the
run
                space_to_fill_left <= 10 - (assignment[3:0] + assignment[7:4] +assignment[11:8]  + assignment[15:12] + assignment[19:16]) -4;

            end
        end

    end
endmodule
`default_nettype wire


`default_nettype none
module get_permutations(
                    input wire clk_in,
                    input wire reset_in,
                    input wire start_in,
                    input wire [2:0] number_of_breaks_in,//at most 4 breaks
                    input wire [2:0] space_to_fill_in, // at most 5 space left
//most signifcant bits encode numebrs
                    output logic [15:0] permutation_out, //  mak of 4 breaks, eahc max encoded by 3 bits 4*3 ==12
                    output logic done,
                    output logic counting, //asserted when it actuall starts sending permutation
                    output logic [5:0] total_counter //returns the tola nbumber of optison returend for a given setging

    );
    //assume that if ther eis one break (2 numbers in the row) or a single sumber (no breaks) - it will be handled by the solver
//5 is the max amount of space to fill (4 bits)

logic [5:0] counter;
logic switch_off_done;

logic  [2:0] number_of_breaks;//at most 4 breaks
logic [2:0] space_to_fill; // at most 5 space left
    always_ff @(posedge clk_in) begin
        if (reset_in) begin
            done <=0;
            counter <=0;
            counting <=0;
             switch_off_done <=0;

            total_counter <=0;
            permutation_out<=0;
            number_of_breaks<=0;
            space_to_fill<=0;
        end else if (switch_off_done) begin
            switch_off_done <=0;
            done <=0;
        end else if (start_in && ~counting) begin

            counting <=1;
            done<=0;
            number_of_breaks<=number_of_breaks_in;
            space_to_fill<=space_to_fill_in;

        end else if (counting) begin
            counter<=counter +1;
            if(number_of_breaks == 1) begin
                if(space_to_fill ==0) begin
                    counter<=counter +1;

                    case(counter)
                        6'b0: permutation_out<=16'b0000_0000_0000;
                        default: permutation_out<= 16'b000_000000000;
                    endcase
                    if(counter ==0) begin
                        counting <=0;
                        total_counter <=total_counter + 1;
                        done<=1;
                        switch_off_done <=1;
                        counter <=0;
                        counting <=0;
```

```
        total_counter <=0;
        permutation_out<=0;
        number_of_breaks<=0;
        space_to_fill<=0;

    end
end else if (space_to_fill ==1) begin
counter<=counter +1;
    case(counter)
        4'b0: permutation_out <= 16'b00000000_0001;
        default: permutation_out<= 16'b0000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==2) begin
    counter<=counter +1;
    case(counter)
        6'b0: permutation_out<= 16'b00000000_0010;
        default: permutation_out<= 16'b0000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==3) begin
    case(counter)
        4'b0: permutation_out<= 16'b00000000_0011;
        default: permutation_out<= 16'b0000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==4) begin
counter<=counter +1;
    case(counter)
        4'b0: permutation_out<= 16'b00000000_0100;
        default: permutation_out<= 16'b0000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==5) begin
    case(counter)
        6'b0: permutation_out<= 16'b00000000_0101;
        default: permutation_out<= 16'b0000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==6) begin
    case(counter)
        6'b0: permutation_out<=12'b00000000_0110;
        default: permutation_out<=12'b000000000000;
    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==7) begin
    case(counter)
        6'b0: permutation_out<=12'b00000000_0111;
        default: permutation_out<=12'b000000000000;

    endcase
    if(counter ==0) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 1;
```

```
            space_to_fill <=space_to_fill - 1;
        end
    end else if (space_to_fill ==8) begin
        case(counter)
            6'b0: permutation_out<=12'b00000000_1000;
            default: permutation_out<=12'b000000000000;

        endcase
        if(counter ==0) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 1;
            space_to_fill <=space_to_fill - 1;
        end
    end

end else if(number_of_breaks == 2) begin
    if(space_to_fill ==0) begin
        case(counter)
            6'b0: permutation_out<=12'b000000000000;
            default: permutation_out<=12'b000000000000;

        endcase
        if(counter ==0) begin
            counting <=0;
            done<=1;
            switch_off_done <=1;
            total_counter <=total_counter + 1;
            counter <=0;
            counting <=0;

            total_counter <=0;
            permutation_out<=0;
            number_of_breaks<=0;
            space_to_fill<=0;
        end
    end else if (space_to_fill ==1) begin
        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0000_0001; // each break is encoded by 4 bits
            6'b1: permutation_out<=16'b0000_0000_0001_0000;
            default: permutation_out<=12'b000000000000;

        endcase
        if(counter ==1) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 2;
            space_to_fill <=space_to_fill - 1;
        end
    end else if (space_to_fill ==2) begin
        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0001_0001; // 11
            6'd1: permutation_out<=16'b0000_0000_0000_0010; //02
            6'd2: permutation_out<=16'b0000_0000_0010_0000; //20
            default: permutation_out<=16'b000000000000;

        endcase
        if(counter ==2) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 3;
            space_to_fill <=space_to_fill - 1;
        end
    end else if (space_to_fill ==3) begin
        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0001_0010; // 12
            6'd1: permutation_out<=16'b0000_0000_0010_0001; //21
            6'd2: permutation_out<=16'b0000_0000_0000_0011; //03
            6'd3: permutation_out<=16'b0000_0000_0011_0000; //30
            default: permutation_out<=12'b000000000000;

        endcase
        if(counter ==3) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 4;
            space_to_fill <=space_to_fill - 1;
        end
    end else if (space_to_fill ==4) begin
        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0010_0010; // 22
            6'd1: permutation_out<=16'b0000_0000_0011_0001; //31
            6'd2: permutation_out<=16'b0000_0000_0001_0011; //13
            6'd3: permutation_out<=16'b0000_0000_0100_0000; //40
            6'd4: permutation_out<=16'b0000_0000_0000_0100; //04
            default: permutation_out<=12'b000000000000;

        endcase
        if(counter ==4) begin
            counting <=1;
```

```
                counter<=0;
                total_counter <=total_counter + 5;
                space_to_fill <=space_to_fill - 1;
            end
        end else if (space_to_fill ==5) begin
            case(counter)
                6'b0: permutation_out<=16'b0000_0000_0101_0000; // 50
                6'd1: permutation_out<=16'b000_0000_0000_0101; //05
                6'd2: permutation_out<=16'b0000_0000_0100_0001; //41
                6'd3: permutation_out<=16'b0000_0000_0001_0100; //14
                6'd4: permutation_out<=16'b0000_0000_0011_0010; //32
                6'd5: permutation_out<=16'b0000_0000_0010_0011; //23
                default: permutation_out<=12'b000000000000;
            endcase
            if(counter ==5) begin
                counting <=1;
                counter<=0;
                total_counter <=total_counter + 6;
                space_to_fill <=space_to_fill - 1;
            end
    end


end else if (number_of_breaks == 3) begin
    if(space_to_fill ==0) begin
        case(counter)
            4'b0: permutation_out<= 12'b000000000_0000;
            default: permutation_out<= 12'b0000000000000;

        endcase
        if(counter ==0) begin
            counting <=0;
            total_counter <=total_counter + 1;
            done<=1;
            switch_off_done <=1;
                counter <=0;
                counting <=0;

                total_counter <=0;
                permutation_out<=0;
                number_of_breaks<=0;
                space_to_fill<=0;
        end
    end else if (space_to_fill ==1) begin

        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0000_0001;
            6'd1: permutation_out<=16'b0000_0000_0001_0000;
            6'd2: permutation_out<=16'b0000_0001_0000_0000;
            default: permutation_out<=12'b000000000000;
        endcase
        if(counter ==2) begin
            counting <=0;
            done<=1;
            switch_off_done <=1;
            counter <=0;
            counter <=0;
                counting <=0;

                total_counter <=0;
                permutation_out<=0;
                number_of_breaks<=0;
                space_to_fill<=0;
        end
    end else if (space_to_fill ==2) begin
        case(counter)
            6'b0: permutation_out<=16'b0000_0000_0001_0001;
            6'd1: permutation_out<=16'b0000_0001_0001_0000;
            6'd2: permutation_out<=16'b0000_0001_0000_0001;
            6'd3: permutation_out<=16'b0000_0000_0000_0010;
            6'd4: permutation_out<=16'b0000_0010_0000_0000;
            6'd5: permutation_out<=16'b0000_0000_0010_0000;
            default: permutation_out<=16'b000000000000;
        endcase
        if(counter ==5) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 6;
            space_to_fill <=space_to_fill - 1;
        end
    end else if (space_to_fill ==3) begin
        case(counter)
            4'b0: permutation_out<=16'b0000_0001_0001_0001; //111
            4'd1: permutation_out<=16'b0000_0000_0000_0011; //003
            4'd2: permutation_out<=16'b0000_0000_0011_0000; //030
            4'd3: permutation_out<=16'b0000_0011_0000_0000; //300
            4'd4: permutation_out<=16'b0000_0000_0010_0001; //021
            4'd5: permutation_out<=16'b0000_0001_0010_0000; //120
            4'd6: permutation_out<=16'b0000_0000_0001_0010; //012
            6'd7: permutation_out<=16'b0000_0010_0001_0000; //210
```

```
            6'd8: permutation_out<=16'b0000_0010_0000_0001; //201
            6'd9: permutation_out<=16'b0000_0001_0000_0010; //102
            default: permutation_out<=12'b000000000000;
        endcase
        if(counter ==9) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 10;
            space_to_fill <=space_to_fill - 1;
        end

    end else if (space_to_fill ==4) begin

        case(counter)
            4'b0: permutation_out<=16'b0000_0000_0100_0000; //004
            4'd1: permutation_out<=16'b0000_0000_0000_0100; //040
            4'd2: permutation_out<=16'b0000_0100_0000_0000; //400
            4'd3: permutation_out<=16'b0000_0000_0011_0001; //031
            4'd4: permutation_out<=16'b0000_0001_0000_0011; //103
            4'd5: permutation_out<=16'b0000_0001_0011_0000; //130
            4'd6: permutation_out<=16'b0000_0011_0000_0001; //301
            4'd7: permutation_out<=16'b0000_0000_0010_0011; //013
            4'd8: permutation_out<=16'b0000_0011_0001_0000; //310
            4'd9: permutation_out<=16'b0000_0000_0010_0010; //022
            4'd10: permutation_out<=16'b0000_0010_0010_0000; //220
            default: permutation_out<=16'b0000_0000_0000_0000;
        endcase
        if(counter ==10) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 11;
            space_to_fill <=space_to_fill - 1;
        end


    end else if (space_to_fill ==5) begin
        case(counter)
            4'b0: permutation_out<=16'b0000_0000_0000_0101; //005
            4'd1: permutation_out<=16'b0000_0000_0101_0000; //050
            4'd2: permutation_out<=16'b0000_0101_0000_0000; //500
            4'd3: permutation_out<=16'b0000_0000_0011_0010; //032
            4'd4: permutation_out<=16'b0000_0010_0011_0000; //230
            4'd5: permutation_out<=16'b0000_0000_0010_0011; //023
            4'd6: permutation_out<=16'b0000_0011_0010_0000; //320
            4'd7: permutation_out<=16'b0000_0010_0000_0011; //203
            4'd8: permutation_out<=16'b0000_0011_0000_0010; //302
            4'd9: permutation_out<=16'b0000_0011_0001_0001; //311
            4'd10: permutation_out<=16'b0000_0001_0011_0001; //131
            4'd11: permutation_out<=16'b0000_0001_0001_0011; //113
            6'd12: permutation_out<=16'b0000_0000_0001_0100; //014
            6'd13: permutation_out<=16'b0000_0100_0001_0000; //410
            6'd14: permutation_out<=16'b0000_0000_0100_0001; //041
            6'd15: permutation_out<=16'b0000_0001_0100_0000; //140
            6'd16: permutation_out<=16'b0000_0100_0000_0001; //401
            6'd17: permutation_out<=16'b0000_0100_0000_0100; //104
            default: permutation_out<=12'b0000_0000_000_0000;
        endcase
        if(counter ==17) begin
            counting <=1;
            counter<=0;
            total_counter <=total_counter + 18;
            space_to_fill <=space_to_fill - 1;
        end

end
    end else if (number_of_breaks == 4) begin

        if (space_to_fill ==0) begin

            case(counter)
                4'b0: permutation_out<=12'b000_000_000_000;
                default: permutation_out<=12'b000000000000;
            endcase
            if(counter ==0) begin
                counting <=0;
                total_counter <=total_counter + 1;
                done<=1;
                switch_off_done <=1;
                                    counter <=0;
                counting <=0;

                total_counter <=0;
                permutation_out<=0;
                number_of_breaks<=0;
                space_to_fill<=0;///01.01.01/.10/.01.01.10./10.10.10
            end
        end else if (space_to_fill ==1) begin
            case(counter)
                6'b0: permutation_out<=16'b0000_0000_0000_0001;
                6'd1: permutation_out<=16'b0000_0000_0001_0000;
```

```
        6'd2: permutation_out<=16'b0000_0001_0000_0000;
        6'd3: permutation_out<=16'b0001_0000_0000_0000;
        default: permutation_out<=16'b000000000000;
    endcase
    if(counter ==3) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 4;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==2) begin
    case(counter)
        4'b0: permutation_out<=12'b000_000_001_001; //0011
        4'd1: permutation_out<=12'b000_001_001_000; //0110
        4'd2: permutation_out<=12'b001_001_000_000; //1100
        4'd3: permutation_out<=12'b000_001_000_001; //0101
        4'd4: permutation_out<=12'b001_000_001_000; //1010
        4'd5: permutation_out<=12'b001_000_000_001; //1001
        4'd6: permutation_out<=12'b000_000_000_010; //0002
        4'd7: permutation_out<=12'b000_000_010_000; //0020
        4'd8: permutation_out<=12'b000_010_000_000; //0200
        4'd9: permutation_out<=12'b010_000_000_000; //2000
        default: permutation_out<=12'b000000000000;
    endcase
    if(counter ==9) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 10;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==3) begin
    case(counter)
        4'b0: permutation_out<=12'b000_000_001_010; //0012
        4'd1: permutation_out<=12'b000_001_010_000; //0120
        4'd2: permutation_out<=12'b001_010_000_000; //1200
        4'd3: permutation_out<=12'b000_001_000_010; //0102
        4'd4: permutation_out<=12'b001_000_010_000; //1020
        4'd5: permutation_out<=12'b001_000_000_010; //1002
        4'd6: permutation_out<=12'b000_000_010_001; //0021
        4'd7: permutation_out<=12'b000_010_001_000; //0210
        4'd8: permutation_out<=12'b010_001_000_000; //2100
        4'd9: permutation_out<=12'b000_010_000_001; //0201
        4'd10: permutation_out<=12'b010_000_001_000; //2010
        4'd11: permutation_out<=12'b010_000_000_001; //2001

        4'd12: permutation_out<=12'b000_000_000_011; //0003
        4'd13: permutation_out<=12'b000_000_011_000; //0030
        4'd14: permutation_out<=12'b000_011_000_000; //0300
        4'd15: permutation_out<=12'b011_000_000_000; //3000
        6'd16: permutation_out<=12'b000_001_001_001; //0111
        6'd17: permutation_out<=12'b001_001_001_000; //1110
        6'd18: permutation_out<=12'b001_001_000_001; //1101
        6'd19: permutation_out<=12'b001_000_001_001; //1011
        default: permutation_out<=12'b000000000000;
    endcase
    if(counter ==19) begin
        counting <=1;
        counter<=0;
        total_counter <=total_counter + 20;
        space_to_fill <=space_to_fill - 1;
    end
end else if (space_to_fill ==4) begin
    case(counter)
        6'b0: permutation_out<=12'b000_000_000_100; //0004
        6'd1: permutation_out<=12'b000_000_100_000; //0040
        6'd2: permutation_out<=12'b000_100_000_000; //0400
        6'd3: permutation_out<=12'b100_000_000_000; //4000
        6'd4: permutation_out<=12'b000_000_011_001; //0031
        6'd5: permutation_out<=12'b000_011_001_000; //0310
        6'd6: permutation_out<=12'b011_001_000_000; //3100
        6'd7: permutation_out<=12'b000_000_001_011; //0013
        6'd8: permutation_out<=12'b000_001_011_000; //0130
        6'd9: permutation_out<=12'b001_011_000_000; //1300
        6'd10: permutation_out<=12'b000_011_000_001; //0301
        6'd11: permutation_out<=12'b011_000_001_000; //3010
        6'd11: permutation_out<=12'b000_001_000_011; //0103
        6'd12: permutation_out<=12'b001_000_011_000; //1030
        6'd13: permutation_out<=12'b011_000_000_001; //3001
        6'd14: permutation_out<=12'b001_000_000_011; //1003
        6'd15: permutation_out<=12'b000_000_010_010; //0022
        6'd16: permutation_out<=12'b000_010_000_010; //0202
        6'd17: permutation_out<=12'b010_000_000_010; //2002
        6'd18: permutation_out<=12'b010_000_010_000; //2020
        6'd19: permutation_out<=12'b010_010_000_000; //2200
        6'd20: permutation_out<=12'b001_001_001_001; //1111
        6'd21: permutation_out<=12'b000_001_001_010; //0112
        6'd22: permutation_out<=12'b001_001_010_000; //1120
        6'd23: permutation_out<=12'b001_001_000_010; //1102
        6'd24: permutation_out<=12'b001_000_001_010; //1012
        6'd25: permutation_out<=12'b000_001_010_001; //0121
```

```
                6'd26: permutation_out<=12'b001_010_001_000; //1210
                6'd27: permutation_out<=12'b001_000_010_001; //1021
                6'd28: permutation_out<=12'b001_010_000_001; //1201

                6'd29: permutation_out<=12'b000_010_001_001; //0211
                6'd30: permutation_out<=12'b010_001_000_001; //2101
                6'd31: permutation_out<=12'b000_010_001_001; //0211
                6'd32: permutation_out<=12'b010_000_001_001; //2011
                default: permutation_out<=12'b000000000000;
            endcase
            if(counter ==31) begin
                counting <=1;
                counter<=0;
                total_counter <=total_counter + 32;
                space_to_fill <=space_to_fill - 1;
            end

        end else if (space_to_fill ==5) begin


            case(counter)
                6'b0: permutation_out<=16'b000_000_000_0100; //0005
                6'd1: permutation_out<=16'b000_000_100_0000; //0050
                6'd2: permutation_out<=16'b000_100_000_0000; //0500
                6'd3: permutation_out<=16'b100_000_000_0000; //5000
                6'd4: permutation_out<=16'b000_000_011_0001; //0041
                6'd5: permutation_out<=16'b000_011_001_0000; //0410
                6'd6: permutation_out<=16'b011_001_000_0000; //4100
                6'd7: permutation_out<=16'b000_000_001_0011; //0014
                6'd8: permutation_out<=16'b000_001_011_0000; //0140
                6'd9: permutation_out<=16'b001_011_000_0000; //1400
                6'd10: permutation_out<=16'b000_011_000_0001; //0401
                6'd11: permutation_out<=16'b011_000_001_0000; //4010
                6'd11: permutation_out<=16'b000_001_000_0011; //0104
                6'd12: permutation_out<=16'b001_000_011_0000; //1040
                6'd13: permutation_out<=16'b011_000_000_0001; //4001
                6'd14: permutation_out<=16'b001_000_000_0011; //1004
                6'd15: permutation_out<=16'b000_000_010_0010; //0023
                6'd16: permutation_out<=12'b000_010_000_0010; //0203
                6'd17: permutation_out<=12'b010_000_000_0010; //2003
                6'd18: permutation_out<=12'b010_000_010_0000; //2030
                6'd19: permutation_out<=12'b010_010_000_0000; //2300
                6'd20: permutation_out<=12'b000_000_010_0010; //0032
                6'd21: permutation_out<=12'b000_010_000_0010; //0302
                6'd22: permutation_out<=12'b010_000_000_0010; //3002
                6'd23: permutation_out<=12'b010_000_010_0000; //3020
                6'd24: permutation_out<=12'b010_010_000_0000; //3200
                6'd25: permutation_out<=12'b000_010_000_0010; //0302
                6'd26: permutation_out<=12'b010_000_000_0010; //3002
                6'd27: permutation_out<=12'b010_000_010_0000; //3020
                6'd28: permutation_out<=12'b010_010_000_0000; //3200
                6'd29: permutation_out<=12'b000_000_010_0010; //1112
                6'd30: permutation_out<=12'b000_000_010_0010; //1121
                6'd31: permutation_out<=12'b000_000_010_0010; //1211
                6'd32: permutation_out<=12'b000_000_010_0010; //2111
                6'd33: permutation_out<=12'b000_001_001_0010; //0113
                6'd34: permutation_out<=12'b001_001_010_0000; //1130
                6'd35: permutation_out<=12'b001_001_000_0010; //1103
                6'd36: permutation_out<=12'b001_000_001_0010; //1013
                6'd37: permutation_out<=12'b000_001_010_0001; //0131
                6'd38: permutation_out<=12'b001_010_001_0000; //1310
                6'd39: permutation_out<=12'b001_000_010_0001; //1031
                6'd40: permutation_out<=12'b001_010_000_0001; //1301
                6'd41: permutation_out<=12'b000_010_001_0001; //0311
                6'd42: permutation_out<=12'b010_001_000_0001; //3101
                6'd43: permutation_out<=12'b000_010_001_0001; //0311
                6'd44: permutation_out<=12'b010_000_001_0001; //3011
                6'd45: permutation_out<=12'b000_001_001_0010; //0221
                6'd46: permutation_out<=12'b001_001_010_0000; //2210
                6'd47: permutation_out<=12'b001_001_000_0010; //2201
                6'd48: permutation_out<=12'b001_000_001_0010; //2021
                6'd49: permutation_out<=12'b000_001_010_0001; //0212
                6'd50: permutation_out<=12'b001_010_001_0000; //2120
                6'd51: permutation_out<=12'b001_000_010_0001; //2012
                6'd52: permutation_out<=12'b001_010_000_0001; //2102
                6'd53: permutation_out<=12'b000_010_001_0001; //0122
                6'd54: permutation_out<=12'b010_001_000_0001; //1202
                6'd55: permutation_out<=12'b000_010_001_0001; //0122
                6'd56: permutation_out<=12'b010_000_001_0001; //1022
                default: permutation_out<=12'b000000000000;
            endcase
            if(counter ==56) begin
                counting <=1;
                counter<=0;
                total_counter <=total_counter + 57;
                space_to_fill <=space_to_fill - 1;
            end
        end //sapce to fill
    end //n of breaks
end // if coutnign
```

```
    end // always_ff
endmodule
`default_nettype wire


module iterative_solver_wth_reset(
                    input wire clk_in,

                    input wire reset_in,
                    input wire [5:0] index_in, //index of row/col beign send - max is 20 so 6 bits
                    input wire [3:0] column_number_in, //grid size - max 10
                    input wire [3:0] row_number_in, //grid size - max 10

                    input wire [19:0] assignment_in, // array of cosntraitnrs in - max of 20 btis since 4 btis * 5 slots

                    input wire start_sending_nonogram, //if asserted to 1, im in the rpcoess of sendifg the puzzle


                    output logic solution_out,
                    output logic [19:0] row1,
                    output logic [19:0] row2,
                    output logic [19:0] row3,
                    output logic [19:0] row4,
                    output logic [19:0] row5,
                    output logic [19:0] row6,
                    output logic [19:0] row7,
                    output logic [19:0] row8,
                    output logic [19:0] row9,
                    output logic [19:0] row10

    );

    logic [799:0] assignment;
    logic [31:0] address_input;
    logic [31:0] counter;
    logic started;
    logic [8:0] limit; // limit 200+200 = 400 > 9 bits

    logic [19:0] data_to_bram; // eacho column is ecnoded by 5*4 > at most 5 numbers, the biggeest is 10 (4bits)
    logic [19:0] data_from_bram;
    logic [31:0] addr;
    logic [3:0] col_number;
    logic [3:0] row_number;
    logic write;
    logic number_of_breaks;
    logic [9:0] mod_rows_in;
    logic [9:0] mod_cols_in;

    logic [9:0] solution [9:0];
    logic [31:0] addr_row; //to put all genrated rows into bram
    logic [31:0] addr_col; //to put all genrated rows into bram
    logic [3:0] counter_mod_col;

    logic [7:0] addr_row_permutation;
    logic [19:0] data_to_row_permutation_bram;
    logic [19:0] data_from_row_permutation_bram;
    logic write_permutation_row;




    logic  [7:0] addr_column_permutation;
    logic [19:0] data_to_column_permutation_bram;
    logic [19:0]data_from_column_permutation_bram;
    logic write_permutation_column;



    //stores the count of permutations for the column at each index

    logic [3:0] addr_column; //out of 10 in binaary so 4 btis

    logic write_permutation_count_column;
    logic [7:0] data_from_permutation_count_column_bram;
    logic [7:0] data_to_permutation_count_column_bram;

    logic data_to_permutation_count_row_bram;
    logic data_from_permutation_count_row_bram;
    logic write_permutation_count_row;



   logic [4:0] addr_constraint_column; //sicne at most 10 (4 bits)

   logic [19:0] data_to_column_bram;
   logic [19:0] data_from_column_bram;
   logic write_constraint_column;

   logic [19:0] data_to_row_bram;
   logic [19:0] data_from_row_bram;
```

```
logic write_constraint_row;
logic [4:0] addr_constraint_row; //sicne 10 at msot

 generate_rows my_generate_rows(
                 .clk_in(clk_in),
                 .reset_in(reset_in),
                 .start_in(start_generating),
                 .assignment(selected_assignment),
                 .outputing(generate_rows_outputing),
                 .done(done_generation),
                 .new_row(new_row_version),
                 .count(current_row_version_index),
                 .total_count(total_n_of_row_versions)
 );


 logic [19:0] new_row_version;

 logic [19:0] selected_assignment;
 logic start_generating;


 logic [2:0] n_of_constraints; //number of "numebrs" in a given array of cosntraints
 logic done_generation; //outptu from generate rows - says if they finsihed returnng all teh eprmutation for a given row
 logic new_permutation;

 logic [6:0] total_n_of_row_versions;
 logic [6:0] current_row_version_index;
 logic generate_rows_outputing;


 logic [19:0] allowable_input; //20 bits since each cell impelemned as 2 bit number



logic write_solution_row;
logic data_from_solution_row_bram;
logic data_to_solution_row_bram;
logic addr_row_solution;


logic write_solution_column;
logic data_from_solution_column_bram;
logic data_to_solution_column_bram;
logic addr_column_solution;

logic [7:0] start_address_h;



 logic [100:0] start_addresses;  //straes teh start adddres of rows/ columns at the index  i
 logic [200:0] allowable_start_addresses; //stores start adderss for each rows/col in the allwoable bram
 logic [9:0] mod_cols_in;
 logic [9:0] mod_rows_in;
 logic [19:0] c;
 logic [9:0] allowable_output_rows [19:0]; // 10 roes each is 20 len
 logic [29:0] results [19:0];
 logic results_counter;
 logic [19:0] row_assignment_collector  [9:0] ; //collects the constraint assginemtns for bram testing
 logic [19:0] col_assignment_collector [9:0]; //collects the constraint assginemtns for bram testing

 logic [19:0] row_permutation_collector [150:0]; //terrrible naming - the are ROW versions (not numbers)
 logic [19:0] col_permutation_collector [150:0]; //collects the constraint assginemtns for bram testing

 logic [15:0] row_permutation_counter [9:0]; //collbram testing - count is whtaever numer fo bits, cna be more since it will trunctae it anyways
 logic [15:0] col_permutation_counter [9:0]; //collng

 logic row_done;
 logic column_done;

 logic move_to_row_generating;

 logic counter_out;

 logic column_number;

 logic saving;

 logic move_to_solving;
 logic called_generate_rows;

 logic [6:0] addr_constraint;

 logic [8:0] row_start_addresses [9:0]; //starting address for thethe state of each row;
 logic [8:0] column_start_addresses [9:0];

 logic [19:0] can_do [9:0];

 logic  move_to_allowable_section;
```

```
    logic [8:0] allowable_counter;
    logic [19:0] allowable_result;

    logic allowable_for_a_row_started;
    logic save_allowable_result;
    logic move_to_for_loop_section;
    logic [8:0] row_counter_allowable; //at moste 10 so 4 bits

    logic [15:0] current_permutation_count;
    logic [15:0] permutation_counter_allowable_section;
    logic [4:0] fix_col_counter; //20

    logic [3:0] range_w_i;

    logic move_to_output;
    logic for_range_w_started;
    logic for_range_w_ended;
    logic [4:0] range_w_i_index; //up to 20
    logic fix_col_section_started;

    logic start_enumerate_for_loop_row;
    logic done_create_c_array;
    logic [7:0] for_c_coln_counter;
    logic [19:0] allowed_things;

    logic [4:0] allowed_thing_counter; //for 10 len
    logic [4:0] allowed_thing_index_counter; //for 20 len
    logic for_range_h_started;
    logic fix_row_section_started;
    logic for_range_h_ended;

    logic [3:0] range_h_i;

    logic [5:0] old_index;
    logic wait_on_clock;
    logic left_to_rest;
    logic [4:0] fix_col_counter_small;
    logic blob;
    logic blab;
    logic [19:0] can_do_seq;
    logic move_to_x;

    logic [6:0] tracker;
    logic [8:0] start_address;
    logic starter_marker;
    logic starter_marker_h;

    logic start_enumerate_for_loop_row_h;

    logic [5:0] range_h_i_index;
    logic [7:0] for_c_row_counter_h;


logic [4:0]    allowed_thing_counter_h;
logic [4:0] allowed_thing_index_counter_h;

logic [19:0] allowed_things_h;

logic done_create_c_array_h;
logic start_enumerate_for_loop_row_h;
logic increment_range_h;
logic increment_range_w;
logic reset_internal;
logic stop_returning;
logic collecting_puzzle_in_progress;


   always_ff @(posedge clk_in) begin
       if(reset_in) begin


           done_create_c_array_h <=0;
           start_enumerate_for_loop_row_h <=0;
           counter_out <=0;
           started<=0;
           column_number <=0;
           row_number <=0;
           write <=0;
           move_to_x<=0;
           reset_internal <=0;
           tracker <=0;
           range_h_i_index <=0;
           for_c_row_counter_h<=0;

           allowed_thing_counter_h<=0;
           allowed_thing_index_counter_h<=0;

           allowed_things_h<=0;

           start_address_h<=0;
```

```
    stop_returning <=0;

    row_done <=0;
    column_done <=0;
    addr_row_permutation <=0;
    addr_column_permutation<=0;
    mod_cols_in <=10'b1111111111;
    mod_rows_in <=10'b0;
    c <=20'b0;
    allowable_result <= 20'b0;
    allowed_things <=20'b0;

    //counters
    allowable_counter <=0;
    addr_constraint_row<=0;

    addr_constraint<=0;
    move_to_row_generating<=0;
    row_counter_allowable<=0;
    permutation_counter_allowable_section<=0;
    fix_col_counter<=0;
    range_w_i<=0;
    range_w_i_index<=0;
    allowed_thing_counter<=0; //for 10 len
    allowed_thing_index_counter<=0; //for 20 len
    range_h_i<=0;
    addr_constraint_column <=0;
    fix_col_counter_small <=0;

    start_address<=0;
    starter_marker<=0;
    starter_marker_h<=0;


    //FSM
    saving<=0;
     move_to_solving<=0;
     called_generate_rows<=0;
      move_to_allowable_section <=0;
    allowable_for_a_row_started<=0;
    save_allowable_result <=0;
    move_to_for_loop_section<=0;

    start_enumerate_for_loop_row_h<=0;

    current_permutation_count <=0;
    move_to_output<=0;
    for_range_w_started<=0;
    for_range_w_ended<=0;
    fix_col_section_started<=0;

    start_enumerate_for_loop_row <=0;
    done_create_c_array<=0;
    for_c_coln_counter<=0;
    for_range_h_started <=0;
    fix_row_section_started <=0;
   for_range_h_ended <=0;
   old_index<=0;
   selected_assignment<=0;
   wait_on_clock <=0;
   left_to_rest<=0;
   increment_range_h <=0;
   increment_range_w <=0;
   limit <=0;
   write_constraint_column<=0;
    solution_out<=0;
    row1 <=0;
    row2 <=0;
    row3 <=0;
    row4 <=0;
    row5 <=0;
    row6 <=0;
    row7 <=0;
    row8 <=0;
    row9 <=0;
    row10 <=0;
     data_to_row_bram<=0;
     collecting_puzzle_in_progress<=0;

end else begin
    if((start_sending_nonogram && ~left_to_rest) || (collecting_puzzle_in_progress && ~left_to_rest)) begin
        //we have just started sending a nongoram - send constraints line by line, first rows then columns

        //COUNTERS RESET:--------------------
        solution_out <= 0;
        mod_cols_in <=10'b1111111111;
        mod_rows_in <=10'b0;
        collecting_puzzle_in_progress<=1;
        //the reset is reset after the reurn of the solution
```

```
                //COUNTERS RESET END--------------------

                column_number <= column_number_in;
                row_number <= row_number_in;
//                saving <=1;
                //done_generation <=0; //marked as 0 if we finished generating al states for all comuns and rows
                limit <= row_number_in + column_number_in; // remebr to subtract 1 sicne we need to zero index

//            end else if (saving) begin
                if(addr_constraint_column >= 10) begin
                    //we have saved everything
                    saving<=0;
                    //addr <= 0;
                    write_constraint_column <=0;
                    write_constraint_row <=0;
                    addr_constraint_column <= 0;
                    addr_constraint_row <= 0;
                    left_to_rest<=1;
                    collecting_puzzle_in_progress<=0;
                    move_to_row_generating<=1;
                end else begin
                    if(index_in >= 10 && index_in<=19) begin
                        //SENDING COLS
                        //we are always sending
                        //we have finished sending all the rows, now we are sending all the columsn

                        write_constraint_column <=1;
                        write_constraint_row <=0;
                        addr_constraint_column <= addr_constraint_column +1;
                       data_to_column_bram  <= assignment_in;
                        col_assignment_collector[addr_constraint_column] <= assignment_in;

                    end else begin
                        //SENDING ROWS
                        //we are still sending rows

                            write_constraint_column <=0;

                            write_constraint_row <=1;
                            addr_constraint_row <= addr_constraint_row+1; // selects teh cosntrin the the to that we passed at teh beginign
                            data_to_row_bram  <= assignment_in;
                            row_assignment_collector[addr_constraint_row] <= assignment_in;


                    end
                end

            end else if(move_to_row_generating) begin
                //----------------------row generating---------------------------------
                //1) gerenate all rows for each row
                if (~wait_on_clock) begin


                    if(~row_done) begin
                        //when im currently gerneting rows
                        selected_assignment <= row_assignment_collector[addr_constraint_row];
                        write_permutation_count_column <=0;
                        row_start_addresses[addr_constraint_row] <= addr_row_permutation;

                    end else begin
                        //when im currently gerneting cols
                        //data_to_permutation_count_column_bram <= n_of_constraints;
                        selected_assignment <= col_assignment_collector[addr_constraint_column];
                        //column_start_addresses[addr_constraint_column] <= addr_column_permutation;
                        //addr_constraint_column <= addr_constraint_column+1;
                        write_permutation_count_row <=0;
                        column_start_addresses[addr_constraint_column] <= addr_column_permutation;

                    end
                    called_generate_rows <=1;
                    wait_on_clock <=1;
                end else if (called_generate_rows) begin
                     called_generate_rows <=0;
                     start_generating <=1;
                     move_to_x<=1;



                end else if (move_to_x) begin
                    start_generating <=0; // we dont need to hold it on
                    wait_on_clock<=1;
                    move_to_x <=0;



                end else if (generate_rows_outputing) begin

                    tracker <= tracker + 1;
                    start_generating <=0; //maybe?
                    //geenrate rows finally outputs
```

```
        if( ~row_done) begin
            //if im genreating rows
            addr_row_permutation <= addr_row_permutation+1;
            data_to_row_permutation_bram <= new_row_version;
            row_permutation_collector[addr_row_permutation] <= new_row_version; //
            write_permutation_column <=0;
            write_permutation_row <=1;
            write_permutation_count_row <=0;
            write_permutation_count_column <=0;
        end else if ( ~column_done) begin
        //if im gerneating columns
            addr_column_permutation <= addr_column_permutation+1;
            data_to_column_permutation_bram <= new_row_version;


            col_permutation_collector[addr_column_permutation] <= new_row_version;
            write_permutation_row <=0;
            write_permutation_column <=1;
            write_permutation_count_row <=0;
            write_permutation_count_column <=0;
        end


    end else if (done_generation && tracker >= (total_n_of_row_versions-1)  ) begin
        //generate rows finished outptuing for a gvien assignemtn - save length and mvoe to the sart of fsm
        called_generate_rows<=0; //restart the fsm
        wait_on_clock<=0;
        tracker <=0;
        //we have just consdiered the last row and wer are savign datat for the last row > change
        if(addr_constraint_row == 9) begin
        //if(addr_constraint_row == 10) begin
            row_done <=1;
            column_done <=0;
            write_permutation_row <=0;
            //addr_constraint_row = 0;

        end
        if(~row_done) begin
            //when im currently gerneting rows
            data_to_permutation_count_row_bram <= total_n_of_row_versions;

            //start_addresses[addr_constraint+1] <=addr_row;
            addr_constraint_row <= addr_constraint_row+1;
            write_permutation_count_row <=1;
            write_permutation_count_column <=0;
            row_permutation_counter[addr_constraint_row] <= total_n_of_row_versions;

        end else begin
            //when im currently gerneting cols
            data_to_permutation_count_column_bram <= n_of_constraints;
            //selected_assignment <= data_from_column_bram;
            addr_constraint_column <= addr_constraint_column+1;
            write_permutation_count_row <=0;
            write_permutation_count_column <=1;
            col_permutation_counter[addr_constraint_column] <= total_n_of_row_versions;
        end

        if(addr_constraint_column == 9) begin

            row_done <=1;
            column_done <=1;
            start_generating <=0;
            write_permutation_column <=0;
            write_permutation_row <=0;
            addr_row <= 0;
            addr_column <= 0;
            move_to_allowable_section <=1;
            move_to_row_generating <=0;
            addr_constraint <=0;
            addr_row_permutation <=0;
            addr_column_permutation <=0;

            save_allowable_result<=0;
            addr_constraint_row <=0;
            row_counter_allowable<=0;

        end

    end

end else if (move_to_allowable_section) begin

        //i have collected all nthe possibel assignements
            //2) do consolidate allowed rows (allowable function)

    if (~allowable_for_a_row_started) begin
        allowable_for_a_row_started <=1;
        permutation_counter_allowable_section <=0;
        current_permutation_count <= row_permutation_counter[row_counter_allowable];
        move_to_allowable_section<=1;
```

```
            end else if (allowable_for_a_row_started && ~ save_allowable_result) begin
                can_do_seq <=row_permutation_collector[addr_row_permutation];
                allowable_result <= allowable_result | row_permutation_collector[addr_row_permutation];
                permutation_counter_allowable_section <= permutation_counter_allowable_section+1;
                if(permutation_counter_allowable_section == (current_permutation_count-1)) begin
                    save_allowable_result <=1;
                end
                addr_row_permutation <= addr_row_permutation+1;
            end else if (save_allowable_result && allowable_for_a_row_started ) begin
                //allowable_for_a_row_started<=0;
                can_do[row_counter_allowable] <=allowable_result;
                row_counter_allowable <=row_counter_allowable +9'b1;
                save_allowable_result<=0;
                allowable_for_a_row_started <=0;
                allowable_result <=0;
                //if(row_counter_allowable == 9) begin
                if(row_counter_allowable == 9'd9) begin
                    move_to_allowable_section<=0;
                    allowable_for_a_row_started <=0;
                    save_allowable_result <=0;
                    move_to_for_loop_section<=1;
                    row_counter_allowable<=0;
                    addr_row_permutation <=0;

                end

            end


        end else if(move_to_for_loop_section) begin

            if((mod_cols_in[0] + mod_cols_in[1] +mod_cols_in[2] +mod_cols_in[3] +mod_cols_in[4] + mod_cols_in[5] +mod_cols_in[6] +mod_cols_in[7]
+mod_cols_in[8] +mod_cols_in[9]) >0) begin
                for_range_w_started <=1;
                move_to_for_loop_section<=0;
            end else begin
                move_to_output <=1;
                move_to_for_loop_section<=0;
            end

        end else if(for_range_w_started) begin

            if (range_w_i ==10) begin
                for_range_w_ended <=1;
                for_range_w_started<=0;
                range_w_i <=0;
                range_w_i_index <=0;
                starter_marker<=0;
            end else begin
                if(mod_cols_in[range_w_i]) begin
                    fix_col_section_started<=1;
                    for_range_w_started<=0;

                end else begin
                    fix_col_section_started<=0;
                    for_range_w_started<=0;
                    increment_range_w<=1;
                end

                c<=0;
                allowable_counter<=0;
                allowable_result <=0;


            end
        end else if (fix_col_section_started) begin


            if (~done_create_c_array && ~start_enumerate_for_loop_row) begin

                if(fix_col_counter == 20) begin
                    done_create_c_array <=1;
                    fix_col_counter <=0;
                    fix_col_counter_small <=0;
                    start_address <=column_start_addresses[range_w_i];

                end else begin
                    fix_col_counter <= fix_col_counter+2;
                    fix_col_counter_small <= fix_col_counter_small+1;
                    c[fix_col_counter] <= can_do[fix_col_counter_small][range_w_i_index];
                    c[fix_col_counter+1] <= can_do[fix_col_counter_small][range_w_i_index+1];
                    blob <=can_do[fix_col_counter_small][range_w_i_index];
                    blab <= can_do[fix_col_counter_small][range_w_i_index+1];
                    can_do_seq <=can_do[fix_col_counter_small];
                end
            end else if (done_create_c_array && ~start_enumerate_for_loop_row) begin
                for_c_coln_counter<=for_c_coln_counter+1;
```

```
                    //if whole row is zero that means it has been deled so you can omit it iwith if statement
                    if((col_permutation_collector[start_address + for_c_coln_counter]!=10'b0) && (
                            ( (c[1:0] & col_permutation_collector[start_address + for_c_coln_counter][1:0])     >0) &&
                            ( (c[3:2] & col_permutation_collector[start_address + for_c_coln_counter][3:2])     >0) &&
                            ( (c[5:4] & col_permutation_collector[start_address + for_c_coln_counter][5:4])     >0) &&
                            ( (c[7:6] & col_permutation_collector[start_address + for_c_coln_counter][7:6])     >0) &&
                            ( (c[9:8] & col_permutation_collector[start_address + for_c_coln_counter][9:8])     >0) &&
                            ( (c[11:10] & col_permutation_collector[start_address + for_c_coln_counter][11:10]) >0) &&
                            ( (c[13:12] & col_permutation_collector[start_address + for_c_coln_counter][13:12]) >0) &&
                            ( (c[15:14] & col_permutation_collector[start_address + for_c_coln_counter][15:14]) >0) &&
                            ( (c[17:16] & col_permutation_collector[start_address + for_c_coln_counter][17:16]) >0) &&
                            ( (c[19:18] & col_permutation_collector[start_address + for_c_coln_counter][19:18]) >0)) ) begin

                        allowed_things <= allowed_things | col_permutation_collector[start_address + for_c_coln_counter]; // range_w_i - why ?
                    end else begin
                              //fit retursn false so decrese teh avilaible numebr of permutations and zero the registr  (as deleted)
                        col_permutation_collector[start_address + for_c_coln_counter] <=0;
                    end

                    if(for_c_coln_counter == (col_permutation_counter[range_w_i]-1)) begin
                        //we finished the for loop to create resutl array
                            start_enumerate_for_loop_row <=1;
                            done_create_c_array<=0;
                            for_c_coln_counter<=0;
                    end
                end else if (start_enumerate_for_loop_row && ~done_create_c_array) begin
                    //this for loop always goes from 0 to 9 inclsuvei since to goes through the itsm in the allwoablae things which is a single arrya f
length 10

                    if(allowed_thing_counter == 10) begin
                         // to increment the  i in for loop

                        start_enumerate_for_loop_row<=0;
                        allowed_thing_counter<=0;
                        allowed_thing_index_counter<=0;
                        fix_col_section_started <=0;
                        allowed_things<=0;
                        increment_range_w<=1;


                    end else begin
                        allowed_thing_counter<=allowed_thing_counter+1; //for 10 len 10.01.01.01.01.01.01.01.01.10
                        allowed_thing_index_counter<=allowed_thing_index_counter+2; //for 20 len

                        if((allowed_things[allowed_thing_index_counter+:1] != can_do[allowed_thing_counter][range_w_i_index+:1])) begin
                            mod_rows_in[allowed_thing_counter] <= 1;
                            can_do[allowed_thing_counter][range_w_i_index] <= can_do[allowed_thing_counter][range_w_i_index] &
allowed_things[allowed_thing_index_counter];
                            can_do[allowed_thing_counter][range_w_i_index +1] <= can_do[allowed_thing_counter][range_w_i_index +1] &
allowed_things[allowed_thing_index_counter+1];
                        end
                    end


                end
            end else if (increment_range_w) begin
                for_range_w_started <=1;
                increment_range_w <=0;
                range_w_i <=range_w_i +1;
                range_w_i_index <= range_w_i_index+2;

            end else if (fix_row_section_started) begin
                //TODO - for now just increments to move forward through the for loop

                if (~done_create_c_array_h && ~start_enumerate_for_loop_row_h) begin
                    c<=can_do[range_h_i];
                    done_create_c_array_h <=1;
                    start_enumerate_for_loop_row_h<=0;
                    start_address_h <=row_start_addresses[range_h_i];

                end else if (done_create_c_array_h && ~start_enumerate_for_loop_row_h) begin
                    for_c_row_counter_h<=for_c_row_counter_h+1;


                    //if whole row is zero that means it has been deled so you can omit it iwith if statement
                    if((row_permutation_collector[start_address_h + for_c_row_counter_h]!=10'b0) && (
                            ( (c[1:0] & row_permutation_collector[start_address_h + for_c_row_counter_h][1:0])     >0) &&
                            ( (c[3:2] & row_permutation_collector[start_address_h + for_c_row_counter_h][3:2])     >0) &&
                            ( (c[5:4] & row_permutation_collector[start_address_h + for_c_row_counter_h][5:4])     >0) &&
                            ( (c[7:6] & row_permutation_collector[start_address_h + for_c_row_counter_h][7:6])     >0) &&
                            ( (c[9:8] & row_permutation_collector[start_address_h + for_c_row_counter_h][9:8])     >0) &&
                            ( (c[11:10] & row_permutation_collector[start_address_h + for_c_row_counter_h][11:10]) >0) &&
                            ( (c[13:12] & row_permutation_collector[start_address_h + for_c_row_counter_h][13:12]) >0) &&
                            ( (c[15:14] & row_permutation_collector[start_address_h + for_c_row_counter_h][15:14]) >0) &&
                            ( (c[17:16] & row_permutation_collector[start_address_h + for_c_row_counter_h][17:16]) >0) &&
                            ( (c[19:18] & row_permutation_collector[start_address_h + for_c_row_counter_h][19:18]) >0)) ) begin

                        allowed_things_h <= allowed_things_h | row_permutation_collector[start_address_h + for_c_row_counter_h]; // range_w_i - why ?
                    end else begin
```

```
                        row_permutation_collector[start_address_h + for_c_row_counter_h] <=0;

                    end

                    if(for_c_row_counter_h == (row_permutation_counter[range_h_i]-1)) begin
                        //we finished the for loop to create resutl array
                            start_enumerate_for_loop_row_h <=1;
                            done_create_c_array_h<=0;
                            for_c_row_counter_h<=0;
                    end


                end else if (start_enumerate_for_loop_row_h && ~done_create_c_array_h) begin
                     //this for loop always goes from 0 to 9 inclusive since to goes through the itsm in the allwoablae things which is a single arrya
f length 10

                    if(allowed_thing_counter_h == 10) begin
                        // to increment the  i in for loop

                        start_enumerate_for_loop_row_h<=0;
                        allowed_thing_counter_h<=0;
                        allowed_thing_index_counter_h<=0;
                        fix_row_section_started <=0;
                        allowed_things_h<=0;
                        increment_range_h<=1;
                    end else begin
                        allowed_thing_counter_h<=allowed_thing_counter_h+1; //for 10 len
                        allowed_thing_index_counter_h<=allowed_thing_index_counter_h+2; //for 20 len

                        if((allowed_things_h[allowed_thing_index_counter_h] != can_do[range_h_i][allowed_thing_index_counter_h])
                         ||  (allowed_things_h[allowed_thing_index_counter_h+1] != can_do[range_h_i][allowed_thing_index_counter_h+1])
                        ) begin
                            mod_cols_in[allowed_thing_counter_h] <= 1;
                            can_do[range_h_i][allowed_thing_index_counter_h] <= can_do[range_h_i][allowed_thing_index_counter_h] &
allowed_things_h[allowed_thing_index_counter_h];
                            can_do[range_h_i][allowed_thing_index_counter_h+1] <= can_do[range_h_i][allowed_thing_index_counter_h+1] &
allowed_things_h[allowed_thing_index_counter_h+1];
                        end
                    end
                end
            end else if (increment_range_h) begin
                for_range_h_started <=1;
                increment_range_h <=0;
                range_h_i<=range_h_i+1; //for 10 len
                range_h_i_index <= range_h_i_index+2; //for 20 len // dont increment

            end else if (for_range_w_ended) begin
                mod_cols_in<=20'b0;
                for_range_w_ended<=0;
                for_range_h_started<=1;
            end else if (for_range_h_ended) begin
                mod_rows_in<=20'b0;
                for_range_h_ended<=0;
                move_to_for_loop_section<=1;

            end else if (for_range_h_started) begin
                if (range_h_i ==10) begin
                    for_range_h_ended <=1;
                    for_range_h_started<=0;
                    range_h_i <=0;
                    range_h_i_index <=0;
                    starter_marker_h<=0; //
                end else begin
                    if(mod_rows_in[range_h_i]) begin
                        fix_row_section_started<=1;
                        for_range_h_started<=0;

                    end else begin
                        fix_row_section_started<=0;

                        increment_range_h<=1;
                        for_range_h_started<=0;
                    end

                    c<=0;

                end



            end else if (move_to_output) begin
                solution_out<=1;
                row1 <= can_do[0];
                row2 <= can_do[1];
                row3 <= can_do[2];
                row4 <= can_do[3];
                row5 <= can_do[4];
                row6 <= can_do[5];
                row7 <= can_do[6];
                row8 <= can_do[7];
```

```
        row9 <= can_do[8];
        row10 <= can_do[9];
        stop_returning <=1;
        move_to_output <=0;
    end else if (stop_returning) begin
        // just reset the whole fsm here to preapre for the new input


        done_create_c_array_h <=0;
        start_enumerate_for_loop_row_h <=0;
        counter_out <=0;
        started<=0;
        column_number <=0;
        row_number <=0;
        write <=0;
        move_to_x<=0;
        reset_internal <=0;
        tracker <=0;
        range_h_i_index <=0;
        for_c_row_counter_h<=0;

        allowed_thing_counter_h<=0;
        allowed_thing_index_counter_h<=0;

        allowed_things_h<=0;

        start_address_h<=0;
        stop_returning <=0;

        row_done <=0;
        column_done <=0;
        addr_row_permutation <=0;
        addr_column_permutation<=0;

        c <=20'b0;
        allowable_result <= 20'b0;
        allowed_things <=20'b0;

        //counters
        allowable_counter <=0;
        addr_constraint_row<=0;

        addr_constraint<=0;
        move_to_row_generating<=0;
        row_counter_allowable<=0;
        permutation_counter_allowable_section<=0;
        fix_col_counter<=0;
        range_w_i<=0;
        range_w_i_index<=0;
        allowed_thing_counter<=0; //for 10 len
        allowed_thing_index_counter<=0; //for 20 len
        range_h_i<=0;
        addr_constraint_column <=0;
        fix_col_counter_small <=0;

        start_address<=0;
        starter_marker<=0;
        starter_marker_h<=0;


        //FSM
        saving<=0;
        move_to_solving<=0;
        called_generate_rows<=0;
        move_to_allowable_section <=0;
        allowable_for_a_row_started<=0;
        save_allowable_result <=0;
        move_to_for_loop_section<=0;

        start_enumerate_for_loop_row_h<=0;

        current_permutation_count <=0;
        move_to_output<=0;
        for_range_w_started<=0;
        for_range_w_ended<=0;
        fix_col_section_started<=0;

        start_enumerate_for_loop_row <=0;
        done_create_c_array<=0;
        for_c_coln_counter<=0;
        for_range_h_started <=0;
        fix_row_section_started <=0;
        for_range_h_ended <=0;
        old_index<=0;
        selected_assignment<=0;
        wait_on_clock <=0;
        left_to_rest<=0;
        increment_range_h <=0;
        increment_range_w <=0;
        limit <=0;
```

```
                write_constraint_column<=0;
                data_to_row_bram<=0;

            end









        end

        end









endmodule
`default_nettype wire


module top_level_solver(
    input wire clk_in,
    input wire start_in, // assered when in the correct sta
    input wire reset_in,
    input wire get_output,
    input wire [15:0] sw,
    output logic sending_assignment,
    output logic [19:0] assignment_out,
    output logic [9:0] row1_out,
    output logic [9:0] row2_out,
    output logic [9:0] row3_out,
    output logic [9:0] row4_out,
    output logic [9:0] row5_out,
    output logic [9:0] row6_out,
    output logic [9:0] row7_out,
    output logic [9:0] row8_out,
    output logic [9:0] row9_out,
    output logic [9:0] row10_out,
    output logic top_level_solver_done,
    output logic assignment_out_done
   );
   translator my_translator(
                    .clk_in(clk_in),
                    .reset_in(reset_in),
                    .start_in(start_in_translator),
                    .row1(row1_in_translator),
                    .row2(row2_in_translator),
                    .row3(row3_in_translator),
                    .row4(row4_in_translator),
                    .row5(row5_in_translator),
                    .row6(row6_in_translator),
                    .row7(row7_in_translator),
                    .row8(row8_in_translator),
                    .row9(row9_in_translator),
                    .row10(row10_in_translator),
                    .row1_out(row1_out_translator),
                    .row2_out(row2_out_translator),
                    .row3_out(row3_out_translator),
                    .row4_out(row4_out_translator),
                    .row5_out(row5_out_translator),
                    .row6_out(row6_out_translator),
                    .row7_out(row7_out_translator),
                    .row8_out(row8_out_translator),
                    .row9_out(row9_out_translator),
                    .row10_out(row10_out_translator),
                    .done(translator_done));

    iterative_solver_wth_reset my_iterative_solver(
                    .clk_in(clk_in),
                    .reset_in(reset_in),
                    .index_in(counter_out), //idnex of row/col beign send - max is 20 so 6 bits
                    .column_number_in(4'd10), //grid size - max 10
                    .row_number_in(4'd10), //grid size - max 10
                    .assignment_in(assignment_in_solver), // array of cosntraitnrs in - max of 20 btis since 4 btis * 5 slots
                    .start_sending_nonogram(sending && nonogram_part), //if asserted to 1, im in the rpcoess of sendifg the puzzle
                    .solution_out(nonogram_solver_done),
                    .row1(row1_in_translator),
```

```
                .row2(row2_in_translator),
                .row3(row3_in_translator),
                .row4(row4_in_translator),
                .row5(row5_in_translator),
                .row6(row6_in_translator),
                .row7(row7_in_translator),
                .row8(row8_in_translator),
                .row9(row9_in_translator),
                .row10(row10_in_translator)

 );


 assignments_registry my_assignments_registry(
     .clk_in(clk_in),
     .reset_in(reset_in),
     .start_in(start_getting_assignment),
     .address_in(sw),
     .assignment_out(assignment_in_solver),
     .sending(sending),
     .counter_out(counter_out),
     .done(done));

logic nonogram_solver_done;
logic start_sending_nonogram;
logic start_in_translator;
logic [19:0] row1_in_translator;
logic [19:0] row2_in_translator;
logic [19:0] row3_in_translator;
logic [19:0] row4_in_translator;
logic [19:0] row5_in_translator;
logic [19:0] row6_in_translator;
logic [19:0] row7_in_translator;
logic [19:0] row8_in_translator;
logic [19:0] row9_in_translator;
logic [19:0] row10_in_translator;
logic [9:0] row1_out_translator;
logic [9:0] row2_out_translator;
logic [9:0] row3_out_translator;
logic [9:0] row4_out_translator;
logic [9:0] row5_out_translator;
logic [9:0] row6_out_translator;
logic [9:0] row7_out_translator;
logic [9:0] row8_out_translator;
logic [9:0] row9_out_translator;
logic [9:0] row10_out_translator;

logic in_progress;
logic move_to_translator;

logic [19:0] assignment_in_solver;
logic start_getting_assignment;
logic [19:0] assignment_out1;
logic sending;
logic [19:0] temporary_storage [19:0];

logic [5:0] counter_out;
logic [5:0] index;
logic take_in;
logic start_getting_output;
logic start_getting_assignment1;
logic nonogram_part;
 logic [5:0] counter_switch;


always_ff @(posedge clk_in) begin
 if(reset_in) begin

     take_in <=0;
     counter_switch <=0;


     start_sending_nonogram<=0;
     //nonogram_solver_done<=0;
     start_in_translator <=0;
     top_level_solver_done<=0;
     in_progress<=0;
     move_to_translator<=0;
     start_getting_assignment <=0;
     assignment_out1<=0;
     //counter_out <=0;
     index <=0;
     start_getting_output <= 0;
     start_getting_assignment1 <= 0;
     sending_assignment<=0;
     nonogram_part <=0;

 end else begin
     assignment_out_done <=0;
     if(get_output && ~start_getting_output) begin
```

```
    start_getting_output <= 1;
    start_getting_assignment <= 1;
    index <= 0;
    row1_out <=0;
    row2_out <=0;
    row3_out <=0;
    row4_out <=0;
    row5_out <=0;
    row6_out <=0;
    row7_out <=0;
    row8_out <=0;
    row9_out <=0;
    row10_out <=0;
    top_level_solver_done <=0;
    assignment_out_done <= 0;
end else if (sending && start_getting_output) begin
    counter_switch <=counter_switch+1;

    if((assignment_in_solver[7:4]>0) && (counter_switch) > 9) begin
        assignment_out <= {assignment_in_solver[19:8],assignment_in_solver[3:0],assignment_in_solver[7:4]};
    end else begin
        assignment_out <= assignment_in_solver;
    end



    start_getting_assignment1 <= 1;
    sending_assignment<=1;
    start_getting_assignment <= 0;

end else if (~sending && start_getting_assignment1) begin
    start_getting_assignment1 <= 0;
    assignment_out_done <= 1;
    start_getting_output <= 0;
    sending_assignment <=0;
    counter_switch <=0;

end

if(start_in && ~in_progress) begin
    start_getting_assignment <=1;
    in_progress <=1;
    index <=0;
    nonogram_part <=1;
    row1_out <=0;
    row2_out <=0;
    row3_out <=0;
    row4_out <=0;
    row5_out <=0;
    row6_out <=0;
    row7_out <=0;
    row8_out <=0;
    row9_out <=0;
    row10_out <=0;
    top_level_solver_done <=0;
    assignment_out_done <= 0;

end else if (sending && in_progress) begin
    start_sending_nonogram<=1;
    nonogram_part <=1;
    //assignment_in_solver <=assignment_out1;

    index <=index + 1;
    start_getting_assignment <=0;

    //index_in_solver <= index_out_registry;
end else if (~sending && start_sending_nonogram ) begin
    start_sending_nonogram<=0;
    take_in <=0;
    nonogram_part <=0;

end else if (nonogram_solver_done && ~move_to_translator) begin
        start_in_translator <=1;
        move_to_translator <=1;
end else if (~translator_done && move_to_translator) begin
    start_in_translator <=0; // free the button

end else if (translator_done && move_to_translator) begin
    move_to_translator <=0;
    start_in_translator <=0;
    top_level_solver_done <=1;
    row1_out <=row1_out_translator;
    row2_out <=row2_out_translator;
    row3_out <=row3_out_translator;
    row4_out <=row4_out_translator;
    row5_out <=row5_out_translator;
    row6_out <=row6_out_translator;
    row7_out <=row7_out_translator;
    row8_out <=row8_out_translator;
```

```
            row9_out <=row9_out_translator;
            row10_out <=row10_out_translator;

            in_progress <=0;
        end


    end

   end
endmodule


`default_nettype none
// tested - on changing input too
module translator(
                    input wire clk_in,
                    input wire reset_in,
                    input wire start_in,
                    input wire [19:0] row1,
                    input wire [19:0] row2,
                    input wire [19:0] row3,
                    input wire [19:0] row4,
                    input wire [19:0] row5,
                    input wire [19:0] row6,
                    input wire [19:0] row7,
                    input wire [19:0] row8,
                    input wire [19:0] row9,
                    input wire [19:0] row10,
                    output logic [9:0] row1_out,
                    output logic [9:0] row2_out,
                    output logic [9:0] row3_out,
                    output logic [9:0] row4_out,
                    output logic [9:0] row5_out,
                    output logic [9:0] row6_out,
                    output logic [9:0] row7_out,
                    output logic [9:0] row8_out,
                    output logic [9:0] row9_out,
                    output logic [9:0] row10_out,

                    output logic done

    );


    logic [19:0] row1_collect;
    logic [19:0] row2_collect;
    logic [19:0] row3_collect;
    logic [19:0] row4_collect;
    logic [19:0] row5_collect;
    logic [19:0] row6_collect;
    logic [19:0] row7_collect;
    logic [19:0] row8_collect;
    logic [19:0] row9_collect;
    logic [19:0] row10_collect;


    logic [9:0] row1_assign;
    logic [9:0] row2_assign;
    logic [9:0] row3_assign;
    logic [9:0] row4_assign;
    logic [9:0] row5_assign;
    logic [9:0] row6_assign;
    logic [9:0] row7_assign;
    logic [9:0] row8_assign;
    logic [9:0] row9_assign;
    logic [9:0] row10_assign;
    logic done_collecting;
    logic start_returning;

    logic [4:0] i; //4 btis since max 20 (i binary)
    logic [4:0] i_big;
    logic in_progress;


    always_ff @(posedge clk_in) begin

        if(reset_in) begin
            i<=0;
            i_big<=0;
            done_collecting <=0;
            start_returning <=0;
            row1_assign<=0;
            row2_assign<=0;
            row3_assign<=0;
            row4_assign<=0;
            row5_assign<=0;
            row6_assign<=0;
            row7_assign<=0;
            row8_assign<=0;
```

```
        row9_assign<=0;
        row10_assign<=0;
        row1_collect <=0;
        row2_collect <=0;
        row3_collect <=0;
        row4_collect <=0;
        row5_collect <=0;
        row6_collect <=0;
        row7_collect <=0;
        row8_collect <=0;
        row9_collect <=0;
        row10_collect <=0;
        in_progress<=0;

    end else begin


        if(start_in && ~in_progress) begin
            row1_collect<=row1;
            row2_collect<=row2;
            row3_collect<=row3;
            row4_collect<=row4;
            row5_collect<=row5;
            row6_collect<=row6;
            row7_collect<=row7;
            row8_collect<=row8;
            row9_collect<=row9;
            row10_collect<=row10;
            done_collecting <=1;
            //reset states on new input
            row1_assign<=0;
            row2_assign<=0;
            row3_assign<=0;
            row4_assign<=0;
            row5_assign<=0;
            row6_assign<=0;
            row7_assign<=0;
            row8_assign<=0;
            row9_assign<=0;
            row10_assign<=0;
            i<=0;
            i_big<=0;
            start_returning <=0;
            done <=0;
            in_progress<=1;

        end else if (done_collecting) begin
            if(i == 9) begin

                done_collecting <=0;
                start_returning <=1;
            end
            i<=i+1;
            i_big <= i_big + 2;
            if((row1_collect[i_big] == 1) && (row1_collect[i_big+1] == 0))begin
                row1_assign[i] <= 1'b1;

            end else begin
                row1_assign[i] <= 1'b0;
            end
            if((row2_collect[i_big] == 1) && (row2_collect[i_big+1] == 0))begin
                row2_assign[i] <= 1'b1;

            end else begin
                row2_assign[i] <= 1'b0;
            end
            if((row3_collect[i_big] == 1) && (row3_collect[i_big+1] == 0))begin
                row3_assign[i] <= 1'b1;

            end else begin
                row3_assign[i] <= 1'b0;
            end
            if((row4_collect[i_big] == 1) && (row4_collect[i_big+1] == 0))begin
                row4_assign[i] <= 1'b1;

            end else begin
                row4_assign[i] <= 1'b0;
            end
            if((row5_collect[i_big] == 1) && (row5_collect[i_big+1] == 0))begin
                row5_assign[i] <= 1'b1;

            end else begin
                row5_assign[i] <= 1'b0;
            end
            if((row6_collect[i_big] == 1) && (row6_collect[i_big+1] == 0))begin
                row6_assign[i] <= 1'b1;

            end else begin
                row6_assign[i] <= 1'b0;
            end
```

```
                if((row7_collect[i_big] == 1) && (row7_collect[i_big+1] == 0))begin
                    row7_assign[i] <= 1'b1;

                end else begin
                    row7_assign[i] <= 1'b0;
                end
                if((row8_collect[i_big] == 1) && (row8_collect[i_big+1] == 0))begin
                    row8_assign[i] <= 1'b1;

                end else begin
                    row8_assign[i] <= 1'b0;
                end
                if((row9_collect[i_big] == 1) && (row9_collect[i_big+1] == 0))begin
                    row9_assign[i] <= 1'b1;

                end else begin
                    row9_assign[i] <= 1'b0;
                end
                if((row10_collect[i_big] == 1) && (row10_collect[i_big+1] == 0))begin
                    row10_assign[i] <= 1'b1;

                end else begin
                    row10_assign[i] <= 1'b0;
                end



            end else if (start_returning) begin
                done <=1;
                row1_out <=row1_assign;
                row2_out <=row2_assign;
                row3_out <=row3_assign;
                row4_out <=row4_assign;
                row5_out <=row5_assign;
                row6_out <=row6_assign;
                row7_out <=row7_assign;
                row8_out <=row8_assign;
                row9_out <=row9_assign;
                row10_out <=row10_assign;
                in_progress<=0;
            end




        end


    end


endmodule
`default_nettype wire


module camera_read_fresh(
    input  p_clock_in,
    input  vsync_in,
    input  href_in,
    input wire button_press,
    input  [7:0] p_data_in,
    output logic [15:0] pixel_data_out,
    output logic pixel_valid_out,
    output logic frame_done_out
    );


    logic [1:0] FSM_state = 0;
    logic pixel_half = 0;

    localparam WAIT_BUTTON_PRESS = 2'b00;
    localparam WAIT_FRAME_START = 2'b10;
    localparam ROW_CAPTURE = 2'b01;


    always_ff@(posedge p_clock_in) begin
        case(FSM_state)
            WAIT_BUTTON_PRESS: begin //wait for VSYNC
                FSM_state <= (button_press) ? WAIT_FRAME_START : WAIT_BUTTON_PRESS;
                frame_done_out <= 0;
                pixel_half <= 0;
                pixel_valid_out <=0;
            end

            WAIT_FRAME_START: begin //wait for VSYNC
                FSM_state <= (!vsync_in) ? ROW_CAPTURE : WAIT_FRAME_START;
                frame_done_out <= 0;
                pixel_half <= 0;
```

```
            end

            ROW_CAPTURE: begin
            FSM_state <= vsync_in ? WAIT_BUTTON_PRESS : ROW_CAPTURE;
            frame_done_out <= vsync_in ? 1 : 0;
            pixel_valid_out <= (href_in && pixel_half) ? 1 : 0;
            if (href_in) begin
                pixel_half <= ~ pixel_half;
                if (pixel_half) pixel_data_out[7:0] <= p_data_in;
                else pixel_data_out[15:8] <= p_data_in;
            end
            end
        endcase
    end

endmodule


module debounce (input reset_in, clock_in, noisy_in,
                 output reg clean_out);
   reg [19:0] count;
   reg new_input;
//   always_ff @(posedge clock_in)
//     if (reset_in) begin new <= noisy_in; clean_out <= noisy_in; count <= 0; end
//     else if (noisy_in != new) begin new <= noisy_in; count <= 0; end
//     else if (count == 650000) clean_out <= new;
//     else count <= count+1;
   always_ff @(posedge clock_in)
     if (reset_in) begin
        new_input <= noisy_in;
        clean_out <= noisy_in;
        count <= 0; end
     else if (noisy_in != new_input) begin new_input<=noisy_in; count <= 0; end
     else if (count == 650000) clean_out <= new_input;
     else count <= count+1;

endmodule

module xvga(input vclock_in,
            output reg [10:0] hcount_out,    // pixel number on current line
            output reg [9:0] vcount_out,     // line number
            output reg vsync_out, hsync_out,
            output reg blank_out);
   parameter DISPLAY_WIDTH  = 1024;      // display width
   parameter DISPLAY_HEIGHT = 768;       // number of lines
   parameter  H_FP = 24;                 // horizontal front porch
   parameter  H_SYNC_PULSE = 136;        // horizontal sync
   parameter  H_BP = 160;                // horizontal back porch
   parameter  V_FP = 3;                  // vertical front porch
   parameter  V_SYNC_PULSE = 6;          // vertical sync
   parameter  V_BP = 29;                 // vertical back porch
   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount_out == (DISPLAY_WIDTH -1));
   assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1));  //1047
   assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1));  // 1183
   assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1));  //1343
   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1));   // 767
   assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1));  // 771
   assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE - 1));  // 777
   assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + V_BP - 1)); // 805
   // sync and blanking
   wire 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;  // active low
      vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
      vblank <= next_vblank;
      vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;  // active low
      blank_out <= next_vblank | (next_hblank & ~hreset);
   end

endmodule

//12/07 - tets fsm on gernation of constrins since it get stuck
module top_level_fresher_testing(
   input clk_100mhz,
   input[15:0] sw,
   input btnc, btnu, btnl, btnr, btnd,
   input [7:0] ja, //pixel data from camera
   input [2:0] jb, //other data from camera (including clock return)
   output   jbclk, //clock FPGA drives the camera with
```

```
   input [2:0] jd,
   output   jdclk,
   output[3:0] vga_r,
   output[3:0] vga_b,
   output[3:0] vga_g,
   output vga_hs,
   output vga_vs,
   output led16_b, led16_g, led16_r,
   output led17_b, led17_g, led17_r,
   output[15:0] led,
   output uart_rxd_out,
   output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
   output[7:0] an    // Display location 0-7
   );

    logic something;
    logic on;
    logic [319:0] digi_photo [219:0];
    logic [39:0] rescaled_out;

    logic clk_65mhz;
    assign clk_65mhz = clk_100mhz;
    // create 65mhz system clock, happens to match 1024 x 768 XVGA timing
    //clk_wiz_lab3 clkdivider(.clk_in1(clk_100mhz), .clk_out1(clk_65mhz));
    wire [31:0] data;      //  instantiate 7-segment display; display (8) 4-bit hex
    wire [6:0] segments;
    assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];
    //display_8hex display(.clk_in(clk_65mhz),.data_in(data), .seg_out(segments), .strobe_out(an));
    //assign seg[6:0] = segments;
    assign  dp = 1'b1;  // turn off the period
    assign led = {state_FSM,state_pixel};                       // turn leds on to check pixel state
    assign data = {state,state_FSM, state_pixel, state_UI, state_button};   // display 0123456 + sw[3:0]
    assign led16_r = btnl;                  // left button -> red led
    assign led16_g = btnc;                  // center button -> green led
    assign led16_b = btnr;                  // right button -> blue led
    assign led17_r = btnl;
    assign led17_g = btnc;

    wire [10:0] hcount;    // pixel on current line
    wire [9:0] vcount;     // line number
    wire hsync, vsync, blank;
    wire [11:0] pixel;
    reg [11:0] rgb;
    // xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.vcount_out(vcount),
    //       .hsync_out(hsync),.vsync_out(vsync),.blank_out(blank));

    // btnc button is user reset
    wire reset;
    //debounce db1(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnc),.clean_out(reset));


    logic xclk;
    logic[1:0] xclk_count;

    logic pclk_buff, pclk_in;
    logic vsync_buff, vsync_in;
    logic href_buff, href_in;
    logic[7:0] pixel_buff, pixel_in;

    logic [11:0] cam;
    logic [11:0] frame_buff_out;
    logic [15:0] output_pixels;
    logic [15:0] old_output_pixels;
    logic [12:0] processed_pixels;
    logic [3:0] red_diff;
    logic [3:0] green_diff;
    logic [3:0] blue_diff;
    logic valid_pixel;
    logic frame_done_out;

    logic [319:0] digi_photo_row;
    logic [39:0] rescaled_photo_stored [29:0];

    logic [16:0] pixel_addr_in;
    logic [16:0] pixel_addr_out;

    assign xclk = (xclk_count >2'b01);
    assign jbclk = xclk;
    assign jdclk = xclk;

    assign reset = sw[0];

//    assign red_diff = (output_pixels[15:12]>old_output_pixels[15:12])?output_pixels[15:12]-old_output_pixels[15:12]:old_output_pixels[15:12]-
output_pixels[15:12];
//    assign green_diff = (output_pixels[10:7]>old_output_pixels[10:7])?output_pixels[10:7]-old_output_pixels[10:7]:old_output_pixels[10:7]-
output_pixels[10:7];
//    assign blue_diff = (output_pixels[4:1]>old_output_pixels[4:1])?output_pixels[4:1]-old_output_pixels[4:1]:old_output_pixels[4:1]-
output_pixels[4:1];
```

```
//    blk_mem_gen_0 jojos_bram(.addra(pixel_addr_in),
//                             .clka(pclk_in),
//                             .dina(processed_pixels),
//                             .wea(valid_pixel),
//                             .addrb(pixel_addr_out),
//                             .clkb(clk_65mhz),
//                             .doutb(frame_buff_out));

    always_ff @(posedge pclk_in)begin
        if (frame_done_out)begin
            pixel_addr_in <= 17'b0;
        end else if (valid_pixel)begin
            pixel_addr_in <= pixel_addr_in +1;
        end
    end

    always_ff @(posedge clk_65mhz) begin
        pclk_buff <= jb[0];//WAS JB
        vsync_buff <= jb[1]; //WAS JB
        href_buff <= jb[2]; //WAS JB
        pixel_buff <= ja;
        pclk_in <= pclk_buff;
        vsync_in <= vsync_buff;
        href_in <= href_buff;
        pixel_in <= pixel_buff;
        old_output_pixels <= output_pixels;
        xclk_count <= xclk_count + 2'b01;

        if(((output_pixels[15:12] >>2) + (output_pixels[10:7]>>1) + (output_pixels[4:1]>>2))>5) begin
                  processed_pixels <= {4'b1111,4'b1111,4'b1111}; //white
       end else begin
                  processed_pixels <= {4'b0000,4'b0000,4'b0000};


              end

              if((hcount<320) &&  (vcount<240) && (frame_buff_out  == {4'b0000,4'b0000,4'b0000}) && (sw[10] ==1)) begin
                 digi_photo[vcount][hcount] <=1'b1;
                 rescaled_photo_stored[(vcount>>3)][(hcount>>3)] <=1'b1;
              end else if ((hcount<320) &&  (vcount<240) && (frame_buff_out  =={4'b1111,4'b1111,4'b1111}) && (sw[10] ==1)) begin
                 digi_photo[vcount][hcount] <=1'b0;
                 rescaled_photo_stored[(vcount>>3)][(hcount>>3)] <=1'b0;
              end

    end
//    assign pixel_addr_out = sw[2]?((hcount>>1)+(vcount>>1)*32'd320):hcount+vcount*32'd320;
    logic               clean;
    logic               old_clean;


    always_ff @(posedge clk_100mhz)begin
        old_clean <= clean;  //for rising edge detection
    end

    logic [119:0] contraint_sent_manual;
    logic [11:0] pixel_3040_manual;
    logic [11:0] pixel_10_10_manual;
    logic center_old;
    logic sending_30_40;

    manual_disp_10x10 my_manual_disp_10x10(
                   .clock(clk_65mhz),
                   .reset(reset),
                   .left(left),
                   .right(right),
                   .up(up),
                   .down(down),
                   .center(center),
                   .memory_read_start(sending_assignment),
                   .constraint_vals(assignment_out),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_10_10_manual));

    manual_disp_30_40 my_manual_disp_30_40(
                   .clock(clk_65mhz),
                  .reset(reset),
                   .left(left),
                   .right(right),
                   .up(up),
                   .down(down),
                   .center(center),
                   .start_sending_constraint(sending_30_40),
                  .constraint_vals(contraint_sent_manual),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_3040_manual));
```

```
    return_UI my_return_UI(
                   .clk_in(clk_65mhz),
                   .reset_in(reset),
                   .state({2'b00, sw[2:1]}),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_UI));

    assign pixel_addr_out = hcount+vcount*32'd320;

    logic [11:0] pixel_UI;

    //solution parser inputs
    logic reset; //reset
    logic left;
    logic top;
    logic sol_vals;
    logic done;
    logic [11:0] pixel_solution_disp;
    //solver
    logic start_solver;
    logic get_constraints;
    logic [19:0] constraint_out;
    logic [9:0] row1_out;
    logic [9:0] row2_out;
    logic [9:0] row3_out;
    logic [9:0] row4_out;
    logic [9:0] row5_out;
    logic [9:0] row6_out;
    logic [9:0] row7_out;
    logic [9:0] row8_out;
    logic [9:0] row9_out;
    logic [9:0] row10_out;

    logic output_assignment_done;
    logic [19:0] assignment_out;
    logic sending_assignment;

    logic constraints_first;

    always_comb begin


        if ((state_pixel == GET_CAMERA_OUTPUT) &&((hcount<320) &&  (vcount<240))) begin
            cam = frame_buff_out;

        end else if (state_pixel ==DISPLAY_MANUAL_30_40) begin
            cam = pixel_3040_manual;

        end else if (state_pixel == DISPLAY_MANUAL_10_10) begin
            cam = pixel_10_10_manual; // probably sth else sith ther eis a manual modue ?


        end else if ((state_pixel == DISPLAY_OPTIONS_10_10)) begin
            cam = pixel_solution_disp;

        end else if ((state_FSM == IDLE) &&((hcount<512) &&  (vcount<384))) begin
            cam = pixel_UI;

        end else if ((state_pixel== DISPLAY_DIGI_PHOTO) &&((hcount<320) &&  (vcount<240))) begin
            if (digi_photo[vcount][hcount] == 1'b1) begin
                cam = {4'b0000,4'b0000,4'b0000};
            end else begin
                cam = {4'b1111,4'b1111,4'b1111};
            end

        end else if ((state_pixel == DISPLAY_RESCALED_PHOTO)&&((hcount<40) &&  (vcount<30))) begin
            if (rescaled_photo_stored[vcount][hcount] == 1'b1) begin
                cam = {4'b0000,4'b0000,4'b0000}; // black
            end else if (rescaled_photo_stored[vcount][hcount] == 1'b0) begin
                cam = {4'b1111,4'b1111,4'b1111}; //white
            end else begin
                cam = {4'b1111,4'b0000,4'b1111};
            end
//       end else if ((state_FSM == SOLVER_STATE)) begin
//            cam = frame_buff_out;
       end else if ((state_FSM == DISPLAY_EMPTY_NONOGRAM)) begin
            cam = {4'b0000,4'b0000,4'b1111};
       end else begin
            cam = {4'b0000,4'b0000,4'b1111};
       end
    end


    //1024x768


//         solved_disp mysolved_disp(
//                .clock(clk_65mhz),
```

```
//                .reset(reset),
//                .solver_done(solver_done), //solver done
//                .memory_read_start(sending_assignment),
//                .constraint_vals(assignment_out),

//                .grid_vals1(row1_out),
//                .grid_vals2(row2_out),
//                .grid_vals3(row3_out),
//                .grid_vals4(row4_out),
//                .grid_vals5(row5_out),
//                .grid_vals6(row6_out),
//                .grid_vals7(row7_out),
//                .grid_vals8(row8_out),
//                .grid_vals9(row9_out),
//                .grid_vals10(row10_out),
//                .hcount(hcount),
//                .vcount(vcount),
//                .switch(sw),
//                .pixel_out(pixel_solution_disp));



   assign pixel_addr_out = hcount+vcount*32'd320;


//     top_level_solver my_top_level_solver(
//                      .clk_in(clk_65mhz),
//                      .start_in(start_solver), // assered when in the correct stata
//                      .reset_in(reset),
//                      .get_output(get_constraints),
//                      .sw(sw),
//                      .assignment_out(assignment_out),
//                      .row1_out(row1_out),
//                      .row2_out(row2_out),
//                      .row3_out(row3_out),
//                      .row4_out(row4_out),
//                      .row5_out(row5_out),
//                      .row6_out(row6_out),
//                      .row7_out(row7_out),
//                      .row8_out(row8_out),
//                      .row9_out(row9_out),
//                      .row10_out(row10_out),
//                      .top_level_solver_done(solver_done),
//                      .assignment_out_done(output_assignment_done),
//                      .sending_assignment(sending_assignment)
//    );



//    camera_read_fresh  my_camera(.p_clock_in(pclk_in),
//                            .vsync_in(vsync_in),
//                            .href_in(href_in),
//                            .p_data_in(pixel_in),
//                            .button_press(up),
//                            .pixel_data_out(output_pixels),
//                            .pixel_valid_out(valid_pixel),
//                            .frame_done_out(frame_done_out));

    // UP and DOWN buttons for pong paddle
    wire up,down,right,center;
//    debounce db2(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnu),.clean_out(up));
//    debounce db3(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnd),.clean_out(down));
//    debounce db4(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnl),.clean_out(left));
//    debounce db5(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnr),.clean_out(right));
//    debounce db6(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnc),.clean_out(center));
    wire phsync,pvsync,pblank;
    logic [11:0] fake_pixel;
    logic [119:0] constraint_generator_storage [69:0];
    wire border = (hcount==0 | hcount==1023 | vcount==0 | vcount==767 |
                   hcount == 512 | vcount == 384);

    logic filter_outputing;
    logic filter_done;
    logic start_filter;
    logic generator_done, start_generator;
    logic [39:0] constrain_input;

//    );

//    logic [39:0] column_constraint_storage [39:0];
//    logic [39:0] row_constraint_storage [29:0];
    logic [119:0] constraint_storage [69:0];

    parameter [5:0] height = 30;
    parameter [5:0] width = 40;
    logic [5:0] run_length;
    logic [5:0] constraint_count; //counts the number of cosntraints in the given row
    logic [5:0] constraint_count_index;
    logic [119:0] temporary_constraint;
```

```
    logic [8:0]  constraint_counter;

    logic process_columns;
//    logic [5:0] i;
//    logic [5:0] j;

    logic process_rows;

    logic run_started;
    logic output_constraints;
    logic move_to_output;
    logic save_constraint;
    logic move_to_columns;
    logic in_progress;
    logic [7:0] counter;
    logic [7:0] counter_out;
    logic start_collecting;
    logic last_fill;
    logic buffer;

    reg b,hs,vs;
    logic [3:0] state;

    parameter CALL_GENERATOR = 4'b0101;
    parameter DISPLAY_EMPTY_NONOGRAM = 4'b1001;
    parameter GET_CAMERA_OUTPUT = 4'b1010;
    //functionality states
    parameter IDLE = 4'b0000;
    parameter MANUAL_STATE = 4'b0001;
    parameter SOLVER_STATE = 4'b0010;
    parameter GENERATE_STATE = 4'b0011;

    //states that govern pixel outptu
    parameter DISPLAY_DIGI_PHOTO  = 4'b0111;
    parameter DISPLAY_RESCALED_PHOTO = 4'b1011;
    parameter DISPLAY_MANUAL_30_40 = 4'b1101;
    parameter DISPLAY_MANUAL_10_10 = 4'b1110;
    parameter DISPLAY_OPTIONS_10_10 = 4'b1111;




    logic [8:0] index;
    logic [8:0] index_rescale;
    logic started_filter;
    logic [7:0] index_c;
    logic generator_outputing;
    logic generator_started;
    logic [9:0] i;
    logic [9:0] j;

    logic [1:0] counter_button;

    logic [3:0] state_UI;
    logic [3:0] state_FSM;
    logic [3:0] state_pixel;
    logic center_started;
    logic start_solver;
    logic get_constraints;
    logic user_selected_nonogram;

    logic [3:0] state_button;
    logic start_constraint_old;

    logic start;
    logic start_old;

    assign start = sw[11];
    assign start_constraint = sw[6];
    logic go_to_solving;
    logic user_selected_nonogram_sw;
    logic user_selected_nonogram_sw_old;

    logic solver_done;
    assign user_selected_nonogram_sw = sw[7];

    assign led17_b  = solver_done;


    assign left = btnl;
    assign center = btnc;
    logic [7:0] counter_30_40;

    always_ff @(posedge clk_65mhz) begin

        if(reset) begin

            start_old <=0;
            start_constraint_old <=0;
            start_solver<=0;
```

```
    get_constraints <=0;
    go_to_solving <=0;
    constraints_first <= 0;
    state_FSM <=IDLE;
    user_selected_nonogram_sw_old<=0;
    state <=0;
    state_UI <=0;
    state_button <=0;
    state_pixel <=0;

    index <=0;
    index_rescale <=0;
    started_filter <= 0;
    index_c <=0;
    generator_started <=0;
    i<=0;
    j <=0;
    constraint_counter<=0;
    counter_button<=0;

    process_columns <=0;
    run_started <= 0;
    j<=0;
    i <=0;
    temporary_constraint<=120'b0;
    output_constraints <=0;
    move_to_output<=0;
    save_constraint <=0;
    move_to_columns<=0;

    process_rows <=0;
    in_progress <=0;
    constraint_count <=0;
    constraint_count_index <= 0;
    run_length <=0;
    counter <=0;
    counter_out <= 0;
    start_collecting <=0;
    user_selected_nonogram_sw_old <=0;

    last_fill <=0;
    buffer <=0;
    center_started <=0;
    user_selected_nonogram <=0;
    counter_button<=0;
    counter_30_40 <=0;
    sending_30_40 <=0;


   rescaled_photo_stored[0] <=   40'b1111111111111111111111111111111111111111;
   rescaled_photo_stored[1] <=   40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[2] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[3] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[4] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[5] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[6] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[7] <=  40'b1111111111111111111111111111111111111111;
  rescaled_photo_stored[8] <=    40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[9] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[10] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[11] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[12] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[13] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[14] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[15] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[16] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[17] <= 40'b1111111111111111111111111111111111111111;
   rescaled_photo_stored[18] <=  40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[19] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[20] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[21] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[22] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[23] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[24] <= 40'b1111111111111111111111111111111111111111;
  rescaled_photo_stored[25] <=   40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[26] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[27] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[28] <= 40'b1111111111111111111111111111111111111111;
    rescaled_photo_stored[29] <= 40'b1111111111111111111111111111111111111111;


end

rgb <= cam;
hs <= hsync;
vs <= vsync;
b <= blank;

   if(down) begin
```

```
        counter_button <=counter_button +1;
    end


 if (state_FSM ==DISPLAY_MANUAL_30_40) begin
     state_pixel <=DISPLAY_MANUAL_30_40;
 //todo - start sending constraints
     if (counter_30_40  == 69) begin
         sending_30_40 <=0;
     end else begin
         sending_30_40 <=1;
         counter_30_40 <=counter_30_40 +1;
         contraint_sent_manual <= constraint_storage[counter_30_40];
     end


end else if (state_FSM == IDLE &&((sw[12] ==0) && (sw[3] ==0))) begin


     if(center) begin
         state_FSM <= {2'b00, sw[2:1]};
     end

end else if (state_FSM == SOLVER_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin
     state_pixel <= DISPLAY_OPTIONS_10_10;

     //cofmir the choice of teh nonogram that you are seing rn to be solved
     if(center ) begin
         user_selected_nonogram <=1;
     end

     if(user_selected_nonogram) begin

         if (!center & center_old !=center) begin //set this to trigger on descending edge of center otherwise it overlaps?
             get_constraints <=1;
         end else begin
              get_constraints <=0;
         end

         if(down) begin
             user_selected_nonogram <=0;
         end
         if (!start & start_old != start) begin
            start_solver <=1;
            if(solver_done)begin
                user_selected_nonogram <=0;
                go_to_solving<=0;
             end
         end else begin
            start_solver <=0;
         end


     end


     //get cosntraints for a new nonogram to sww

     //---------------TODO - fill in with teh fsm version for solver ----------
end else if (state_FSM == MANUAL_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin

     state_pixel <= DISPLAY_MANUAL_10_10;


    //cofmir the choice of teh nonogram that you are seing rn to be solved
     if(center) begin
         user_selected_nonogram <=1;
     end

     if(user_selected_nonogram) begin

         if (center&  !constraints_first) begin //used to be just center
             get_constraints <=1;
             constraints_first <=1;
         end else begin
              get_constraints <=0;
         end

         if(sw[4]) begin
             user_selected_nonogram <=0;
         end


     end


 end else if (state_FSM == GENERATE_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin
     // start generator
     hs <= hsync;
     vs <= vsync;
```

```
 b <= blank;
 rgb <= cam;
 //state_display <=SHOW_CAMERA;
// state_FSM <= GET_CAMERA_OUTPUT;
 state_pixel <= GET_CAMERA_OUTPUT;
 center_started <=1;

 if (left) begin
     state <= CALL_GENERATOR;

     //reset gerntor stuff on new input

     constraint_counter<=0;
     process_columns <=0;
     run_started <= 0;
     j<=0;
     i <=0;
     temporary_constraint<=120'b0;
     output_constraints <=0;
     move_to_output<=0;
     save_constraint <=0;
     move_to_columns<=0;
     buffer <=0;

     constraint_count <=0;
     constraint_count_index <= 0;
     run_length <=0;
     process_rows <=1;
 end

 if (state == CALL_GENERATOR) begin


     if (process_rows && ~save_constraint  && ~buffer) begin

         if( i == height) begin
             //dong zero run legnth here since its gonna be used in the save step
             process_rows <=0;
             move_to_columns <=1;
             run_started <=0;
             j<=0;
             i <=0;
             save_constraint <=1;
             constraint_count_index<=0;
         end else begin
             if((rescaled_photo_stored[i][j] == 1) && run_started) begin
                 run_length <=run_length +1;
                 j <= j + 1;
             end else if((rescaled_photo_stored[i][j] == 1) && ~run_started) begin
                 run_length <=6'b1;
                 j <= j + 1;
                 run_started <=1;

             end else if((rescaled_photo_stored[i][j] == 0) && run_started) begin
                 run_started <=0;

                 constraint_count_index <= 6 +constraint_count_index; // add 6 since there are 6 bots per number
                 j <= j + 1;
             //run_length <=0;
                 temporary_constraint[constraint_count_index] <=run_length[0];
                 temporary_constraint[constraint_count_index+1] <=run_length[1];
                 temporary_constraint[constraint_count_index+2] <=run_length[2];
                 temporary_constraint[constraint_count_index+3] <=run_length[3];
                 temporary_constraint[constraint_count_index+4] <=run_length[4];
                 temporary_constraint[constraint_count_index+5] <=run_length[5];

             end else if(rescaled_photo_stored[i][j] == 0 && ~run_started) begin
                 j <= j + 1;
                 run_started <=0;
             end else begin
                 j <= j + 1;
             end
             if( j == width-1) begin

                 j<=0;
                 i <= i + 1;


                 buffer <=1;
                 if (run_started) begin
                     temporary_constraint[constraint_count_index] <=run_length[0];
                     temporary_constraint[constraint_count_index+1] <=run_length[1];
                     temporary_constraint[constraint_count_index+2] <=run_length[2];
                     temporary_constraint[constraint_count_index+3] <=run_length[3];
                     temporary_constraint[constraint_count_index+4] <=run_length[4];
                     temporary_constraint[constraint_count_index+5] <=run_length[5];
                 end

             end
```

```
    end

end else if (buffer) begin
    run_started <=0;
    buffer <=0;
    run_length <=0;
    save_constraint <=1;
    if (run_started) begin
        temporary_constraint[constraint_count_index] <=run_length[0];
        temporary_constraint[constraint_count_index+1] <=run_length[1];
        temporary_constraint[constraint_count_index+2] <=run_length[2];
        temporary_constraint[constraint_count_index+3] <=run_length[3];
        temporary_constraint[constraint_count_index+4] <=run_length[4];
        temporary_constraint[constraint_count_index+5] <=run_length[5];
    end
end else if (save_constraint) begin
    run_length <=0;
    run_started <=0;
    save_constraint <=0;
    temporary_constraint <=0;
    constraint_storage[constraint_counter] <=temporary_constraint;
    //constraint_counter <= constraint_counter + 1; // coutns row/columnn
    constraint_count_index <= 0;


    if(move_to_columns) begin
        process_columns <=1;
        move_to_columns<=0;
        i<=0;
        j<=0;
        constraint_counter <= constraint_counter ;
    end else begin
        constraint_counter <= constraint_counter + 1;

    end
    if(move_to_output)begin

        output_constraints <=1;
        move_to_output<=0;
        i<=0;
        j<=0;
        state_pixel <= DISPLAY_MANUAL_30_40;
        state_FSM <= DISPLAY_MANUAL_30_40;
        center_started <=0;
    end
end else if ( process_columns && ~save_constraint) begin

    if(width == j) begin
        process_rows <=0;
        process_columns <=0;
        run_started <=0;
        j<=0;
        i <=0;
        move_to_output <=1;

        save_constraint <=1;

    end else begin
        if(rescaled_photo_stored[i][j] == 1 && run_started) begin
            run_length <=run_length +1;
            i <= i + 1;
        end else if(rescaled_photo_stored[i][j] == 1 && ~run_started) begin
            run_length <=1;
            i <= i + 1;
            run_started <=1;

        end else if(rescaled_photo_stored[i][j] == 0 && run_started) begin
            run_started <=0;
            constraint_count <=1 +constraint_count;
            constraint_count_index <= 6 +constraint_count_index; // add 6 since there are 6 bots per number
            i <= i + 1;
            //run_length <=0;
            temporary_constraint[constraint_count_index] <=run_length[0];
            temporary_constraint[constraint_count_index+1] <=run_length[1];
            temporary_constraint[constraint_count_index+2] <=run_length[2];
            temporary_constraint[constraint_count_index+3] <=run_length[3];
            temporary_constraint[constraint_count_index+4] <=run_length[4];
            temporary_constraint[constraint_count_index+5] <=run_length[5];
        end else if(rescaled_photo_stored[i][j] == 0 && ~run_started) begin
            i <= i + 1;
        end else begin
            i <= i + 1;
        end
        if( i == (height-1)) begin

            //save_constraint <=1;
            buffer <=1;
            j<=j+1;
```

```
                        i <= 0;

                        //run_started <=0;

                        if (run_started) begin
                            temporary_constraint[constraint_count_index] <=run_length[0];
                            temporary_constraint[constraint_count_index+1] <=run_length[1];
                            temporary_constraint[constraint_count_index+2] <=run_length[2];
                            temporary_constraint[constraint_count_index+3] <=run_length[3];
                            temporary_constraint[constraint_count_index+4] <=run_length[4];
                            temporary_constraint[constraint_count_index+5] <=run_length[5];
                        end
                    end
                end
            end
        end


    end else if (sw[3] == 1)begin
        rgb <= cam;
        state_pixel <= DISPLAY_DIGI_PHOTO;
        state <= {constraint_storage[1][31:0]};

        if(down) begin
            i <= i+1;
        end

        if(right) begin
            j <= j+1;
        end
    end else if (sw[12] == 1)begin
        rgb <= cam;
        state_pixel <=DISPLAY_RESCALED_PHOTO;
        state <= {rescaled_photo_stored[1][30:0]};

        if(down) begin
            i <= i+1;
        end

        if(right) begin
            j <= j+1;
        end

    end else begin
          something <=1;
          hs <= hsync;
          vs <= vsync;
          b <= blank;


    end
    center_old<=center;
    start_old <= start;
    start_constraint_old <= start_constraint;
    user_selected_nonogram_sw_old <= user_selected_nonogram_sw;
 end

// the following lines are required for the Nexys4 VGA circuit - do not change
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
//////////////////////////////////////////////////////////////////////////////////

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////////////
module top_level_fresher_fsm(
   input clk_100mhz,
   input[15:0] sw,
   input btnc, btnu, btnl, btnr, btnd,
   input [7:0] ja, //pixel data from camera
   input [2:0] jb, //other data from camera (including clock return)
   output   jbclk, //clock FPGA drives the camera with
   input [2:0] jd,
   output   jdclk,
   output[3:0] vga_r,
   output[3:0] vga_b,
   output[3:0] vga_g,
   output vga_hs,
   output vga_vs,
   output led16_b, led16_g, led16_r,
   output led17_b, led17_g, led17_r,
   output[15:0] led,
   output uart_rxd_out,
   output ca, cb, cc, cd, ce, cf, cg, dp,  // segments a-g, dp
   output[7:0] an    // Display location 0-7
   );
```

```
logic something;
logic on;
logic [319:0] digi_photo [219:0];
//logic [39:0] rescaled_out;

logic clk_65mhz;
// create 65mhz system clock, happens to match 1024 x 768 XVGA timing
clk_wiz_lab3 clkdivider(.clk_in1(clk_100mhz), .clk_out1(clk_65mhz));
wire [31:0] data;      //  instantiate 7-segment display; display (8) 4-bit hex
wire [6:0] segments;
assign {cg, cf, ce, cd, cc, cb, ca} = segments[6:0];
display_8hex display(.clk_in(clk_65mhz),.data_in(data), .seg_out(segments), .strobe_out(an));
//assign seg[6:0] = segments;
assign  dp = 1'b1;  // turn off the period
assign led = {state_FSM,state_pixel};                       // turn leds on to check pixel state
assign data = {state,state_FSM, state_pixel, state_UI, state_button};   // display 0123456 + sw[3:0]
assign led16_r = btnl;                  // left button -> red led
assign led16_g = btnc;                  // center button -> green led
assign led16_b = btnr;                  // right button -> blue led
assign led17_r = btnl;
assign led17_g = btnc;

wire [10:0] hcount;    // pixel on current line
wire [9:0] vcount;     // line number
wire hsync, vsync, blank;
wire [11:0] pixel;
reg [11:0] rgb;
xvga xvga1(.vclock_in(clk_65mhz),.hcount_out(hcount),.vcount_out(vcount),
      .hsync_out(hsync),.vsync_out(vsync),.blank_out(blank));

// btnc button is user reset
wire reset;
logic xclk;
logic[1:0] xclk_count;

logic pclk_buff, pclk_in;
logic vsync_buff, vsync_in;
logic href_buff, href_in;
logic[7:0] pixel_buff, pixel_in;

logic [11:0] cam;
logic [11:0] frame_buff_out;
logic [15:0] output_pixels;
logic [15:0] old_output_pixels;
logic [12:0] processed_pixels;
logic valid_pixel;
logic frame_done_out;

//logic [319:0] digi_photo_row;
logic [39:0] rescaled_photo_stored [29:0];

logic [16:0] pixel_addr_in;
logic [16:0] pixel_addr_out;

assign xclk = (xclk_count >2'b01);
assign jbclk = xclk;
assign jdclk = xclk;

assign reset = sw[0];



blk_mem_gen_0 jojos_bram(.addra(pixel_addr_in),
                         .clka(pclk_in),
                         .dina(processed_pixels),
                         .wea(valid_pixel),
                         .addrb(pixel_addr_out),
                         .clkb(clk_65mhz),
                         .doutb(frame_buff_out));

always_ff @(posedge pclk_in)begin
    if (frame_done_out)begin
        pixel_addr_in <= 17'b0;
    end else if (valid_pixel)begin
        pixel_addr_in <= pixel_addr_in +1;
    end
end

always_ff @(posedge clk_65mhz) begin
    pclk_buff <= jb[0];//WAS JB
    vsync_buff <= jb[1]; //WAS JB
    href_buff <= jb[2]; //WAS JB
    pixel_buff <= ja;
    pclk_in <= pclk_buff;
    vsync_in <= vsync_buff;
    href_in <= href_buff;
    pixel_in <= pixel_buff;
    old_output_pixels <= output_pixels;
    xclk_count <= xclk_count + 2'b01;
```

```
         if(((output_pixels[15:12] >>2) + (output_pixels[10:7]>>1) + (output_pixels[4:1]>>2))>5) begin
                  processed_pixels <= {4'b1111,4'b1111,4'b1111}; //white
        end else begin
                  processed_pixels <= {4'b0000,4'b0000,4'b0000};


              end

              if((hcount<320) &&  (vcount<240) && (frame_buff_out  == {4'b0000,4'b0000,4'b0000}) && (sw[10] ==1)) begin
                 digi_photo[vcount][hcount] <=1'b1;
                 if((vcount>>3) == 0 || (hcount>>3) == 0 || (vcount>>3) == 29 || (hcount>>3) == 39 ) begin
                     rescaled_photo_stored[(vcount>>3)][(hcount>>3)] <=1'b0;
                 end else  begin
                     rescaled_photo_stored[(vcount>>3)][(hcount>>3)] <=1'b1;
                 end

              end else if ((hcount<320) &&  (vcount<240) && (frame_buff_out  ==={4'b1111,4'b1111,4'b1111}) && (sw[10] ==1)) begin
                 digi_photo[vcount][hcount] <=1'b0;
                 rescaled_photo_stored[(vcount>>3)][(hcount>>3)] <=1'b0;
              end

    end
//    assign pixel_addr_out = sw[2]?((hcount>>1)+(vcount>>1)*32'd320):hcount+vcount*32'd320;
    logic               clean;
    logic               old_clean;



    logic [119:0] contraint_sent_manual;
    logic [11:0] pixel_3040_manual;
    logic [11:0] pixel_10_10_manual;
    logic center_old;
    logic sending_30_40;

    manual_disp_10x10 my_manual_disp_10x10(
                   .clock(clk_65mhz),
                   .reset(reset),
                   .left(left),
                   .right(right),
                   .up(up),
                   .down(down),
                   .center(center),
                   .memory_read_start(sending_assignment),
                   .constraint_vals(assignment_out),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_10_10_manual));

    manual_disp_30_40 my_manual_disp_30_40(
                   .clock(clk_65mhz),
                   .start_sending_photo(sending_digi_30_40),
                   .photo_in(photo_in_30_40),
                   .show_sol(sw[7]),
                  .reset(reset),
                   .left(left),
                   .right(right),
                   .up(up),
                   .down(down),
                   .center(center),
                   .start_sending_constraint(sending_30_40),
                  .constraint_vals(contraint_sent_manual),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_3040_manual));

    return_UI my_return_UI(
                   .clk_in(clk_65mhz),
                   .reset_in(reset),
                   .state({2'b00, sw[2:1]}),
                   .hcount(hcount),
                   .vcount(vcount),
                   .switch(sw),
                   .pixel_out(pixel_UI));

    assign pixel_addr_out = hcount+vcount*32'd320;

    logic [11:0] pixel_UI;
   logic [39:0] photo_in_30_40;

    //solution parser inputs
    logic reset; //reset
    logic left;
    logic top;
    logic sol_vals;
    logic done;
    logic [11:0] pixel_solution_disp;
    //solver
```

```
logic start_solver;
logic get_constraints;
logic [19:0] constraint_out;
logic [9:0] row1_out;
logic [9:0] row2_out;
logic [9:0] row3_out;
logic [9:0] row4_out;
logic [9:0] row5_out;
logic [9:0] row6_out;
logic [9:0] row7_out;
logic [9:0] row8_out;
logic [9:0] row9_out;
logic [9:0] row10_out;

logic output_assignment_done;
logic [19:0] assignment_out;
logic sending_assignment;

logic constraints_first;

always_comb begin


    if ((state_pixel == GET_CAMERA_OUTPUT) &&((hcount<320) &&  (vcount<240))) begin
        cam = frame_buff_out;

    end else if (state_pixel ==DISPLAY_MANUAL_30_40) begin
        cam = pixel_3040_manual;

    end else if (state_pixel == DISPLAY_MANUAL_10_10) begin
        cam = pixel_10_10_manual; // probably sth else sith ther eis a manual modue ?


    end else if ((state_pixel == DISPLAY_OPTIONS_10_10)) begin
        cam = pixel_solution_disp;

    end else if ((state_FSM == IDLE) &&((hcount<512) &&  (vcount<384))) begin
        cam = pixel_UI;

    end else if ((state_pixel== DISPLAY_DIGI_PHOTO) &&((hcount<320) &&  (vcount<240))) begin
        if (digi_photo[vcount][hcount] == 1'b1) begin
            cam = {4'b0000,4'b0000,4'b0000};
        end else begin
            cam = {4'b1111,4'b1111,4'b1111};
        end

    end else if ((state_pixel == DISPLAY_RESCALED_PHOTO)&&((hcount<40) &&  (vcount<30))) begin
        if (rescaled_photo_stored[vcount][hcount] == 1'b1) begin
            cam = {4'b0000,4'b0000,4'b0000}; // black
        end else if (rescaled_photo_stored[vcount][hcount] == 1'b0) begin
            cam = {4'b1111,4'b1111,4'b1111}; //white
        end else begin
            cam = {4'b1111,4'b0000,4'b1111};
        end

   end else if ((state_FSM == DISPLAY_EMPTY_NONOGRAM)) begin
        cam = {4'b0000,4'b0000,4'b1111};
   end else begin
        cam = {4'b0000,4'b0000,4'b1111};
   end
end


//1024x768


     solved_disp mysolved_disp(
            .clock(clk_65mhz),
            .reset(reset),
            .solver_done(solver_done), //solver done
            .memory_read_start(sending_assignment),
            .constraint_vals(assignment_out),

            .grid_vals1(row1_out),
            .grid_vals2(row2_out),
            .grid_vals3(row3_out),
            .grid_vals4(row4_out),
            .grid_vals5(row5_out),
            .grid_vals6(row6_out),
            .grid_vals7(row7_out),
            .grid_vals8(row8_out),
            .grid_vals9(row9_out),
            .grid_vals10(row10_out),
            .hcount(hcount),
            .vcount(vcount),
            .switch(sw),
            .pixel_out(pixel_solution_disp));
```

```
    assign pixel_addr_out = hcount+vcount*32'd320;


     top_level_solver my_top_level_solver(
                     .clk_in(clk_65mhz),
                     .start_in(start_solver), // assered when in the correct stata
                     .reset_in(reset),
                     .get_output(get_constraints),
                     .sw(sw),
                     .assignment_out(assignment_out),
                     .row1_out(row1_out),
                     .row2_out(row2_out),
                     .row3_out(row3_out),
                     .row4_out(row4_out),
                     .row5_out(row5_out),
                     .row6_out(row6_out),
                     .row7_out(row7_out),
                     .row8_out(row8_out),
                     .row9_out(row9_out),
                     .row10_out(row10_out),
                     .top_level_solver_done(solver_done),
                     .assignment_out_done(output_assignment_done),
                     .sending_assignment(sending_assignment)
    );



    camera_read_fresh  my_camera(.p_clock_in(pclk_in),
                          .vsync_in(vsync_in),
                          .href_in(href_in),
                          .p_data_in(pixel_in),
                          .button_press(up),
                          .pixel_data_out(output_pixels),
                          .pixel_valid_out(valid_pixel),
                          .frame_done_out(frame_done_out));

     // UP and DOWN buttons for pong paddle
     wire up,down,right,center;
     debounce db2(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnu),.clean_out(up));
     debounce db3(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnd),.clean_out(down));
     debounce db4(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnl),.clean_out(left));
     debounce db5(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnr),.clean_out(right));
     debounce db6(.reset_in(reset),.clock_in(clk_65mhz),.noisy_in(btnc),.clean_out(center));
     wire phsync,pvsync,pblank;

     logic [119:0] constraint_storage [69:0];

     parameter [5:0] height = 30;
     parameter [5:0] width = 40;
     logic [5:0] run_length;
     logic [5:0] constraint_count; //counts the number of cosntraints in the given row
     logic [5:0] constraint_count_index;
     logic [119:0] temporary_constraint;
     logic [8:0]  constraint_counter;

     logic process_columns;
//     logic [5:0] i;
//     logic [5:0] j;

     logic process_rows;

     logic run_started;
     logic output_constraints;
     logic move_to_output;
     logic save_constraint;
     logic move_to_columns;
     logic in_progress;
     logic [7:0] counter;
     logic [7:0] counter_out;
     logic start_collecting;
     logic last_fill;
     logic buffer;

     reg b,hs,vs;
     logic [3:0] state;

     parameter CALL_GENERATOR = 4'b0101;
     parameter DISPLAY_EMPTY_NONOGRAM = 4'b1001;
     parameter GET_CAMERA_OUTPUT = 4'b1010;
     //functionality states
     parameter IDLE = 4'b0000;
     parameter MANUAL_STATE = 4'b0001;
     parameter SOLVER_STATE = 4'b0010;
     parameter GENERATE_STATE = 4'b0011;

     //states that govern pixel outptu
     parameter DISPLAY_DIGI_PHOTO  = 4'b0111;
     parameter DISPLAY_RESCALED_PHOTO = 4'b1011;
     parameter DISPLAY_MANUAL_30_40 = 4'b1101;
```

```
 parameter DISPLAY_MANUAL_10_10 = 4'b1110;
 parameter DISPLAY_OPTIONS_10_10 = 4'b1111;



 logic [8:0] index;
 logic [8:0] index_rescale;
 logic started_filter;
 logic [7:0] index_c;
 logic generator_outputing;
 logic generator_started;
 logic [9:0] i;
 logic [9:0] j;

 logic [1:0] counter_button;

 logic [3:0] state_UI;
 logic [3:0] state_FSM;
 logic [3:0] state_pixel;
 logic center_started;
 logic start_solver;
 logic get_constraints;
 logic user_selected_nonogram;

 logic [3:0] state_button;
 logic start_constraint_old;

 logic start;
 logic start_old;
 logic eight;
 logic old_8;

 assign start = sw[11];
 assign eight = sw[8];
 assign start_constraint = sw[6];
 logic go_to_solving;
 logic user_selected_nonogram_sw;
 logic user_selected_nonogram_sw_old;

 logic [7:0] c_const;

 logic solver_done;
 logic sending_digi_30_40;
 assign user_selected_nonogram_sw = sw[7];


 assign led17_b  = solver_done;
 logic const_done;
 logic photo_done;

 logic [7:0] counter_30_40;
logic [7:0] counter_digi_30_40;
 always_ff @(posedge clk_65mhz) begin

     if(reset) begin
         const_done <=0;
         sending_digi_30_40 <=0;
         counter_digi_30_40 <=0;



         start_old <=0;
         c_const <=0;
         start_constraint_old <=0;
         start_solver<=0;
         get_constraints <=0;
         go_to_solving <=0;
         constraints_first <= 0;
         state_FSM <=IDLE;
         user_selected_nonogram_sw_old<=0;
         state <=0;
         state_UI <=0;
         state_button <=0;
         state_pixel <=0;

         index <=0;
         index_rescale <=0;
         started_filter <= 0;
         index_c <=0;
         generator_started <=0;
         i<=0;
         j <=0;
         constraint_counter<=0;
         counter_button<=0;

         process_columns <=0;
         run_started <= 0;
         j<=0;
         i <=0;
```

```
    temporary_constraint<=120'b0;
    output_constraints <=0;
    move_to_output<=0;
    save_constraint <=0;
    move_to_columns<=0;

    process_rows <=0;
    in_progress <=0;
    constraint_count <=0;
    constraint_count_index <= 0;
    run_length <=0;
    counter <=0;
    counter_out <= 0;
    start_collecting <=0;
    user_selected_nonogram_sw_old <=0;

    last_fill <=0;
    buffer <=0;
    center_started <=0;
    user_selected_nonogram <=0;
    counter_button<=0;
    counter_30_40 <=0;
    sending_30_40 <=0;
    photo_done <=0;


end

rgb <= cam;
hs <= hsync;
vs <= vsync;
b <= blank;

   if(down) begin
       counter_button <=counter_button +1;
   end


if (state_FSM == DISPLAY_MANUAL_30_40 &&((sw[12] ==0) && (sw[3] ==0))) begin
    state_pixel <=DISPLAY_MANUAL_30_40;
//todo - start sending constraints


    if(~const_done) begin
        if (counter_30_40  == 69) begin
            sending_30_40 <=0;
            const_done <=1;


        end else begin
            sending_30_40 <=1;
            counter_30_40 <=counter_30_40 +1;
            contraint_sent_manual <= constraint_storage[counter_30_40];

        end

    end else if (~photo_done && const_done) begin
        if (counter_digi_30_40  == 29) begin
            sending_digi_30_40 <=0;
            photo_done <=1;
        end else begin
            sending_digi_30_40 <=1;
            counter_digi_30_40 <=  counter_digi_30_40 +1;
            photo_in_30_40 <= rescaled_photo_stored[ counter_digi_30_40];

        end

    end




end else if (state_FSM == IDLE &&((sw[12] ==0) && (sw[3] ==0))) begin


    if(center) begin
        state_FSM <= {2'b00, sw[2:1]};
    end

end else if (state_FSM == SOLVER_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin
    state_pixel <= DISPLAY_OPTIONS_10_10;

    //cofmir the choice of teh nonogram that you are seing rn to be solved
    if( center) begin
        user_selected_nonogram <=1;
    end

    if(user_selected_nonogram) begin
```

```
        if (!old_8 && old_8 != eight) begin //set this to trigger on descending edge of center otherwise it overlaps?
            get_constraints <=1;
        end else begin
             get_constraints <=0;
        end

        if(down) begin
            user_selected_nonogram <=0;
        end
        if (!start & start_old != start) begin
           start_solver <=1;
           if(solver_done)begin
               user_selected_nonogram <=0;
               go_to_solving<=0;
            end
        end else begin
           start_solver <=0;
        end


    end


    //get cosntraints for a new nonogram to sww

end else if (state_FSM == MANUAL_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin

    state_pixel <= DISPLAY_MANUAL_10_10;


   //cofmir the choice of teh nonogram that you are seing rn to be solved
    if(center) begin
        user_selected_nonogram <=1;
    end

    if(user_selected_nonogram) begin

        if (center&  !constraints_first) begin //used to be just center
            get_constraints <=1;
            constraints_first <=1;
        end else begin
             get_constraints <=0;
        end

        if(sw[4]) begin
            user_selected_nonogram <=0;
        end


    end

 end else if (state_FSM == GENERATE_STATE &&((sw[12] ==0) && (sw[3] ==0))) begin
    // start generator
    hs <= hsync;
    vs <= vsync;
    b <= blank;
    rgb <= cam;
    //state_display <=SHOW_CAMERA;
   // state_FSM <= GET_CAMERA_OUTPUT;
    state_pixel <= GET_CAMERA_OUTPUT;
    center_started <=1;

    if (left) begin
        state <= CALL_GENERATOR;

        //reset gerntor stuff on new input

        constraint_counter<=0;
        process_columns <=0;
        run_started <= 0;
        j<=0;
        i <=0;
        temporary_constraint<=120'b0;
        output_constraints <=0;
        move_to_output<=0;
        save_constraint <=0;
        move_to_columns<=0;
        buffer <=0;

        constraint_count <=0;
        constraint_count_index <= 0;
        run_length <=0;
        process_rows <=1;
    end

    if (state == CALL_GENERATOR) begin


        if (process_rows && ~save_constraint  && ~buffer) begin
```

```
    if( i == height) begin
        //dong zero run legnth here since its gonna be used in the save step
        process_rows <=0;
        move_to_columns <=1;
        run_started <=0;
        j<=0;
        i <=0;
        save_constraint <=1;
        constraint_count_index<=0;
    end else begin
        if((rescaled_photo_stored[i][j] == 1) && run_started) begin
            run_length <=run_length +1;
            j <= j + 1;
        end else if((rescaled_photo_stored[i][j] == 1) && ~run_started) begin
            run_length <=6'b1;
            j <= j + 1;
            run_started <=1;

        end else if((rescaled_photo_stored[i][j] == 0) && run_started) begin
            run_started <=0;

            constraint_count_index <= 6 +constraint_count_index; // add 6 since there are 6 bots per number
            j <= j + 1;
        //run_length <=0;
            temporary_constraint[constraint_count_index] <=run_length[0];
            temporary_constraint[constraint_count_index+1] <=run_length[1];
            temporary_constraint[constraint_count_index+2] <=run_length[2];
            temporary_constraint[constraint_count_index+3] <=run_length[3];
            temporary_constraint[constraint_count_index+4] <=run_length[4];
            temporary_constraint[constraint_count_index+5] <=run_length[5];

        end else if(rescaled_photo_stored[i][j] == 0 && ~run_started) begin
            j <= j + 1;
            run_started <=0;
        end else begin
            j <= j + 1;
        end
        if( j == width-1) begin

            j<=0;
            i <= i + 1;


            buffer <=1;
            if (run_started) begin
                temporary_constraint[constraint_count_index] <=run_length[0];
                temporary_constraint[constraint_count_index+1] <=run_length[1];
                temporary_constraint[constraint_count_index+2] <=run_length[2];
                temporary_constraint[constraint_count_index+3] <=run_length[3];
                temporary_constraint[constraint_count_index+4] <=run_length[4];
                temporary_constraint[constraint_count_index+5] <=run_length[5];
            end

        end



    end

end else if (buffer) begin
    run_started <=0;
    buffer <=0;
    run_length <=0;
    save_constraint <=1;
    if (run_started) begin
        temporary_constraint[constraint_count_index] <=run_length[0];
        temporary_constraint[constraint_count_index+1] <=run_length[1];
        temporary_constraint[constraint_count_index+2] <=run_length[2];
        temporary_constraint[constraint_count_index+3] <=run_length[3];
        temporary_constraint[constraint_count_index+4] <=run_length[4];
        temporary_constraint[constraint_count_index+5] <=run_length[5];
    end
end else if (save_constraint) begin
    run_length <=0;
    run_started <=0;
    save_constraint <=0;
    temporary_constraint <=0;
    constraint_storage[constraint_counter] <=temporary_constraint;
    //constraint_counter <= constraint_counter + 1; // coutns row/columnn
    constraint_count_index <= 0;


    if(move_to_columns) begin
        process_columns <=1;
        move_to_columns<=0;
        i<=0;
        j<=0;
        constraint_counter <= constraint_counter ;
    end else begin
        constraint_counter <= constraint_counter + 1;
```

```
            end
            if(move_to_output)begin

                output_constraints <=1;
                move_to_output<=0;
                i<=0;
                j<=0;
                state_pixel <= DISPLAY_MANUAL_30_40;
                state_FSM <= DISPLAY_MANUAL_30_40;
                //state_FSM <= IDLE;
                center_started <=0;
            end
        end else if ( process_columns && ~save_constraint) begin

            if(width == j) begin
                process_rows <=0;
                process_columns <=0;
                run_started <=0;
                j<=0;
                i <=0;
                move_to_output <=1;

                save_constraint <=1;

            end else begin
                if(rescaled_photo_stored[i][j] == 1 && run_started) begin
                    run_length <=run_length +1;
                    i <= i + 1;
                end else if(rescaled_photo_stored[i][j] == 1 && ~run_started) begin
                    run_length <=1;
                    i <= i + 1;
                    run_started <=1;

                end else if(rescaled_photo_stored[i][j] == 0 && run_started) begin
                    run_started <=0;
                    constraint_count <=1 +constraint_count;
                    constraint_count_index <= 6 +constraint_count_index; // add 6 since there are 6 bots per number
                    i <= i + 1;
                    //run_length <=0;
                    temporary_constraint[constraint_count_index] <=run_length[0];
                    temporary_constraint[constraint_count_index+1] <=run_length[1];
                    temporary_constraint[constraint_count_index+2] <=run_length[2];
                    temporary_constraint[constraint_count_index+3] <=run_length[3];
                    temporary_constraint[constraint_count_index+4] <=run_length[4];
                    temporary_constraint[constraint_count_index+5] <=run_length[5];
                end else if(rescaled_photo_stored[i][j] == 0 && ~run_started) begin
                    i <= i + 1;
                end else begin
                    i <= i + 1;
                end
                if( i == (height-1)) begin

                    //save_constraint <=1;
                    buffer <=1;
                    j<=j+1;
                    i <= 0;

                    //run_started <=0;

                    if (run_started) begin
                        temporary_constraint[constraint_count_index] <=run_length[0];
                        temporary_constraint[constraint_count_index+1] <=run_length[1];
                        temporary_constraint[constraint_count_index+2] <=run_length[2];
                        temporary_constraint[constraint_count_index+3] <=run_length[3];
                        temporary_constraint[constraint_count_index+4] <=run_length[4];
                        temporary_constraint[constraint_count_index+5] <=run_length[5];
                    end
                end
            end
        end
    end


end else if (sw[3] == 1)begin

//assign data = {state,   state_FSM, state_pixel, state_UI, state_button};


    rgb <= cam;
    state_pixel <= DISPLAY_DIGI_PHOTO;
    state <= {constraint_storage[c_const][31:0]};

    if (!old_8 && old_8 != eight) begin //set this to trigger on descending edge of center otherwise it overlaps?
            c_const <= c_const +1;
    end

    if(right) begin
        j <= j+1;
    end
```

```
        end else if (sw[12] == 1)begin
             rgb <= cam;
             state_pixel <=DISPLAY_RESCALED_PHOTO;
             state <= {rescaled_photo_stored[1][30:0]};

             if(down) begin
                 i <= i+1;
             end

             if(right) begin
                 j <= j+1;
             end

         end else begin
               something <=1;
               hs <= hsync;
               vs <= vsync;
               b <= blank;


         end
         old_8 <= eight;
         center_old<=center;
         start_old <= start;
         start_constraint_old <= start_constraint;
         user_selected_nonogram_sw_old <= user_selected_nonogram_sw;
      end

    // the following lines are required for the Nexys4 VGA circuit - do not change
    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
//////////////////////////////////////////////////////////////////////////////////


module display_8hex(
    input clk_in,                 // system clock
    input [31:0] data_in,         // 8 hex numbers, msb first
    output reg [6:0] seg_out,     // seven segment display output
    output reg [7:0] strobe_out   // digit strobe
    );
    localparam bits = 13;

    reg [bits:0] counter = 0;  // clear on power up

    wire [6:0] segments[15:0]; // 16 7 bit memorys
    assign segments[0]  = 7'b100_0000;  // inverted logic
    assign segments[1]  = 7'b111_1001;  // gfedcba
    assign segments[2]  = 7'b010_0100;
    assign segments[3]  = 7'b011_0000;
    assign segments[4]  = 7'b001_1001;
    assign segments[5]  = 7'b001_0010;
    assign segments[6]  = 7'b000_0010;
    assign segments[7]  = 7'b111_1000;
    assign segments[8]  = 7'b000_0000;
    assign segments[9]  = 7'b001_1000;
    assign segments[10] = 7'b000_1000;
    assign segments[11] = 7'b000_0011;
    assign segments[12] = 7'b010_0111;
    assign segments[13] = 7'b010_0001;
    assign segments[14] = 7'b000_0110;
    assign segments[15] = 7'b000_1110;

    always_ff @(posedge clk_in) begin
      // Here I am using a counter and select 3 bits which provides
      // a reasonable refresh rate starting the left most digit
      // and moving left.
      counter <= counter + 1;
      case (counter[bits:bits-2])
          3'b000: begin  // use the MSB 4 bits
                  seg_out <= segments[data_in[31:28]];
                  strobe_out <= 8'b0111_1111 ;
                end
          3'b001: begin
                  seg_out <= segments[data_in[27:24]];
                  strobe_out <= 8'b1011_1111 ;
                end
          3'b010: begin
                   seg_out <= segments[data_in[23:20]];
                   strobe_out <= 8'b1101_1111 ;
                  end
          3'b011: begin
                  seg_out <= segments[data_in[19:16]];
                  strobe_out <= 8'b1110_1111;
                end
          3'b100: begin
                  seg_out <= segments[data_in[15:12]];
```

```
                strobe_out <= 8'b1111_0111;
               end
          3'b101: begin
                  seg_out <= segments[data_in[11:8]];
                  strobe_out <= 8'b1111_1011;
                 end
          3'b110: begin
                   seg_out <= segments[data_in[7:4]];
                   strobe_out <= 8'b1111_1101;
                  end
          3'b111: begin
                  seg_out <= segments[data_in[3:0]];
                  strobe_out <= 8'b1111_1110;
                 end
       endcase
      end
endmodule
//////////////////////////////////////////////////////////////////////////////////
// Update: 8/8/2019 GH
// Create Date: 10/02/2015 02:05:19 AM
// Module Name: xvga
//
// xvga: Generate VGA display signals (1024 x 768 @ 60Hz)
//
//                              ---- HORIZONTAL -----     ------VERTICAL -----
//                              Active                    Active
//                    Freq      Video   FP  Sync   BP      Video   FP  Sync  BP
//   640x480, 60Hz    25.175    640     16    96   48       480    11   2    31
//   800x600, 60Hz    40.000    800     40   128   88       600     1   4    23
//   1024x768, 60Hz   65.000    1024    24   136  160       768     3   6    29
//   1280x1024, 60Hz  108.00    1280    48   112  248       768     1   3    38
//   1280x720p 60Hz   75.25     1280    72    80  216       720     3   5    30
//   1920x1080 60Hz   148.5     1920    88    44  148      1080     4   5    36
//
// change the clock frequency, front porches, sync's, and back porches to create
// other screen resolutions
//////////////////////////////////////////////////////////////////////////////////
module xvga(input vclock_in,
            output reg [10:0] hcount_out,    // pixel number on current line
            output reg [9:0] vcount_out,     // line number
            output reg vsync_out, hsync_out,
            output reg blank_out);
   parameter DISPLAY_WIDTH  = 1024;      // display width
   parameter DISPLAY_HEIGHT = 768;       // number of lines
   parameter  H_FP = 24;                 // horizontal front porch
   parameter  H_SYNC_PULSE = 136;        // horizontal sync
   parameter  H_BP = 160;                // horizontal back porch
   parameter  V_FP = 3;                  // vertical front porch
   parameter  V_SYNC_PULSE = 6;          // vertical sync
   parameter  V_BP = 29;                 // vertical back porch
   // horizontal: 1344 pixels total
   // display 1024 pixels per line
   reg hblank,vblank;
   wire hsyncon,hsyncoff,hreset,hblankon;
   assign hblankon = (hcount_out == (DISPLAY_WIDTH -1));
   assign hsyncon = (hcount_out == (DISPLAY_WIDTH + H_FP - 1));  //1047
   assign hsyncoff = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE - 1));  // 1183
   assign hreset = (hcount_out == (DISPLAY_WIDTH + H_FP + H_SYNC_PULSE + H_BP - 1));  //1343
   // vertical: 806 lines total
   // display 768 lines
   wire vsyncon,vsyncoff,vreset,vblankon;
   assign vblankon = hreset & (vcount_out == (DISPLAY_HEIGHT - 1));   // 767
   assign vsyncon = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP - 1));  // 771
   assign vsyncoff = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE - 1));  // 777
   assign vreset = hreset & (vcount_out == (DISPLAY_HEIGHT + V_FP + V_SYNC_PULSE + V_BP - 1)); // 805
   // sync and blanking
   wire 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;  // active low
      vcount_out <= hreset ? (vreset ? 0 : vcount_out + 1) : vcount_out;
      vblank <= next_vblank;
      vsync_out <= vsyncon ? 0 : vsyncoff ? 1 : vsync_out;  // active low
      blank_out <= next_vblank | (next_hblank & ~hreset);
   end

Endmodule
```

```
#------------------------PYTHON TESTING CODE / COE GENERATION CODE----------

from functools import reduce

def gen_row(w, s):
    """Create all patterns of a row or col that match given runs."""
    def gen_seg(o, sp):
        print("o", o)
        if not o:
            return [[2] * sp]
        return [[2] * x + o[0] + tail
                for x in range(1, sp - len(o) + 2)
                for tail in gen_seg(o[1:], sp - x)]
    a = [x[1:] for x in gen_seg([[1] * i for i in s], w + 1 - sum(s))]
    print("hello", a)

    return a


def my_gen_rows(length, setting):
    address = {i:0 for i in range(400)}
    start = [2]*length

    n_settings = len(setting)
    min_len = sum(setting) + n_settings - 1
    ans = []

    if min_len == length:
        #works - once i get the puzzle, run through all the possbiel combiantins,
        counter = 0
        for s in setting:
            print("s",s)
            for i in range(counter, s+counter):
                start[i] = 1

            counter = counter + s+1 #give a sapce break

        return [start]
    elif len(setting) == 1:
        #works
        start = [2]*setting[0] + [1]*(length - setting[0])
        limit = length - setting[0]
        offset = 0
        while limit >=0:
            print(len([2]*limit + [1]*setting[0] +[2]*offset))
            ans.append([2]*limit + [1]*setting[0] +[2]*offset )
            # print("asn", ans)
            offset +=1
            limit -=1
    else:
        #allcoation with single space
        #alcoation with 1 and 2 and 1 and 1 ect
        #alcoation with 1 and 1 and 2 and 1 ect
        n_settings = len(setting) #for 10 by 10 max is 5
        min_len = sum(setting) + n_settings - 1
        space_left = length - min_len
        breaks = n_settings
        ans = []
        # stack = [[1]*breaks]
        # while stack:
        #     ans.append(stack.pop())
        counter = 0

        for s in setting:

            for i in range(counter, s+counter):
                start[i] = 1
            print(start)

            counter = counter + s+1 #give a sapce break

        ans.append(start)

        limit = length - min_len
        offset = 0
        element = start[:min_len]
        while limit >=0:

            ans.append([2]*limit +  element +[2]*offset )
            offset +=1
            limit -=1
        shifts_applied = 1
        shift = 1
        min_len = sum(setting) + n_settings - 1
        for l in range(n_settings):
            shift = 1
```

```
            while min_len <=length:
                counter = 0
                start = [2] * length


                for i,s in enumerate(setting):



                    for j in range(counter, s+counter):
                        start[j] = 1
                    if i == l:
                        counter = counter + s+shift #give a space break
                    else:

                        counter = counter + s+1 #give a space break

                element = start[:min_len]

                limit = length - min_len
                offset = 0
                while limit >=0:
                    ans.append([2]*limit +   element +[2]*offset )
                    offset +=1
                    limit -=1
                shift = shift+1
                min_len = sum(setting) + (n_settings) + shift-1 - 1

    return ans


from functools import reduce

def gen_row(w, s):
    """Create all patterns of a row or col that match given runs."""
    def gen_seg(o, sp):
        if not o:
            return [[2] * sp]
        return [[2] * x + o[0] + tail
                for x in range(1, sp - len(o) + 2)
                for tail in gen_seg(o[1:], sp - x)]

    return [x[1:] for x in gen_seg([[1] * i for i in s], w + 1 - sum(s))]



def deduce(hr, vr):
    """Fix inevitable value of cells, and propagate."""
    def allowable(row):
        print("row", row)
        return reduce(lambda a, b: [x | y for x, y in zip(a, b)], row)

    def fits(a, b):
        return all(x & y for x, y in zip(a, b))

    def fix_col(n):
        """See if any value in a given column is fixed;
        if so, mark its corresponding row for future fixup.
        we are adding stuff to mod_row
        which are then used ni fix_col
        use a n array to mark wih 1 the indexes which were addded to fix_col > you can get theri count by just summing

        """

        c = []
        for i in range(len(can_do)):
            c.append(can_do[i][n])
        results = []
        indexes_included = []
        for index,x in enumerate(cols[n]):
            if fits(x, c):
                indexes_included.append(index)
                results.append(x)



        # print("resutls,", len(results), results)
        cols[n] = results
        allowed_things = allowable(results)


        for i, x in enumerate(allowed_things):
            if x != can_do[i][n]:

                mod_rows_in[i] = 1
                can_do[i][n] &= x


    def fix_row(n):
```

```
    """Ditto, for rows."""
    c = can_do[n]
    #print("B", rows[n]) #get all possible states of row at index n in the puzzle
    rows[n] = [x for x in rows[n] if fits(x, c)]
    for i, x in enumerate(allowable(rows[n])):
        if x != can_do[n][i]:
            mod_cols_in[i] = 1
            #mod_cols.add(i) #at index i of array mod_cols, set it to 1
            can_do[n][i] &= x


def get_bin(x):
    """
    Get the binary representation of x.
    Parameters
    ----------
    x : int
    n : int
        Minimum number of digits. If x needs less digits in binary, the rest
        is filled with zeros.
    Returns
    -------
    str
    """
    return format(x, 'b').zfill(2)

def show_gram(m):
    # If there's 'x', something is wrong.
    # If there's '?', needs more work.

    print("m", m )
    for x in m:
        row = "".join(list(map(get_bin, x)))
        # print(len(row))

        # decimal_representation = int(row, 2)
        # hexadecimal_string = hex(decimal_representation)
        # #print(row)
        # print(hexadecimal_string)
        print(" ".join("x#.?"[i] for i in x))
    print()

w, h = len(vr), len(hr)
rows = [gen_row(w, x) for x in hr]
cols = [gen_row(h, x) for x in vr]

print("lastlow", rows[-2])

print("length", list(map(len, rows)))
print("length col", list(map(len, cols)))
print("lengthsum", sum(list(map(len, rows))))
print("lengthsum-col", sum(list(map(len, cols))))

rows_len = []

can_do= []

can_d_hex = []
for i,r in enumerate(rows):
    r_new = [tuple(x) for x in r]

    for el in r:
        print(el)



    rows_len.append(len(list(set(r_new))))
    #print("allwable(r)", allowable(r))

    allowabler = allowable(r)
    ans = ""
    for i in allowabler:
        f = '{0:02b}'.format(i)
        print("3f", f)
        ans+=f
    print("ans", ans)
    can_d_hex.append(hex(int("0b" +(str(ans)),2)))
    can_do.append(allowable(r))

print("1candohexs", can_do)

# Initially mark all columns for update.
mod_rows, mod_cols = set(), set(range(w))
mod_rows_in = [0]*h
mod_cols_in = [1]*w

counter = 0
while sum(mod_cols_in) >0:
    counter+=1
    for i in range(w):
```

```
            if mod_cols_in[i]:
                fix_col(i)
        mod_cols_in = [0] * w

        print("oteration mod rows in", mod_rows_in, counter)
        for j in range(h):
            if mod_rows_in[j]:
                fix_row(j)
        mod_rows_in = [0] * h
    print("coutner", counter)



    if all(can_do[i][j] in (1, 2) for j in range(w) for i in range(h)):
        print("Solution would be unique")  # but could be incorrect!
    else:
        print("Solution may not be unique, doing exhaustive search:")
    print()
    print("AAAAAAAAAAA")
    show_gram(can_do)



def solve(s, show_runs=True):

    print("Horizontal runs:", s[0])
    print("Vertical runs:", s[1])

    #deduce(ho,ve)


    deduce(s[0],s[1])

    # deduce(
    #     [[3], [2, 1], [3, 2], [2, 2], [6], [1, 5], [6], [1], [2]],
    #        [[1, 2], [3, 1], [1, 5], [7, 1], [5], [3], [4], [3]])

ho = [[2]]
ve=  [[1],[1]]
l1 = [[3],[2,1],[2,3],[1,2,1],[2,1],[1,1],[1,3],[3,4],[4,4],[4,2]]

l2 = [
        [2],
        [4],
        [4],
        [8],
        [1,1],
        [1,1],
        [1,1,2],
        [1,1,4],
        [1,1,4],
        [9]
    ]

b1 =[[0],[3],[3],[3],[0]]
b2 =[[0],[3],[3],[3],[0]]
d1 =[[3], [1,2], [2,3], [2, 2], [6], [5,1], [6], [1], [2], [0]]
d2 =   [[2,1], [1,3], [ 5,1], [ 1,7], [5], [3], [4], [3], [0], [0]]
a1 =  [[3], [2, 1], [3, 2], [2, 2], [6], [1, 5], [6], [1], [2], [0]] #row constraitns - left to righ
a2 =  [[1, 2], [3, 1], [1, 5], [7, 1], [5], [3], [4], [3], [0], [0]] #column constraints
solve([a1,a2])



s1 = "1010_0101010101010101"
     #"1001010101010101010100"
     #"1010101010101010010"
s2 =  "1001010101010101010110"
s3 = "0101010101010101011010"
a1 = bin(int("bf7ff", 16))
print("a1",a1)
a2 = hex(int(s2, 2))
a3 = hex(int(s3, 2))

a4 = bin(int("995a5", 16))[2:]

#----------GENERATE NONOGRAM CONSTRAINTS coes ----------

from nonograms import *
import pprint
#dimensiosn are idnexed by index
#col asn row assingemtns are indexed by address
coe_hdr = '''memory_initialization_radix=2;
memory_initialization_vector=
'''
#change to 800 for a default
nonogram_max_column = 10
nonogram_max_row = 10
#size of the entry fore the colum assingemtn for ht estorign of numebrs in the binary
```

```
nonogram_entry_size =  20 #2*8 bit number (2x2 nonogram), change to 800 for a default
dimensions_entry_size= 4 #number of bits for one dimension i nthe fimensison field (so size of the whole field is 2* this)
def generate_coe(nonograms):
    #passes a list of abstraction of nonograms as the input
    #saves all of the to teh same coe (returns 2 coes for dimensions and actual abstraction)
    #"{0:b}".format(37)
    dimensions = [el[0] for el in nonograms]

    addresses = []
    row_assignments = []
    col_assignments = []
    assignments = []
    address = 0
    total_row = 0
    total_col = 0
    address_row = 0
    address_col = 0
    for d in range(len(dimensions)):
        addresses.append(format(address, '#034b')[2:])
        total_row+=dimensions[d][0]
        total_col+=dimensions[d][1]
        address += dimensions[d][0]+dimensions[d][1]
        address_col += dimensions[d][0]
        address_row += dimensions[d][1]
    print("dimensiosn", dimensions)
    print("addersses", addresses)
    input_dimensions = ["0"*16 for el in range(len(nonograms))]
    print("input", input_dimensions)
    for i,d in enumerate(dimensions):

        input_dimensions[i] = produce_given_length(str(format(d[0], '#08b'))[2:],dimensions_entry_size)  + produce_given_length(str(format(d[1], '#
08b'))[2:],dimensions_entry_size)
        print(input_dimensions[i])
    print(input_dimensions)

    #who cares about efficiceny - lets go two for loops for readability <3
    for i,n in enumerate(nonograms):
        print("create")
        rows = n[1]  #get row assingemtns for a given nonogram
        cols = n[2]  #get col assingemtns for a given nonogram
        print("rwos,cols", rows,cols)
        a = []
        r_as = []
        c_as = []
        for row in rows:
            #print("row", row)
            el1 = [format(r, '#06b')[2:] for r in row]
            #print("el", el)
            #print([format(r, '#010b')[2:] for r in row])
            el1 = produce_given_length("".join(el1),nonogram_entry_size)
            row_assignments.append(el1)
            r_as.append(el1)

        for col in cols:

            el2 = [format(c, '#06b')[2:] for c in col]
           # print("el", el)
            el2 = produce_given_length("".join(el2),nonogram_entry_size)
            col_assignments.append(el2)
            c_as.append(el2)
        assignments += r_as+ c_as #all_rows + all_columns

    print("output")
    print(col_assignments, len(col_assignments))
    print(row_assignments, len(row_assignments))
    print(assignments, len(assignments))
    with open("nonogram_dimensions.coe", "w") as f:
        f.write(coe_hdr)
        for el in input_dimensions:
            f.write( el + ",\n")
        f.close()
    with open("nonogram_address.coe", "w") as f:
        f.write(coe_hdr)
        print("add", addresses)
        for el in addresses:
            print("el", el)
            if el ==0:
                print("ZEEEEEEEEEEEERO")

                f.write( format(0, '#034b') + ",\n")
            else:
                f.write( el + ",\n")



    with open("nonogram_assignments.coe", "w") as f:
        f.write(coe_hdr)
        for row in assignments:
            f.write(row+ ",\n")
        f.close()
```

```
    with open("nonogram_assignment_rows.coe", "w") as f:
        f.write(coe_hdr)

        for row in row_assignments:
            f.write(row+ ",\n")
        f.close()
    with open("nonogram_assignment_columns.coe", "w") as f:
        f.write(coe_hdr)
        for col in col_assignments:
            f.write(col+ ",\n")
        f.close()
def visualize_nonogram(nonogram):
    pass
def produce_given_length(string,length):
    if len(string) == length:
        return string
    else:
        print(len(string))
        difference = length - len(string)
        print("difference", difference)
        add_on = "0"*difference
        return add_on + string

def decode_nonogram(index):
    with open("nonogram_address.coe", "r") as f:
        lines = f.readlines()
        addresses = lines[2:] #remove the radix header
    with open("nonogram_dimensions.coe", "r") as f:
        dimensions = f.readlines()
        dimensions = dimensions[2:] #remove the radix header
    print("ad", addresses)
    print("dims", dimensions)

    address = int(addresses[index][:-2],2)
    print("Address", address)
    dimension = dimensions[index].strip().strip(",")
    size1 = int(dimension[0:8],2)
    size2 = int(dimension[8:],2)
    start_index_row = address
    end_index_row = address + size1 -1


    start_index_column = end_index_row +1
    end_index_column = start_index_column + size2 -1
    col_assignments = []
    row_assignments = []
    with open("nonogram_assignments.coe", "r") as f:
        lines = f.readlines()
        lines = lines[2:] #remove the radix header
        counter = 0
        for line in lines:
            if counter >= start_index_row and counter <= end_index_row:

                l = len(line)

                items = l//8 #number of elements stored in the entr, this cna be hardcoded but no, divide by 8 since eahc number is encode by 8 bit
numebr (due to 200 limit)
                rows = [line[i*8:(i+1)*8] for i in range(items)]

                row_assignments.append([int(r,2) for r in rows])
            elif counter >=  start_index_column and counter <= end_index_column:
                #print("col")

                l = len(line)

                items = l//8 #number of elements stored in the entr, this cna be hardcoded but no, divide by 8 since eahc number is encode by 8 bit
numebr (due to 200 limit)
                cols = [line[i*8:(i+1)*8] for i in range(items)]

                col_assignments.append([int(r,2) for r in cols])
            elif counter == end_index_column +1:
                break
            counter+=1
        f.close()

    print("row", row_assignments)
    print("col", col_assignments)




if __name__ == "__main__":
    print()
    nonograms = [

    [[10,10],
    [[3], [1,2], [2,3], [2, 2], [6], [5,1], [6], [1], [2], [0]],
    [[2,1], [1,3], [ 5,1], [ 1,7], [5], [3], [4], [3], [0], [0]]]
```

```
]
    a = generate_coe(nonograms)
    #decode_nonogram(1)

    # return_nonogram(1)

# ------GENERATE UI coes -----------------
import sys
from PIL import Image
import math
import numpy as np
import cv2
coe_hdr = '''memory_initialization_radix=2;
memory_initialization_vector=
'''
red = (255, 0, 0)
green = (0, 255,0)
blue = (0, 0,255)
black = (0, 0,0)
black_code = "00"
red_code ="01"
blue_code ="10"
green_code ="11"
im = Image.open('IDLE.png', 'r')
width, height = im.size
print(width*height)#196608
pixel_values = list(im.getdata())
for i, pixel in enumerate(pixel_values):
    if pixel[0] >120:
        pixel_values[i] = red
    elif pixel[1] >120:
        pixel_values[i] = green
    elif pixel[2] >120:
        pixel_values[i] = blue
    else:
        pixel_values[i] = black


with open("IDLE_color_map.coe", "w") as f:
    f.write(coe_hdr)
    for i in pixel_values:
        if i ==red:
            f.write(red_code + ",\n")
        elif i == green:
            f.write(green_code + ",\n")
        elif i == blue:
            f.write(blue_code + ",\n")
        else:
            f.write(black_code + ",\n")


#----------GENERATE NUMBER coes -----------

import sys
from PIL import Image
import math
import numpy as np
import cv2
coe_hdr = '''memory_initialization_radix=2;
memory_initialization_vector=
'''


red = (255, 0, 0)
green = (0, 255,0)
blue = (0, 0,255)
black = (0, 0,0)
white = (255, 255,255)
black_code = "0"
white_code = "1"


im = Image.open("10.png", 'r')
for i in range(10,11):
    name = str(i) + ".png"
    name_output = str(i) + "_small.coe"
    im = Image.open(name, 'r')
    width, height = im.size
    print(width*height)#196608
    pixel_values = list(im.getdata())
    for i, pixel in enumerate(pixel_values):
        if pixel[0] >120:
            pixel_values[i] = black
        else:
            pixel_values[i] = white
    with open(name_output, "w") as f:
        f.write(coe_hdr)
        for i in pixel_values:
```

```
            if i ==black:
                f.write(black_code + ",\n")
            elif i == white:
                f.write(white_code + ",\n")
            # elif i == blue:
            #     f.write(blue_code + ",\n")
            # else:
            #     f.write(black_code + ",\n")


 # ------BILINEAR FILTERING TESTING --------

import math
import numpy as np
import cv2

img = cv2.imread("example.jpg", cv2.IMREAD_COLOR)
for row in img:

    for i, el in enumerate(row):
        if 0.25*el[0] + 0.5*el[1] + 0.25*el[2] >130:

            row[i] = [255, 255, 255]
        else:
            row[i] = [0, 0, 0]

    #img[index] = [el[0], el[0], el[0]]

original_img = img
new_h = 30
new_w = 40
    #get dimensions of original image
old_h, old_w, c = original_img.shape
#create an array of the desired shape.
#We will fill-in the values later.
resized = np.zeros((new_h, new_w, c))
print(old_h, old_w)
#Calculate horizontal and vertical scaling factor
w_scale_factor = (old_w ) / (new_w )
h_scale_factor = (old_h ) / (new_h )




for i in range(new_h):
    for j in range(new_w):
        #map the coordinates back to the original image
        x = i * h_scale_factor
        y = j * w_scale_factor
        #calculate the coordinate values for 4 surrounding pixels.
        # print("xy",x,y)
        x_floor = math.floor(x)
        x_ceil = min( old_h - 1, math.ceil(x))
        y_floor = math.floor(y) #smaller or equal to y so (y-y_floor) is positive or zero
        y_ceil = min(old_w - 1, math.ceil(y))

        print(x,y,x_floor,x_ceil,y_floor,y_ceil)
        if (x_ceil == x_floor) and (y_ceil == y_floor):

            q = original_img[int(x), int(y), :]

        elif (x_ceil == x_floor):

            q1 = original_img[int(x), int(y_floor), :]
            q2 = original_img[int(x), int(y_ceil), :]
            print("q1,q2", q1, q2)
            q = q1 * (y_ceil - y) + q2 * (y - y_floor)
        elif (y_ceil == y_floor):

            q1 = original_img[int(x_floor), int(y), :]
            q2 = original_img[int(x_ceil), int(y), :]
            print("q1,q2", q1, q2)
            q = (q1 * (x_ceil - x)) + (q2    * (x - x_floor))
        else:

            v1 = original_img[x_floor, y_floor, :]
            v2 = original_img[x_ceil, y_floor, :]
            v3 = original_img[x_floor, y_ceil, :]
            v4 = original_img[x_ceil, y_ceil, :]

            q1 = v1 * (x_ceil - x) + v2 * (x - x_floor)
            q2 = v3 * (x_ceil - x) + v4 * (x - x_floor)
            q = q1 * (y_ceil - y) + q2 * (y - y_floor)
            print("q", q)


        resized[i,j,:] = q
ans = resized.astype(np.uint8)
bit_image = []
```

```
for row in ans:
    st_row = ""
    for el in row:
        if sum(el) >0:
            st_row += "0"
        else:
            st_row += "1"
    bit_image.append(st_row)
print(bit_image)
with open("bit_image.txt", "w") as f:
    for i,el in enumerate(bit_image):
        f.write("image_in[" +str(i) + "] = " + el + ";\n")
    f.close()
```