University of Essex
CE264 Digital Systems Design

VGA Controller Design Report

Prepared by
Michal Borowski
1904535
mb19424@essex.ac.uk

The purpose of this report is to present the VGA-driver design created as a part of “CE264 Digital Systems Design” module coursework. The aim of the design is to generate controllable colourful pattern (3x4 squares) on the screen using VGA port of “Digilent Basys 3” FPGA board. The design was implemented using “Multisim” and “Vivado” software. Creation of this design involved 3 guided stages [1] [2] [3]. All 3 stages were completed, of which results are shown in “Performance and tests” section.

31st January 2020

Table of Contents

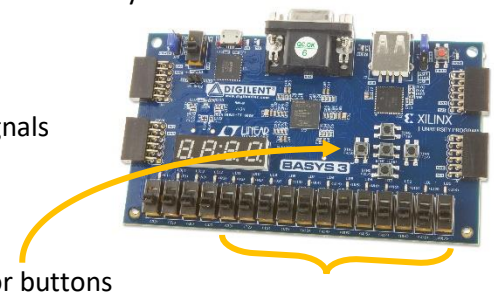
List of Figures

List of Tables

1. Design requirements

1.1 Overview

The main goal of the design is to manipulate colours of VGA display split into 3x4 regions. This must be accomplished by the Basys 3 FPGA board [4] programmed in a way to:



- generate signals to run VGA display, including:
  - 25MHz pixel clock signal
  - horizontal and vertical synchronization signals
  - blanking signal
  - 12-bit colour signals
- store the colour of each region of the screen
- select the current region of the screen by its cursor buttons
- overwrite the selected region colour (value of colour being a copy of 12 switches state) upon central button press

According to the "Laboratory Stage 3" document [3] the currently selected region should be indicated by the 7-segment display, or/and by flashing it on the screen.

1.2 Clock

Basys 3 board contains a 100MHz oscillator which could be used to create the desired 25MHz pixel clock and 500Hz clock for driving the 7-segment display as suggested in the "Laboratory Stage 1" document [1].

1.3 Synchronisation signals

As described in the "Laboratory Stage 2" document [2] the synchronisation signals must incorporate "front porch" and "back porch" periods for the sake of maintaining compability with the VGA interface (initially limited by physical capabilities of monitors). This is illustrated in Figure 1.

1.4 Blanking signal

During the "Blanking Time" (as seen in Figure 1) the signals representing RGB values should be set to LOW.

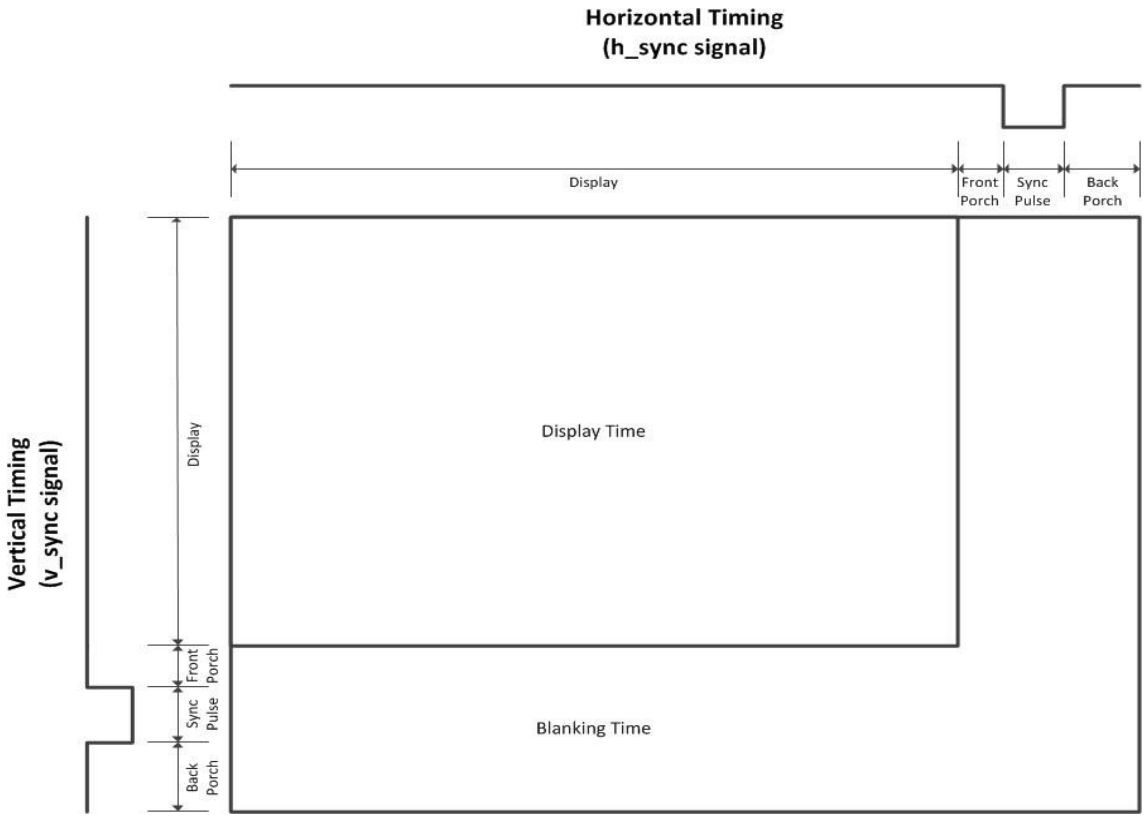


Figure 1. Horizontal and vertical timing signals illustration. [5]

2. Design description

2.1 Overview

At the top level, the design consists of 7 hierarchical blocks. Relationships between these blocks are shown in Figure 2.

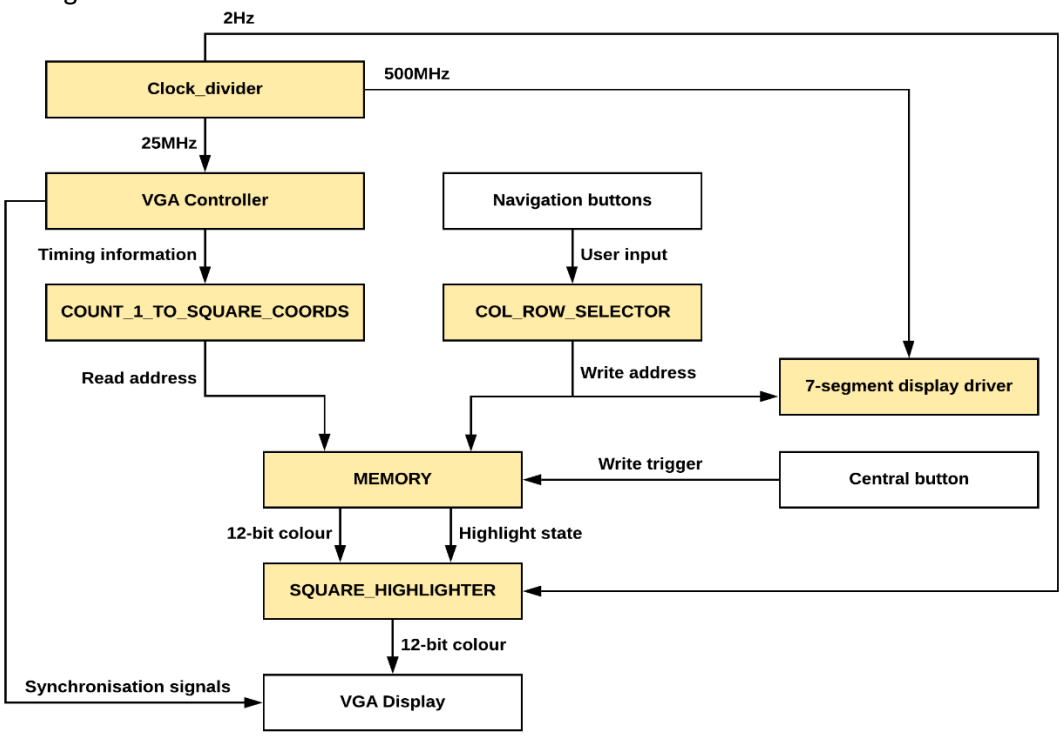


Figure 2. Diagram of top-level design. Hierarchical blocks are marked with orange colour.

Each of the 7 hierarchical blocks from Figure 2 is described in a separate section, with any child-blocks mentioned within their parent-block sections. Table 1 shows overview of the main responsibilities of the top-level hierarchical blocks.

Hierarchical block	Responsibility
VGA Controller	Provide essential signals for running VGA (sync. and blanking).
COUNT_T_TO_SQUARE_COORDS	Convert VGA counters state into square position (for reading).
COL_ROW_SELECTOR	Convert user input into square position (for selecting/writing).
MEMORY	Store colour data, allow reading and overwriting it.
7-segment display driver	Display currently selected square position.
SQUARE_HIGHLIGHTER	Invert colour of currently selected square position.
Clock_divider	Provide different clock frequencies to different components.

Table 1. Top-level hierarchical blocks overview.

2.2 VGA Controller (Figure 3)

It takes 25MHz clock as input and provides synchronization and blanking signals. It additionally outputs the state of both counters (1st used for horizontal timing and 2nd used for vertical timing), this allows to recognize which area of the screen is currently painted.

Internally it consists of 2 main blocks that are almost identical, these are called:
- Horizontal timing
- Vertical timing

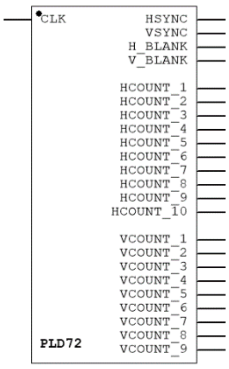


Figure 3. VGA Controller

Each of these blocks is responsible for providing its own synchronisation signal, blanking signal, and counters state output. Horizontal timing block outputs 10 bits of counter state, whereas Vertical timing block outputs 9 bits of counter state. That is because visible area horizontally consists of 640 pixels (10-bit number), and visible area vertically consists of 480 pixels (9-bit number).

The working principle of both these blocks is shown in Figure 4. Conditionals (orange shapes) are implemented using 10-bit comparators. Setting and resetting of "blank" and "sync" signals is done by J (set) and K (reset) inputs of JK-flip flops.

The differences between horizontal and vertical versions of the timing circuit include the following:

- Horizontal timing is supplied with 25MHz clock, Vertical timing clock is supplied with "Line_completed" output of the Horizontal timing circuit instead.

- both compare "count" against different values as shown in the Table 2.

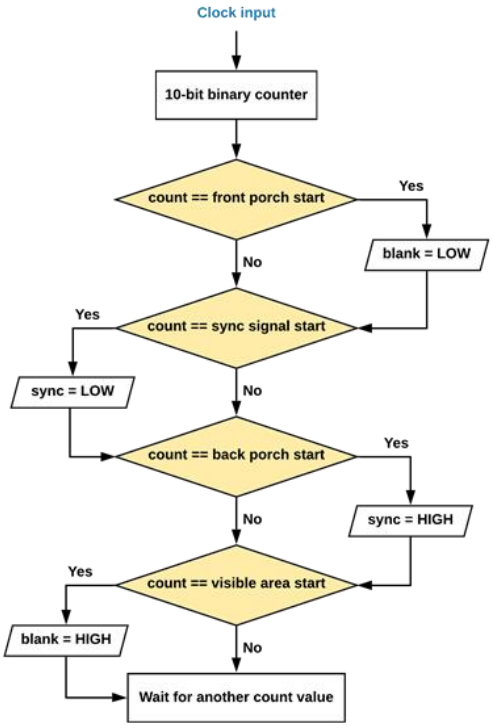


Figure 4. Horizontal timing and Vertical timing blocks working principle.

	Front porch start	Sync signal start	Back porch start	Visible area start
Horizontal timing	640	656	752	800
Vertical timing	480	490	492	525

Table 2. Comparison values against the counters state [6].

10-bit counters and 10-bit comparators were designed by cascading smaller components together. That was to improve the clarity of horizontal and vertical timing circuits. Figure 5 and Figure 6 show how these components were created.

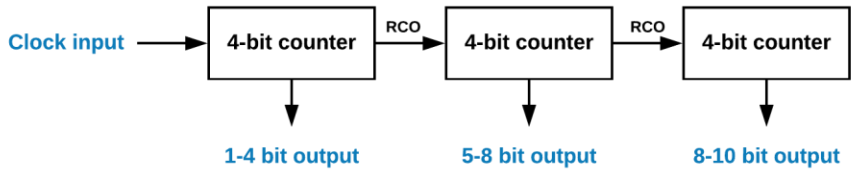


Figure 5. 10-bit counter simplified working principle.

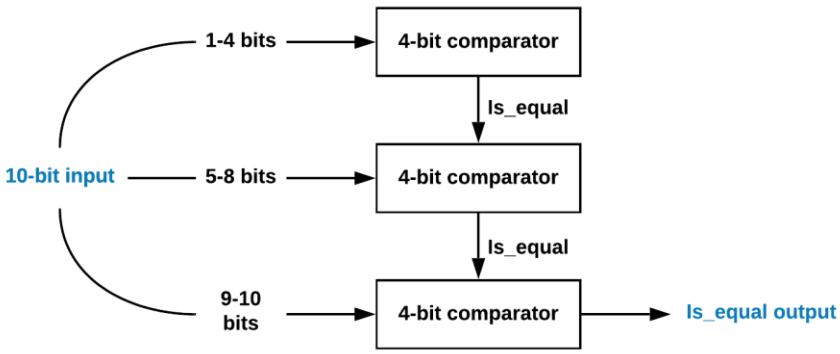


Figure 6. 10-bit comparator simplified working principle.

2.3 COUNT_1_TO_SQUARE_COORDS (Figure 7)

It outputs the row and column number (2-bits each) of the currently displayed square based on the output of VGA Controller counters and blanking signals.

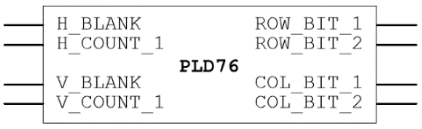


Figure 7. COUNT_1_TO_SQUARE_COORDS

Internally it contains 2 copies of the same element called "COUNT_1_TO_SQUARE_COORD" (Figure 8).

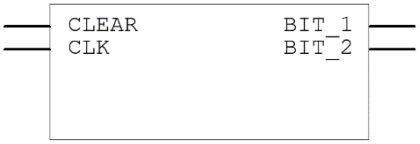


Figure 8. COUNT_1_TO_SQUARE_COORD

These elements are 2-bit counters incremented whenever tick count of CLK reaches a multiple of 80.

12 squares supposed to be displayed on the screen in a 3x4 pattern using 640x480 resolution. This means that each square will occupy 160x160 area of the screen. Therefore H_COUNT_1 and V_COUNT_1 will "tick" 80 times each for each square.

Figure 9 shows the behaviour of COUNT_1_TO_SQUARE_COORD element. The conditional checks for 80 and 81 counter state are done using AND gates. The 2-bit counter (implemented using 4-bit counter) has asynchronous clear. Figure 10 shows what components were used to implement internal (2 AND gates) and external (OR gate and inverter) clears.

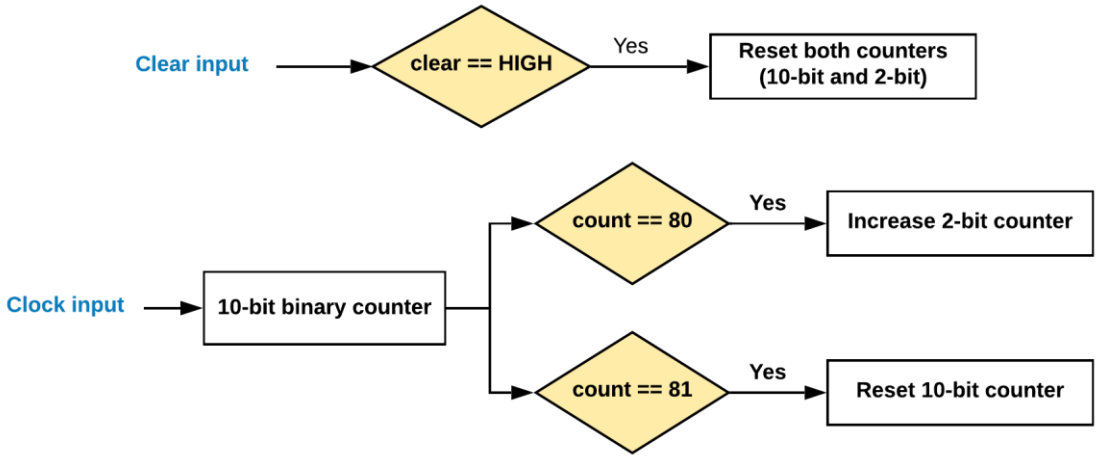


Figure 9. COUNT_1_TO_SQUARE_COORD behaviour.

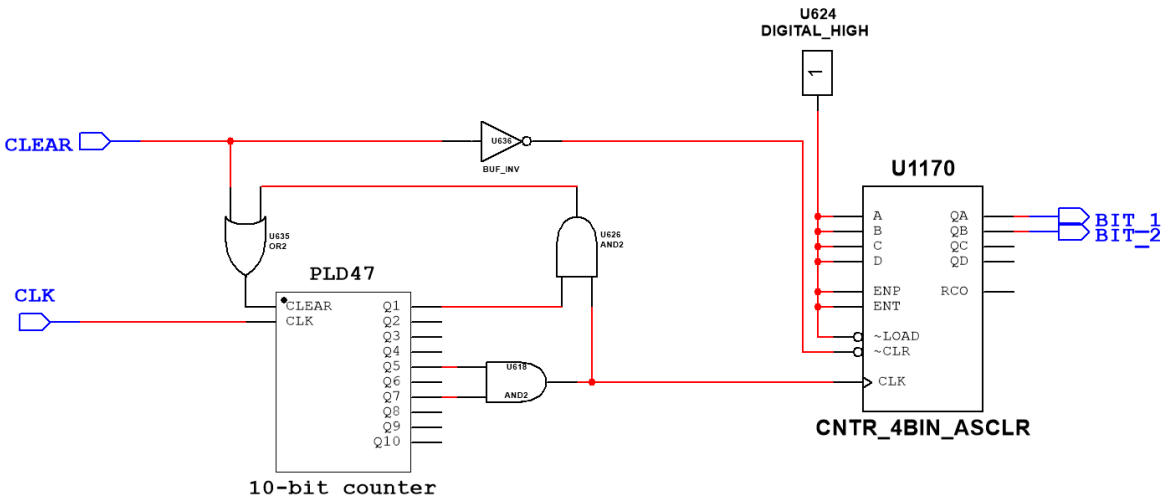


Figure 10. COUNT_1_TO_SQUARE_COORD internals.

2.4 COL_ROW_SELECTOR (Figure 11)

It outputs the row and column number of the square currently selected by the user, based on navigation buttons.

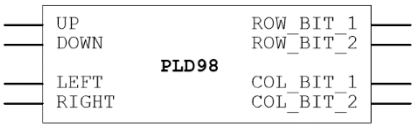


Figure 11. COL_ROW_SELECTOR

The behaviour of this component is shown in Figure 12. Increasing/decreasing is achieved by using counter with up/down input pin, to which the “DOWN” input is supplied.

The maximum number of rows is 3, (2, starting with 0). That is why NAND gate was used to detect that the row number is out of range. The output of the NAND gate goes to the ~LOAD pin of the counter.

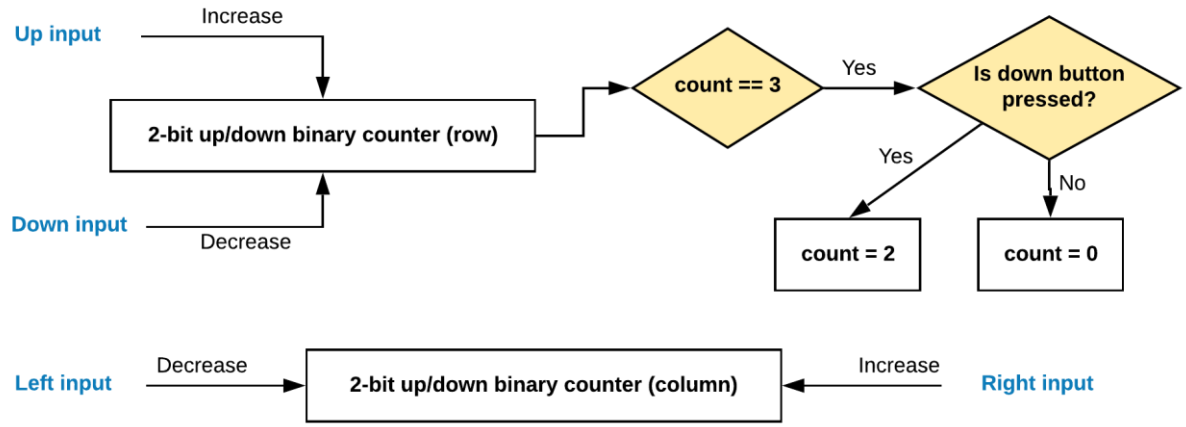
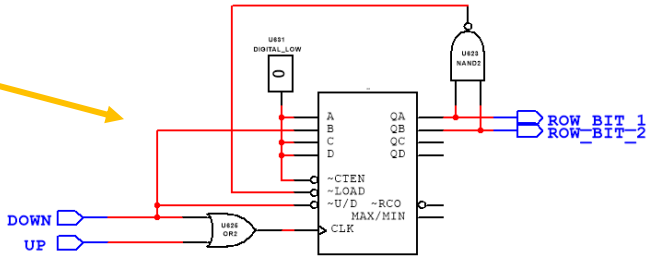


Figure 12. COL_ROW_SELECTOR behaviour.

The 2nd bit (B pin) of LOAD data input bits (A, B, C, D) of the counter is supplied with the state of “DOWN” input. This way the value becomes either 2 or 0 when the range is exceeded.



2.5 MEMORY (Figure 13)

If the “VGA Controller” block is considered the heart of the design, then MEMORY is the brain of it. It stores 12 x 12-bit colours data. It has separate address input for reading/writing and Allows reading/writing of 2 separate cells at the same time, where reading is asynchronous, and writing occurs upon central button press (using state of 12 switches as data input).

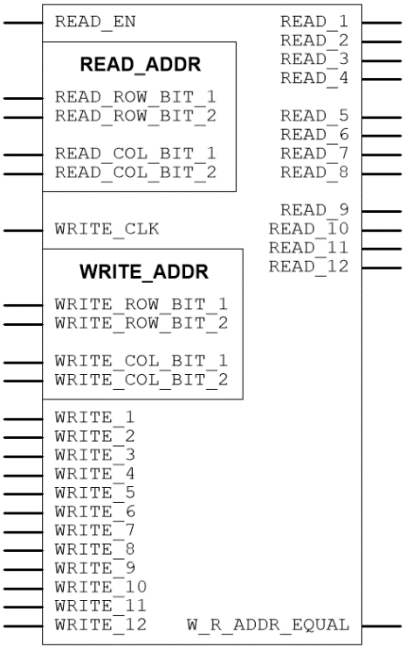


Figure 13. MEMORY

Internally it consists of 12 memory cells arranged in a 3x4 matrix and 4x 2 to 4 decoders responsible for selecting the cell in the right row/column for reading/writing. For that purpose each cell contains 4 “enable” pins, 2 for row and 2 for column. In Figure 14, each black line represents 2 separate “enable” signals, 1 for selection of the cell to read from, and 1 for selection of the cell to write to.

It also contains “READ_EN” which forces memory output (READ_1 to READ_10 pins) to be LOW regardless of selected row and column. This allows to act upon blanking signal state to prevent colour output during blanking time.

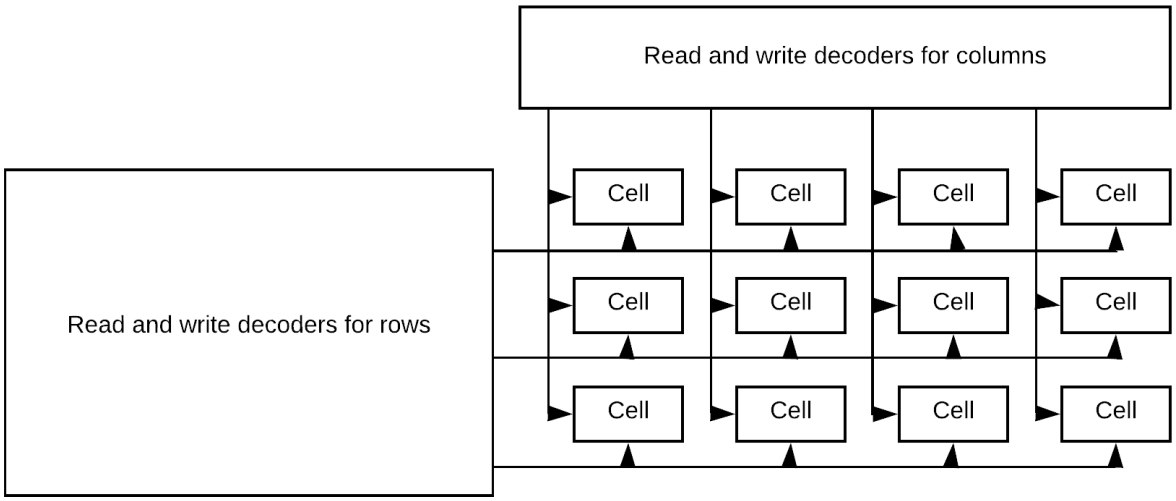


Figure 14. MEMORY addressing.

Figure 15 gives a closer look at how elements responsible for selecting column are wired.

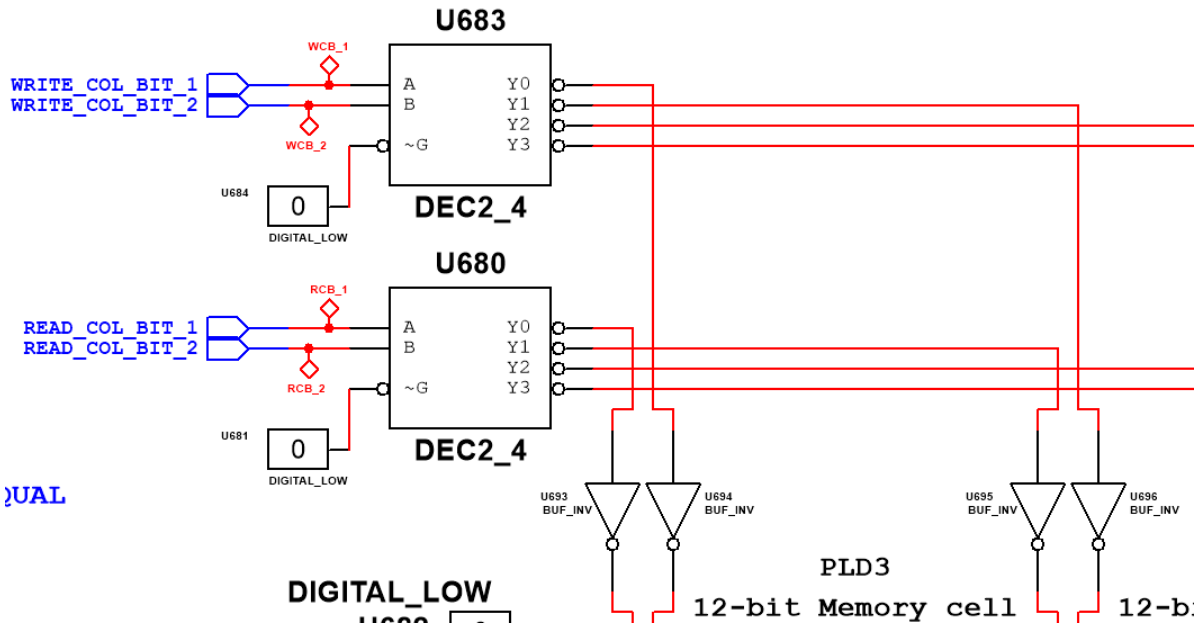


Figure 15. Decoders used for selecting appropriate column.

All the cells are connected to each other and pass their output through OR gates. The output of the last cell in the matrix is the output of the whole MEMORY block as shown in the Figure 16. Only 1 cell should be selected for reading at a time (the same applies to writing), that is why OR gates can be used without corrupting the output.

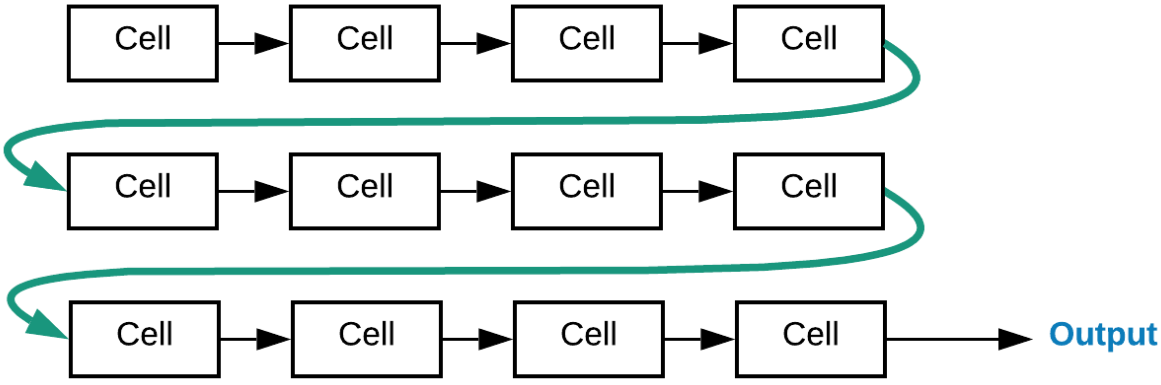


Figure 16. MEMORY - each cell is passing output through its internal OR gate. Only the last cell is directly connected to the output of the whole MEMORY block.

Each cell contains “WRITE_CLK” pin (Figure 17) that triggers write action if the appropriate row and column is selected.

The “PREV_” inputs allow chaining multiple cells through OR gates, so there is no need to use multiplexers to choose output of a particular cell.

Inside 12-bit Memory cell there are 12 D-flip flops, each storing 1 bit of data.

AND gate is used as a clock for the flip-flops, which makes them overwrite their value using WRITE_1 to WRITE_12 inputs. The 3 inputs for the AND gates are:
- WRITE_CLK
- WRITE_EN_ROW
- WRITE_EN_COL

Equivalent AND gate is used to enable the output of 12 flip-flops, containing the following 3 inputs:
- READ_EN
- READ_EN_ROW
- READ_EN_COL

Figure 18 shows how exactly it looks like.

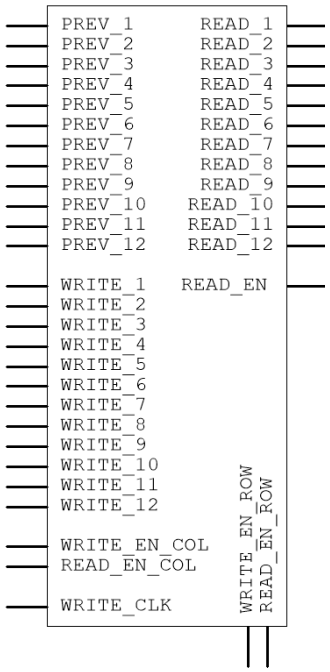


Figure 17. 12-bit Memory cell

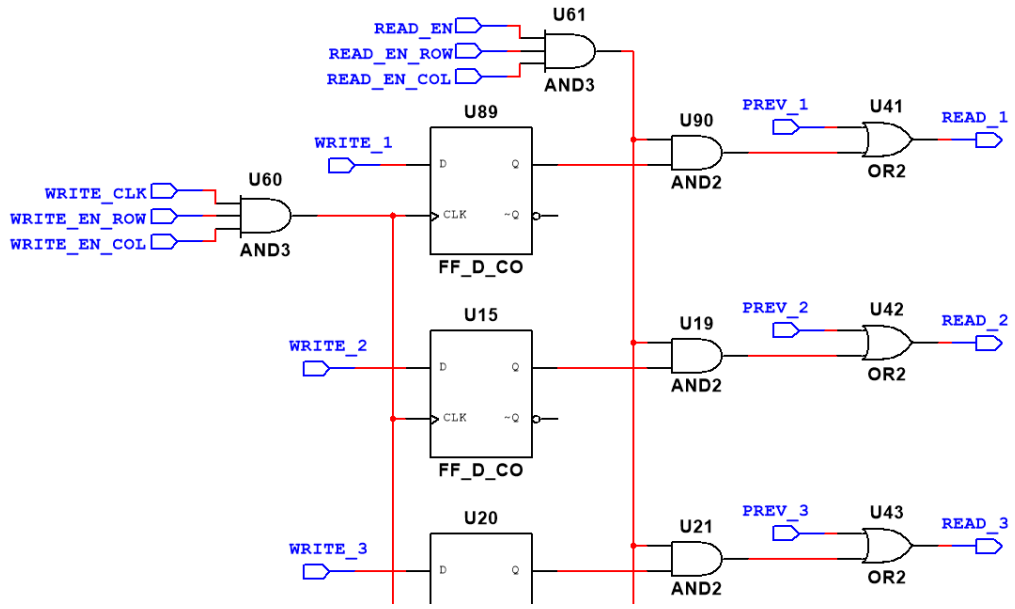


Figure 18. 12-bit Memory cell internals. The remaining 9 flip-flops are not shown on the image above but they are wired in the same way.

2.6 7 – segment display driver (Figure 19)

It displays row and column that is currently selected for writing.

Internally it consists of:

- 2-bit counter driven by 500Hz clock
- 4-set 4-bit multiplexer driven by the 2-bit counter
- Anode controller driven by the 2-bit counter (which sets the currently selected anode to LOW)
- BCD to 7-segment display driver
- “7-segment row col override” block

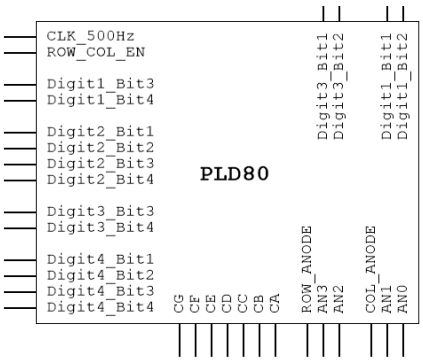


Figure 19. 7-segment display driver

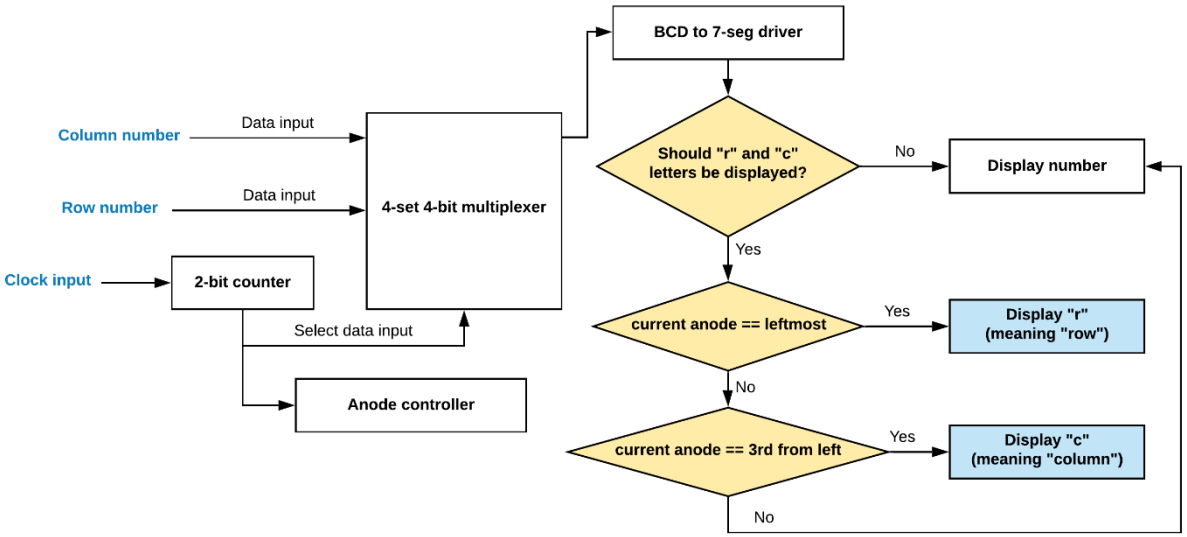


Figure 20. 7-segment display behaviour. Conditionals (orange colour) and blue colored rectangles represent behaviour of the "7-seg row col override" block.

2.6.1 2-bit counter

The 2-bit counter used in the 7-segment display driver design consists of 2 D-flip-flops and a XOR gate [7].

2.6.2 4-set 4-bit multiplexer

It is necessary to use multiplexer to display different values on different areas of the 7-segment display. That is because of the way the display is wired. Each of the 8 cathodes of the segments is shared among all 4 digits. It allows to decrease the number of pins required to drive the full display, on the other side it allows to illuminate only 1 digit at a time (by setting state of its corresponding anode). Multiplexer is used to supply different values during the time period dedicated for particular digit of the display.

2.6.3 Anode controller

Anode controller is using combinational logic (3x OR with a single NAND gate as seen in Figure 21) and acts as 2-line to 4-line decoder which sets appropriate anode to LOW depending on the supplied 2-bit number. It could be noticed that both, cathodes and anodes are set to LOW in order to illuminate a segment, that is because the Basys 3 board "uses transistors to drive current into the common anode point" as described in Basys 3 reference manual [8].

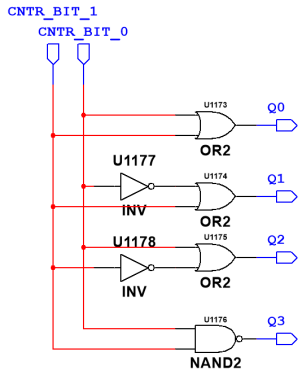


Figure 21. Anode controller internals

2.6.4 7-seg row col override

It is responsible for displaying "r" and "c" letters representing "row" and "column". It prevents digits from being displayed where the letters are displayed instead (Figure 23).

The block contains "enable" pin (EN) which allows to maintain original functionality of the 7-segment display driver (if displaying of letters is not needed, and "enable" is set to LOW). It makes the element reusable for other projects that require displaying all 4 digits.

The ROW_ANODE pin is wired to the leftmost anode (where "r" should be displayed).

The COL_ANODE pin is wired to the 3rd anode from left (where "c" should be displayed).

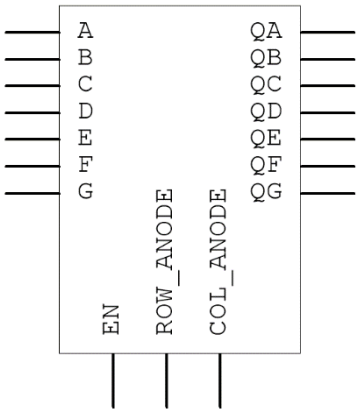


Figure 22. "7-seg row col override" block.

Receiving the state of these signals allows this block to apply changes (display letter instead of digit) only when it is the right time to do so.

Figure 24 shows combinational logic used to display letters instead of digits. It is worth to notice that it is dealing with negative logic since segments are illuminated with cathodes are set to LOW, which may be harder to grasp than dealing with positive logic [9]. Aside from displaying the letters, additional objective of the circuit shown in Figure 24 is to leave the signals in their original state if either:

- EN is LOW

- ROW_ANODE and COL_ANODE are HIGH (inactive)

Figure 23. "r" and "c" letters being displayed. Currently selected row is 2 (3rd), currently selected column is 3 (4th). The values respond to user input using navigation buttons.

The purpose of 7 OR gates array is to force signals to be HIGH if EN is HIGH (which "turns off" the segments).

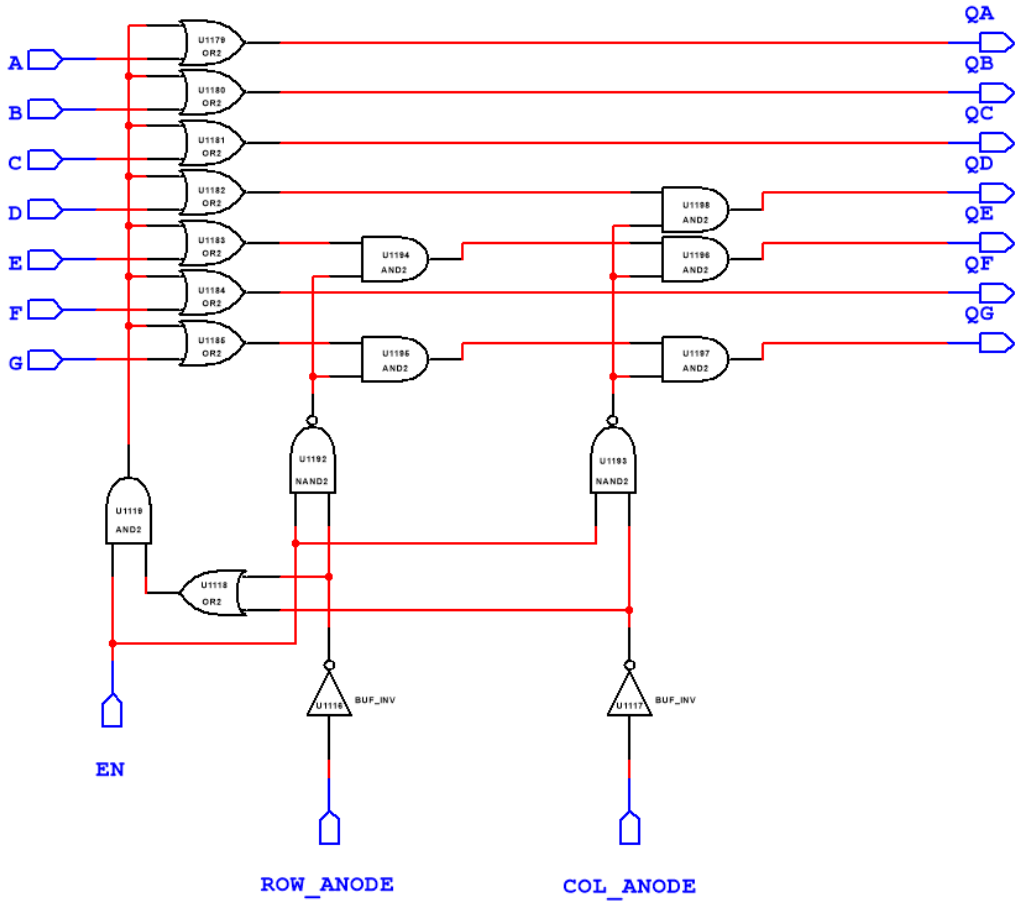


Figure 24. Combinational logic used to display letters instead of digits.

2.7 SQUARE_HIGHLIGHTER (Figure 25)

It inverts each bit of colour data (if it's not blanking time and if the reading address of memory equals writing address). It is made of 12x XOR gates and a single AND gate. This can be seen on Figure 26.

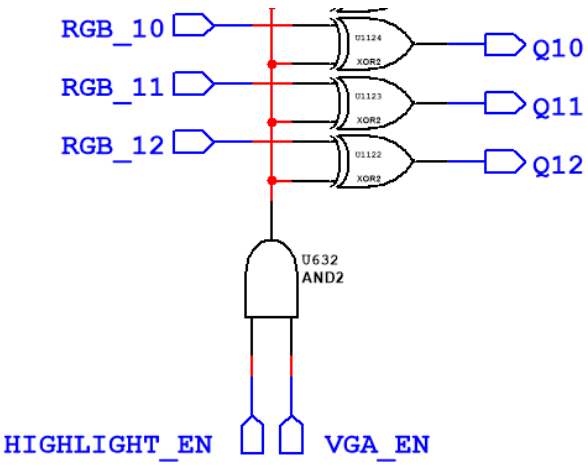


Figure 26. Internals of SQUARE_HIGHLIGHTER. The remaining 9 XOR gates are not shown.

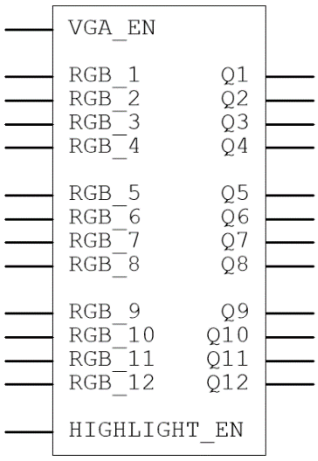


Figure 25. SQUARE_HIGHLIGHTER

Two separate enable pins are used for convenient wiring of:

- the clock/current-square-indicator (HIGLIGHT_EN) to flash the area of the screen occupied by the currently selected square, and to do it at given frequency

- blanking signal (VGA_EN) to avoid inverting RGB signals during blanking time

Figure 27 shows the behaviour of the SQUARE_HIGHLIGHTER block.

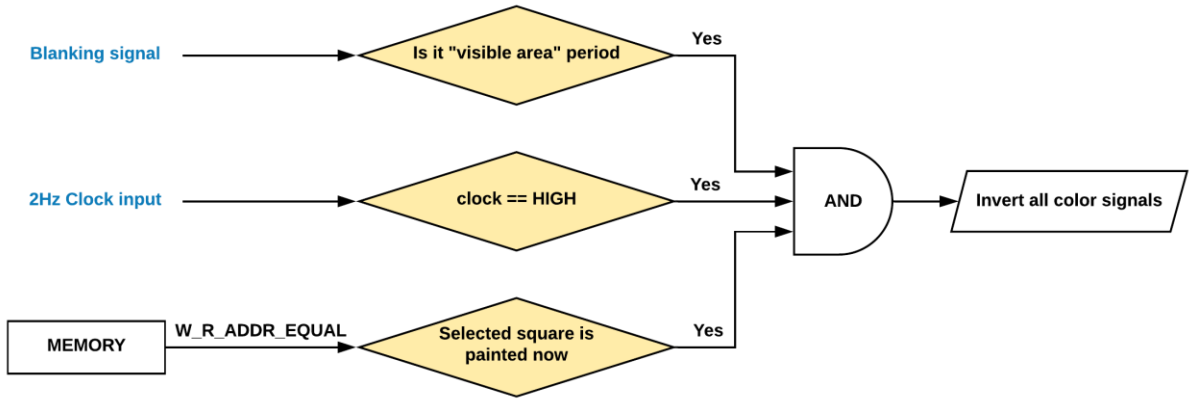


Figure 27. Behaviour of the SQUARE_HIGHLIGHTER. In the design itself, 2x 2-input AND gates (1 internal and 1 external) are used instead of a single 3-input AND gate, flowchart above aims to show behaviour of the highlighter, not its structure.

2.8 Clock_divider (Figure 28)

Takes 100MHz clock as input and provides 25MHz pixel clock for the VGA Controller (CLOCK_A), 500Hz clock for the 7-segment display driver (CLOCK_B), and 2Hz clock for flashing the current square (CLOCK_C). It was created in accordance with the "Laboratory Stage 1" document.

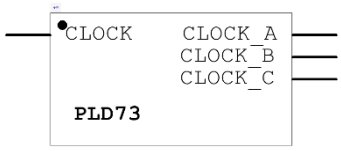


Figure 28. Clock_divider

Internally, it consists of a single 4-bit binary counter and 8 binary coded decimal counters.

The binary counter is used to divide clock frequency by 4.

Binary coded decimal counters are used to divide the clock frequency by 200 thousand (e.g. 100MHz turning into 500hz) and 50 million (e.g. 100MHz turning into 2Hz).

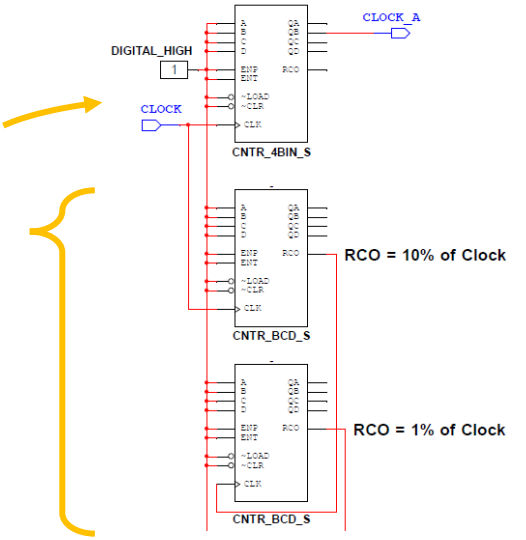


Figure 29. Clock_divider internals. (Only 2 of the BCD counters are shown).

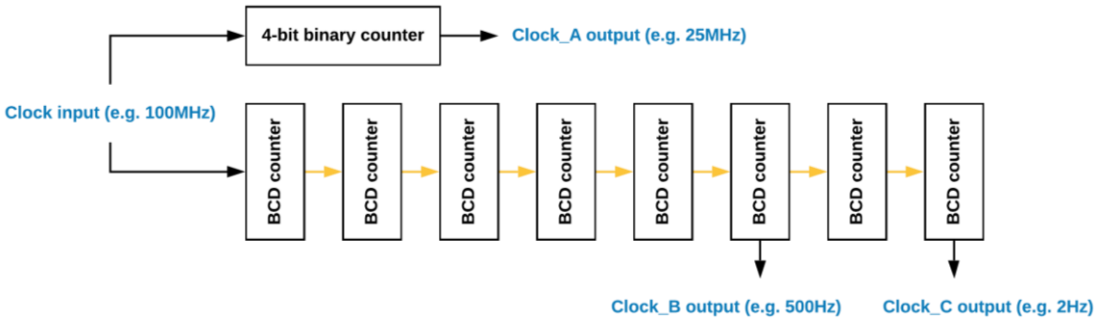
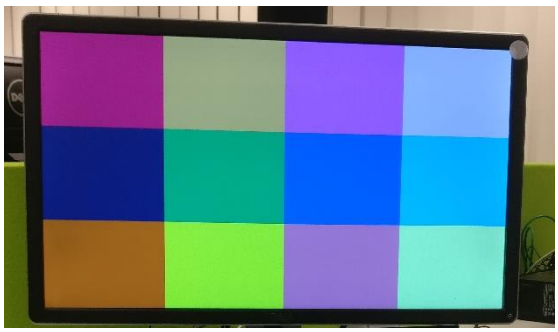


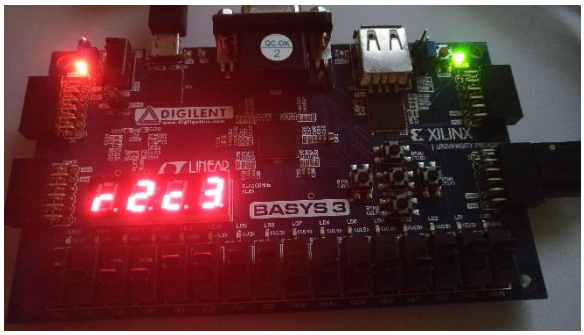
Figure 30. Clock_divider structure. Orange arrows indicate ripple carry output (RCO) being supplied as clock input.

3. Performance and tests

3.1 Whole design

All the objectives listed in the "Design requirements" section were met. The screen is split into 3x4 regions. The navigation buttons allow selecting region. The currently selected region flashes on the screen, additionally the row and column number of the selected square is displayed neatly on the 7-segment display. The central button overwrites the colour of the selected square using the state of 12 switches.





The behaviour of the design can be viewed online at:
https://www.youtube.com/watch?v=4RAvjZRb09s

3.2 Separate components

3.2.1 VGA controller

Seeing squares being displayed on the screen is probably sufficient indicator of "VGA controller" working properly, however multiple measurements were made to verify the timings during the development process.

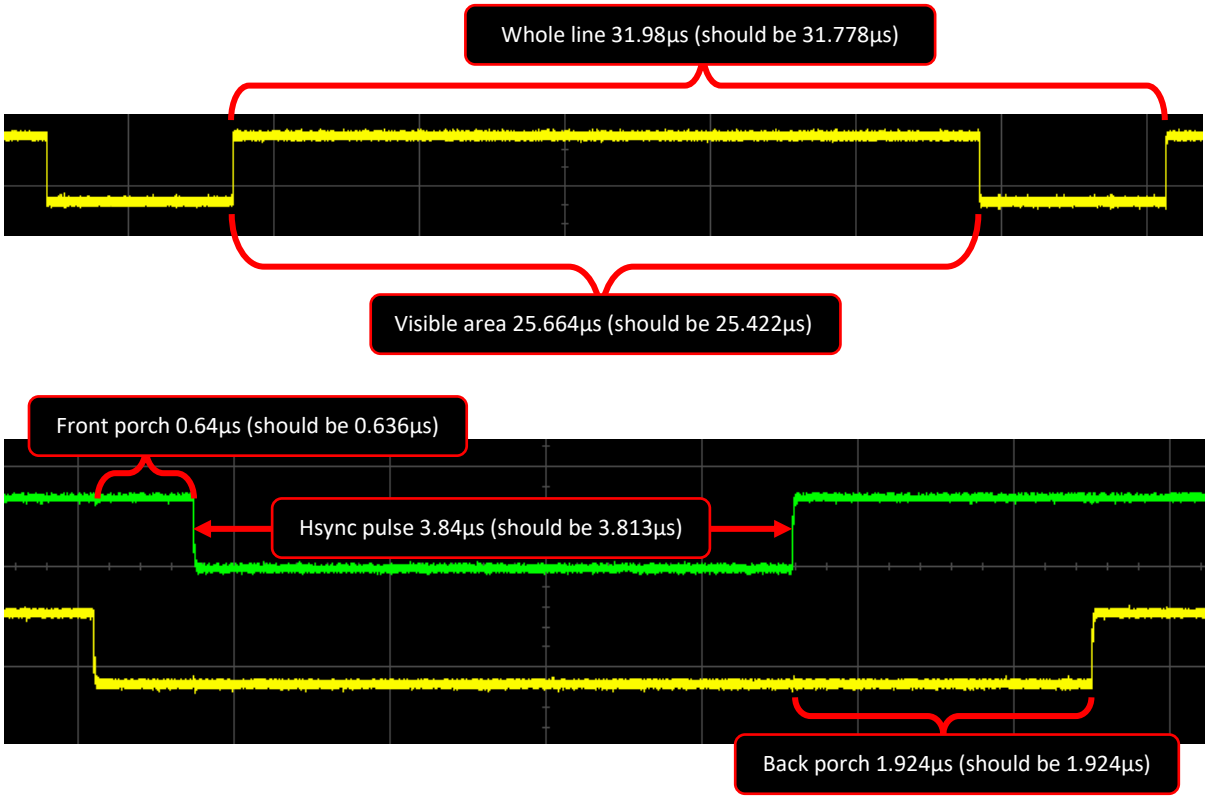
In order to test "Horizontal timing" and "Vertical timing" blocks, their "HSYNC/VSYNC" and "H_BLANK/V_BLANK" signals were connected to JC headers and measured with oscilloscope using "cursor measurement" feature [10].

The collected timings were compared against desired timings when driving VGA display using 640x480 resolution, 25MHz pixel clock and 60Hz refresh rate. The desired timings are presented in Table 3.

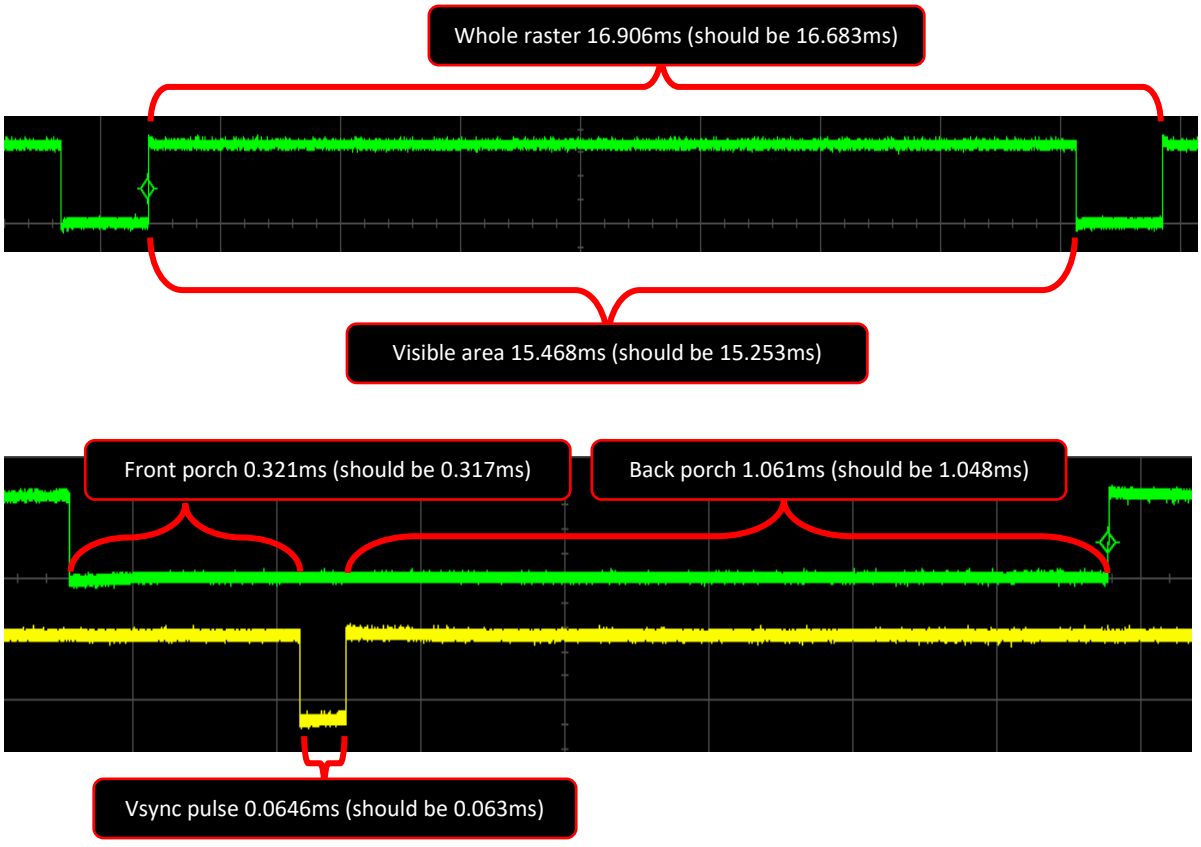
	Visible area	Front porch	Sync pulse	Back porch	Whole line
Horizontal (µs)	25.422	0.636	3.813	1.907	31.778
Vertical (ms)	15.253	0.317	0.063	1.048	16.683

Table 3. Horizontal and vertical synchronisation timings (VGA 640x480, 25MHz pixel clock, 60Hz refresh rate). Adapted from tinyvga.com [6].

The results of “Horizontal timing” block measurements are presented below.



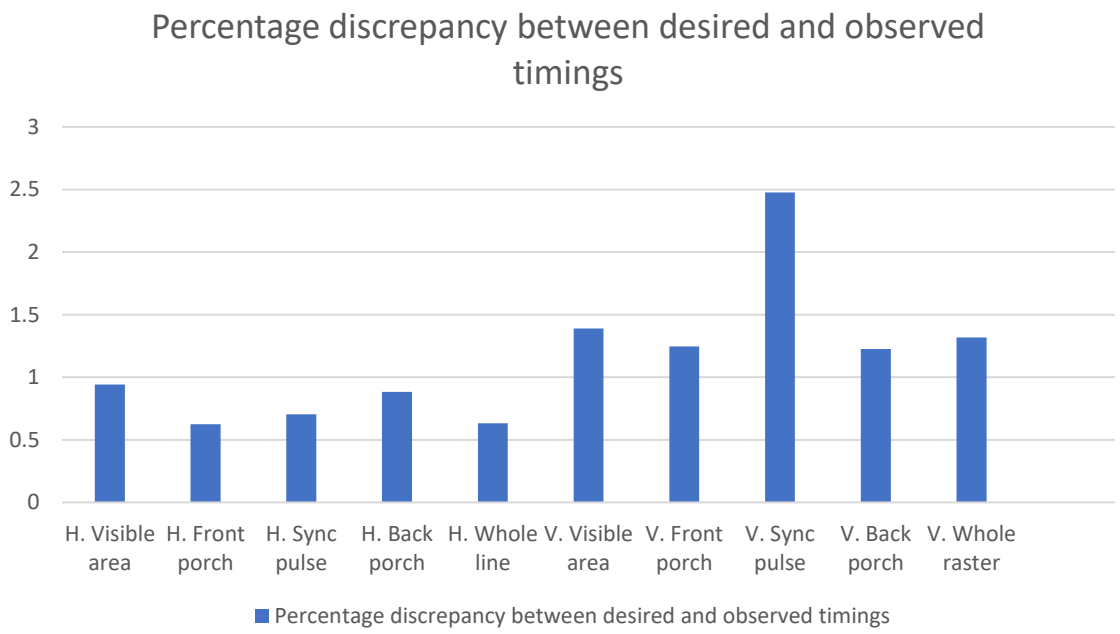
The results of “Vertical timing” block measurements are presented below.



The collected frequency readings were slower (took more time) than the ones listed in Table 3. That is because of 2 reasons:

- Mainly it's due to the 25MHz pixel clock being used instead of the suggested 25.175MHz clock.

- Horizontal timing counter being reset at 801 instead of 800. Vertical timing counter being reset at 526 instead of 525.



Mean horizontal discrepancy = 0.76% (of observed value)
Mean vertical discrepancy = 1.53%

Pixel clock suggested by tinyvga = 25.175 MHz
Pixel clock used = 25 MHz
Discrepancy between suggested and used pixel clock frequencies = 0.70% (of suggested value)

Discrepancy caused by using 801 counter state check (instead of 800) in "Horizonal timing" circuit = 0.13%
Discrepancy caused by using 526 counter state check (instead of 525) in "Vertical timing" circuit = 0.19%

The following formula was used:

$$discrepancy = \frac{(observed\_value - desired\_value)}{observed\_value} * 100$$

3.2.2 7-segment display driver

Aside from observation of the display itself, the 7-segment display driver internal components were tested by wiring them to JC pins and measuring their state using logic analyser of which results are presented in Figure 31. Each distinct state of the 2-bit counter results in different anode being activated (by setting it LOW) which is achieved by “Anode controller” block (decoder).

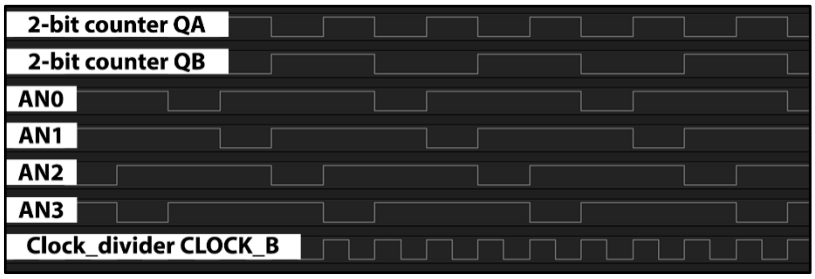


Figure 31. Relation between input clock (CLOCK_B), 2-bit counter state and the anode signals.

Figure 32 shows the results of another test which verified the state of 4-set 4-bit multiplexer output. During the measurement all the switches were “turned off” except 2 switches that are part of 4 switches responsible for the value of a number displayed through the first anode (AN0 from Figure 31). On the Figure 32 each high pulse of multiplexer output takes ¼ of the period required to display all 4 digits on the display, and occurs precisely when 2-bit counter outputs “0”, just as it should.

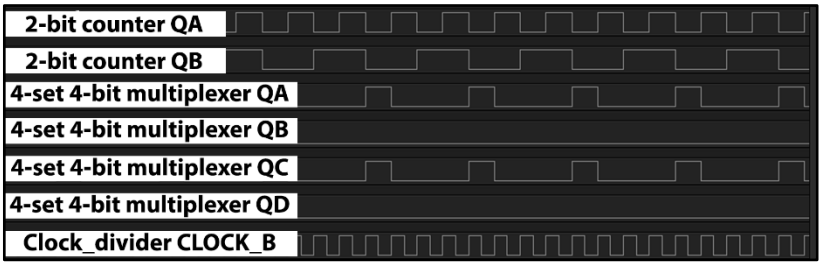


Figure 32. Relation between input clock (CLOCK_B), 2-bit counter and the 4-set 4-bit multiplexer (using switches as data source).

3.2.3 Clock_divider

It supposed to provide 3 clocks (square wave signals with different frequencies, CLOCK_A - 25MHz, CLOCK_B - 500Hz, CLOCK_C - 2Hz).

CLOCK_A and CLOCK_B were measured by connecting their output to JC pins of the Basys 3 board and checking their state using oscilloscope of which readings are presented in Figure 34. CLOCK_C was measured by connecting it to LED_0 and observing the flashing (Figure 33).

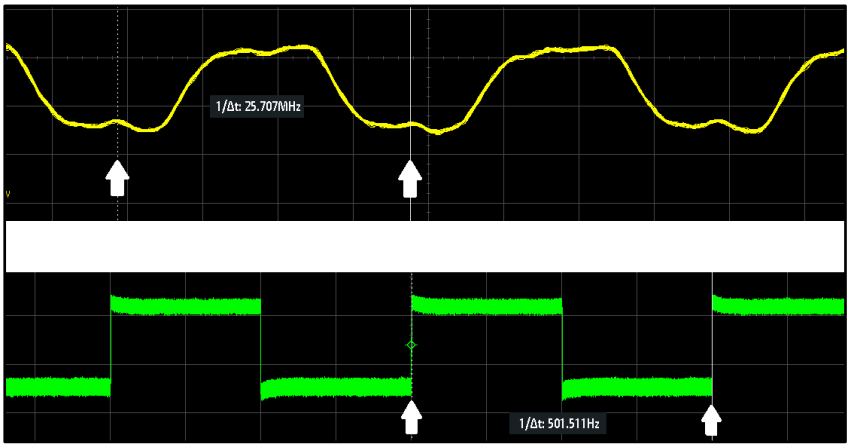


Figure 34. Clock_divider measurements.
***CLOCK_A** output - 25MHz (measured: 25.707MHz)*
***CLOCK_B** output – 500Hz (measured: 501.511Hz)*

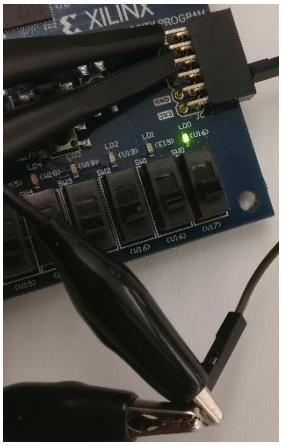


Figure 33. LED_0 flashing (using CLOCK_C). The JC pins used for CLOCK_A and CLOCK_B tests can be seen near the top of the image.

4. Areas for improvement

4.1 Consistent and more accurate naming

Hierarchical blocks, as well as their I/O pins contain all types of naming conventions (e.g. “Clock_divider”, “VGA controller”, “SQUARE_HIGHLIGHTER”).

Square highlighter does not always highlight the square (if the “highlighted” square has a bright colour by default then “highlighter” makes it darker), “square flasher/ color inverter” would be more accurate name.

In some parts of design the position of the square is referred to as “row” and “column” (or “col”), in other parts it is referred to as “COORDS”.

4.2 Better use of hierarchy

For example, the Clock_divider could contain hierarchical blocks such as “Clock_divider_10”, instead of large built-in Multisim components. This way it would be much cleaner. Making more variations of such dividers would allow to simplify COUNT_1_TO_SQUARE_COORD block by removing 10-bit counter, removing its surrounding combinational logic and replacing these with appropriate clock dividers.

4.3 Reusability

Avoid using I/O pin names that prevent or limit reuse in different design (e.g. CLK_500Hz input pin name in 7-segment display driver which could be supplied with other frequencies of the clock and work well too).

4.4 Avoid over-engineering things

"Anode controller" could be replaced with DEC2_4 component that is built-in Multisim.

The 4-bit up/down counter is not necessary in the COL_ROW_SELECTOR, it could be replaced by 2-bit up/down counter.

Instead of chaining the memory cells (using "PREV_" inputs and OR gates), the memory block could use tri-state buffers and have outputs of memory cells connected to each other using single bus.

5. Reflective statement

Working on this design allowed me to get more familiar with logic components and learn how to use Multisim/Vivado software (including their simulation features).

It also helped me to learn about ways to solve technical problems, I understood the value of using scientific method while debugging "black box" systems. Systems of which the working principle is unknown, and only the input/output can be observed. I learned that often in such case it might be optimal to do only 1 change of input at a time. I also learned that it is useful to note what tests I did while debugging (including their purpose and results), otherwise the results or purpose of the tests may be forgotten (especially if the problem persists and the number of tests becomes large).

If I was going to create this design again I would apply all the changes listed in "Areas for improvement" section. Additionally, I would be more distrustful about the reliability of single components. I would test the design more frequently by uploading it to the board, especially when adding new components. Even if some component should work properly in theory (or even if it worked well in Multisim and Vivado simulations) I would test it on the board because of the hard to debug issues arising from usage of specific component in the past (7-bit counter). Spending more time on testing the design could save me much longer debugging time, and as a result allow me to develop more advanced features.

6. References

[1] S. Sangwine, “Laboratory Stage 1 – Guided design of clock divider and 7-segment display driver,” 1 10 2019. [Online]. Available: https://moodle.essex.ac.uk/pluginfile.php/471733/mod_resource/content/10/Laboratory%20stage%201.pdf. [Accessed 19 12 2019].

[2] S. Sangwine, “Laboratory Stage 2,” 1 10 2019. [Online]. Available: https://moodle.essex.ac.uk/pluginfile.php/1106516/mod_resource/content/6/Laboratory%20stage%202.pdf. [Accessed 19 12 2019].

[3] S. Sangwine, “Laboratory Design Work – Stage 3,” 31 10 2019. [Online]. Available: https://moodle.essex.ac.uk/pluginfile.php/1181183/mod_folder/content/0/Laboratory%20stage%203.pdf?forcedownload=1. [Accessed 19 12 2019].

[4] digilentinc.com, “Basys 3 Artix-7 FPGA Trainer Board: Recommended for Introductory Users,” 2019. [Online]. Available: https://store.digilentinc.com/basys-3-artix-7-fpga-trainer-board-recommended-for-introductory-users/. [Accessed 19 12 2019].

[5] S. Larson, “VGA Controller (VHDL),” 7 3 2018. [Online]. Available: https://www.digikey.com/eewiki/pages/viewpage.action?pageId=15925278. [Accessed 19 12 2019].

[6] http://tinyvga.com, “VGA Signal 640 x 480 @ 60 Hz Industry standard timing,” SECONS Ltd., 2008. [Online]. Available: http://tinyvga.com/vga-timing/640x480@60Hz. [Accessed 20 12 2019].

[7] T. Floyd, Digital Fundamentals 11th edition (p. 508, Figure 9-12), Pearson, 2015.

[8] Digilent, “Basys 3 reference manual,” 2019. [Online]. Available: https://reference.digilentinc.com/basys3/refmanual. [Accessed 20 12 2019].

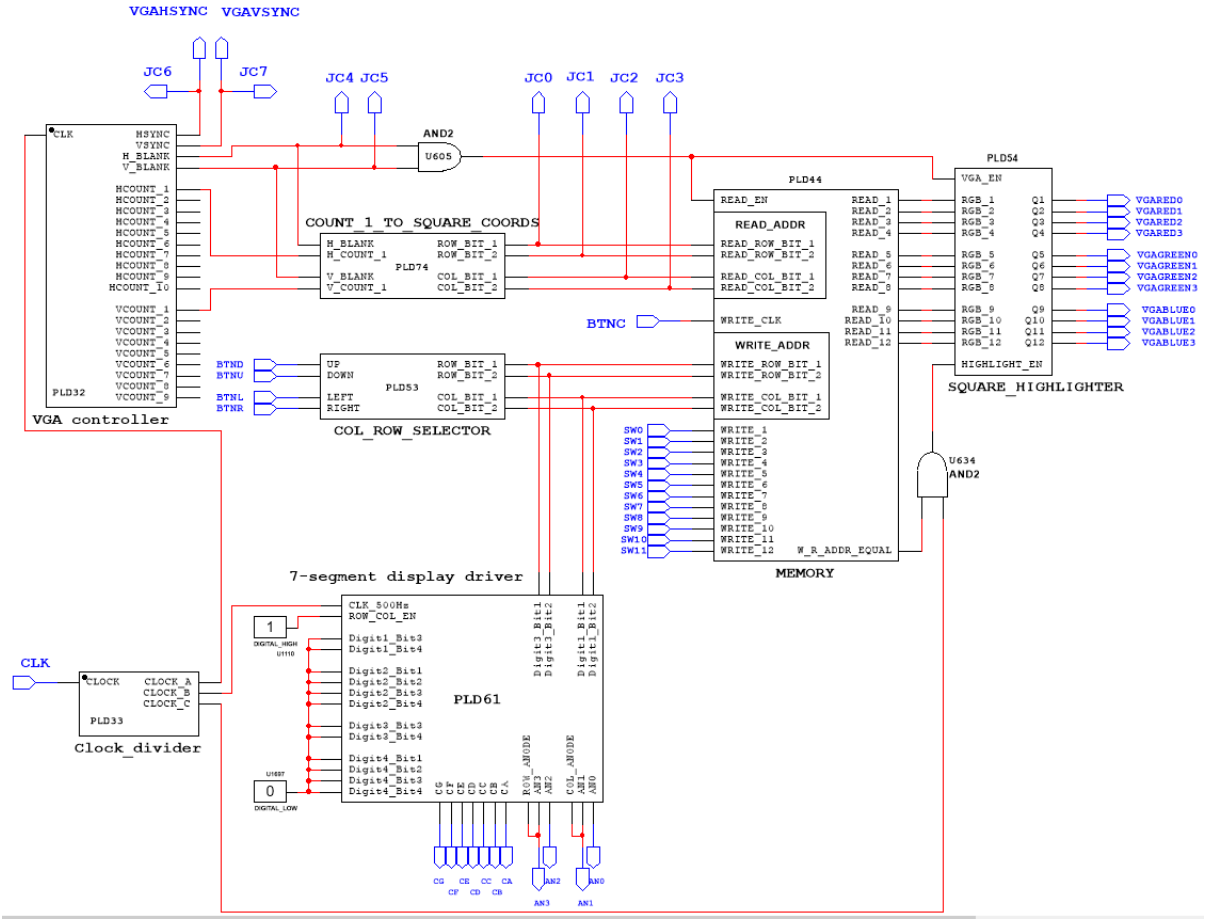
[9] R. Martin, “Clean Code - A Handbook of Agile Software Craftsmanship,” 2008, p. 302.

[10] R. &. Schwarz, “RTB2000 Digital Oscilloscope - User Manual (Cursor Measurements section),” 2017. [Online]. Available: https://www.batronix.com/pdf/Rohde-Schwarz/RTB2000/RTB_UserManual_en.pdf. [Accessed 22 12 2019].

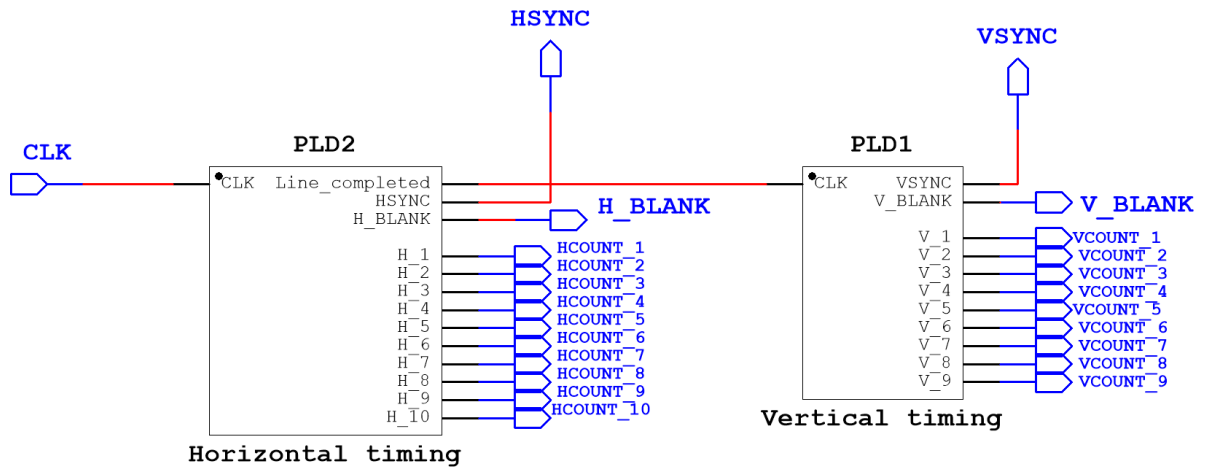
7. Appendices

7.1 Schematics

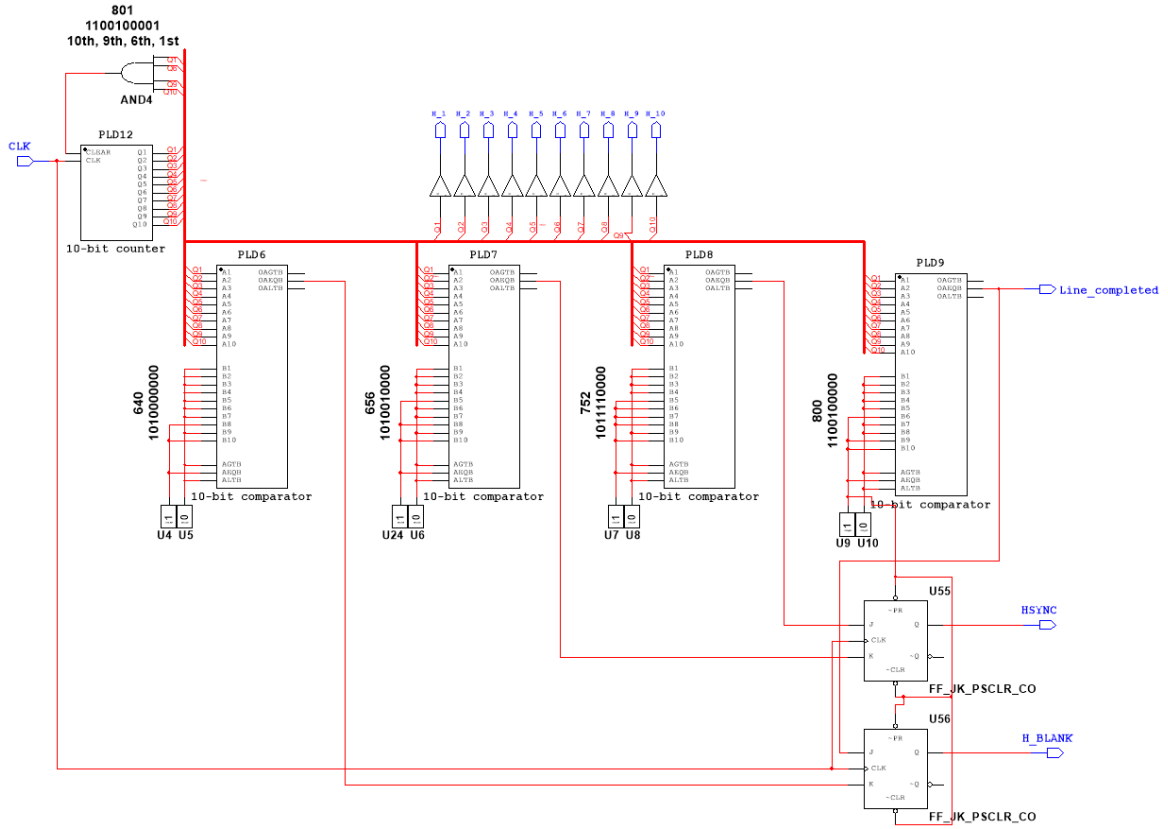
Top level



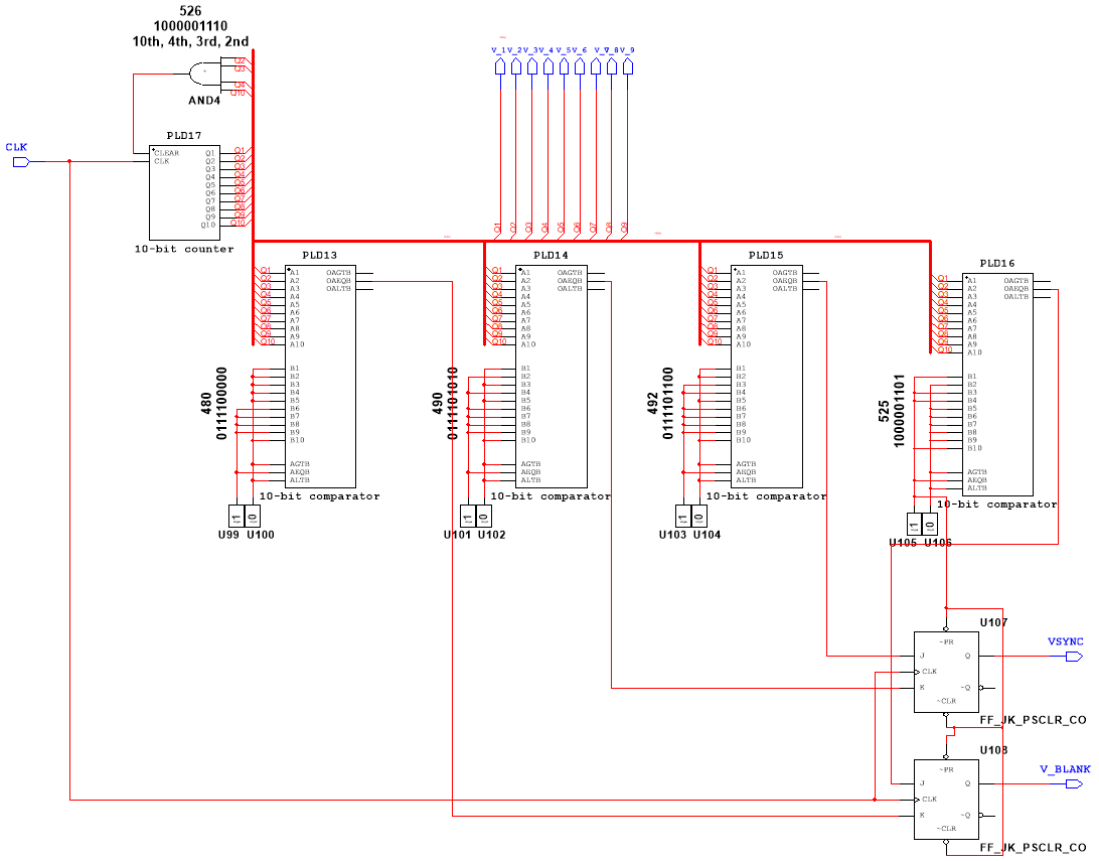
VGA Controller



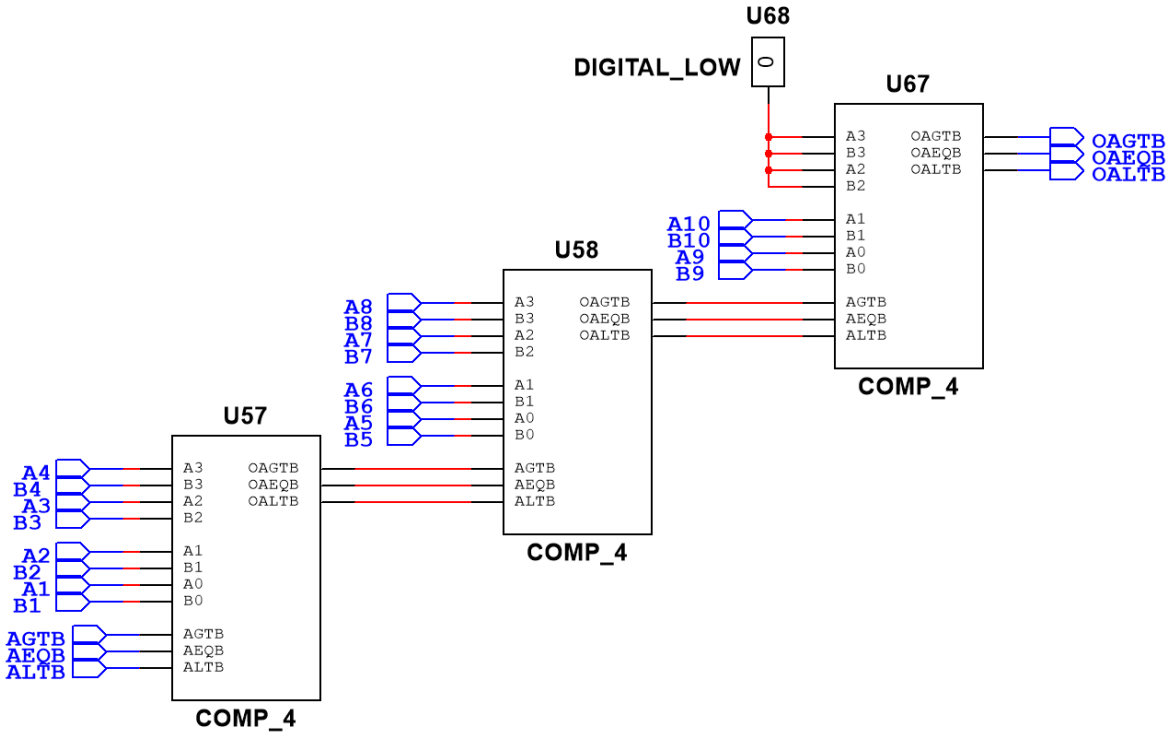
Horizontal timing



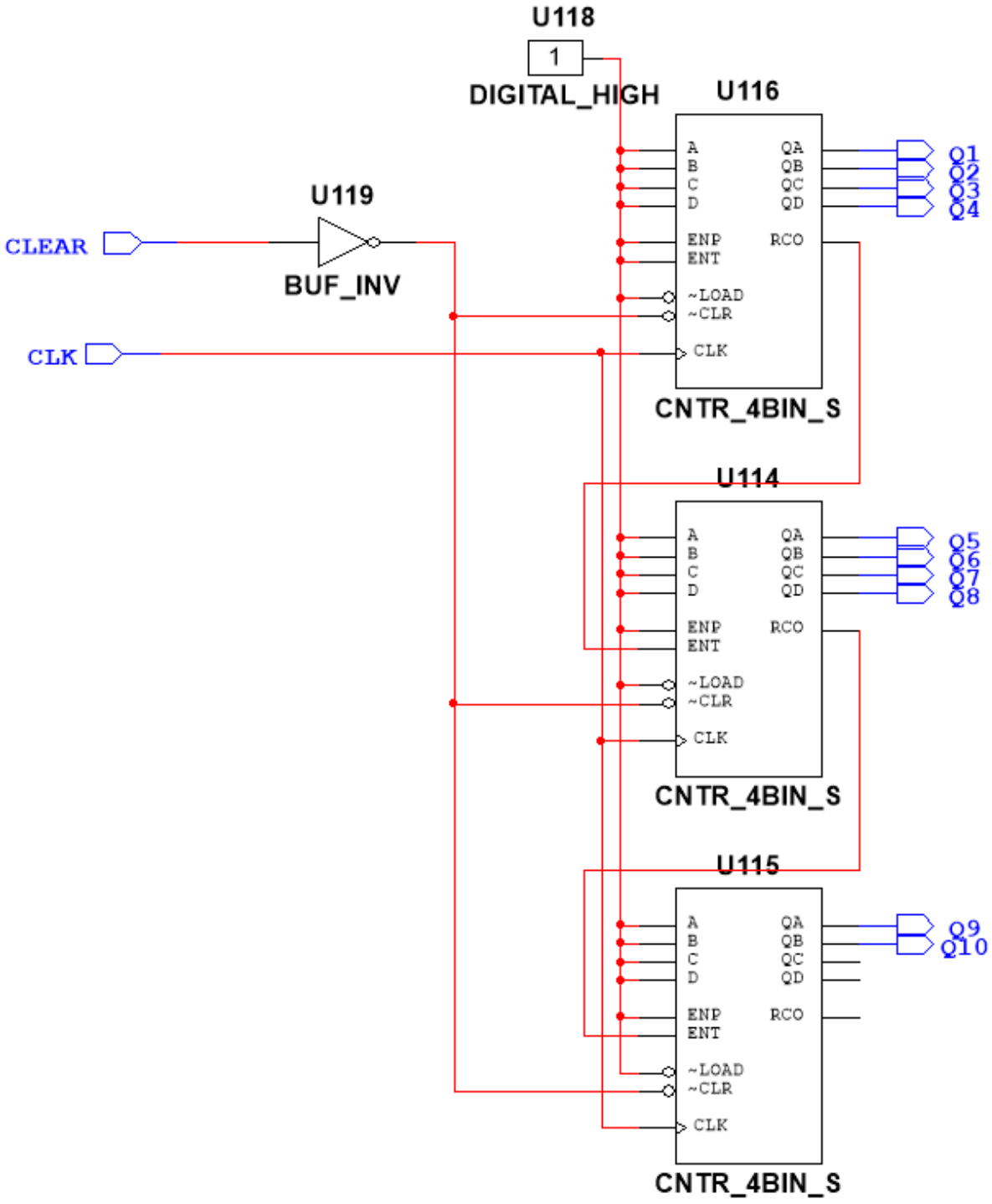
Vertical timing



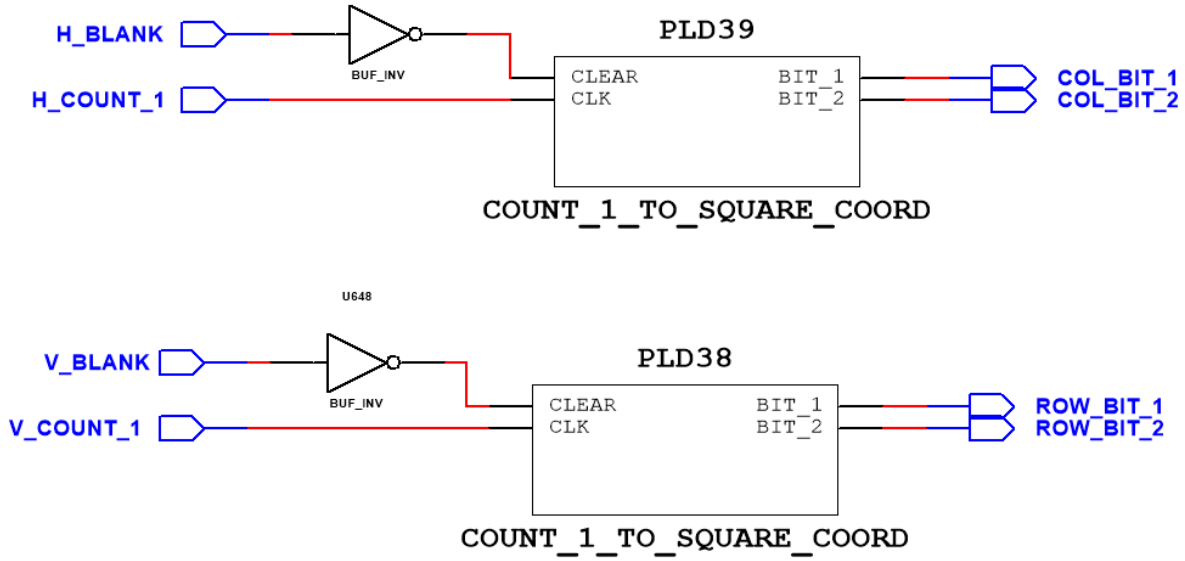
10 – bit comparator



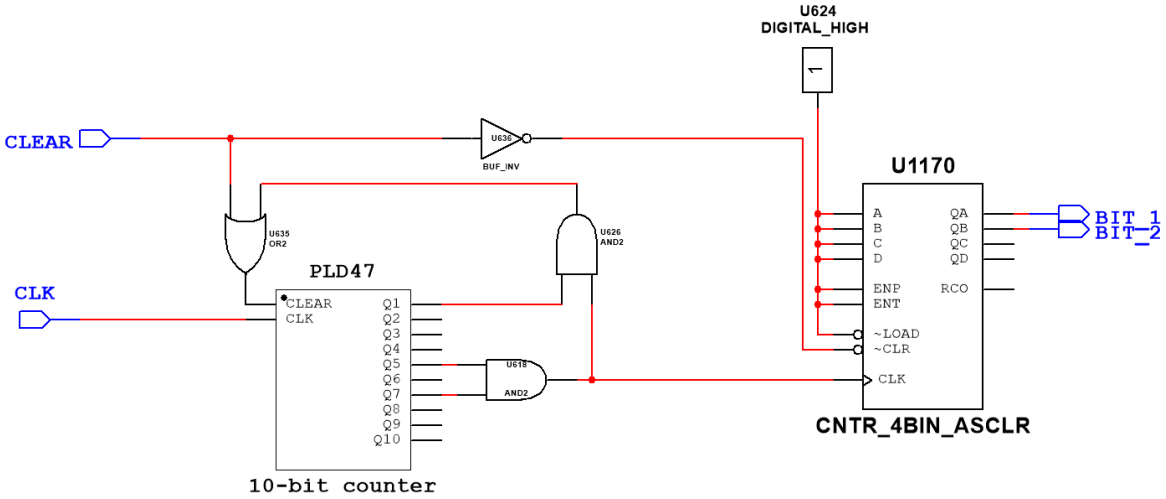
10-bit counter



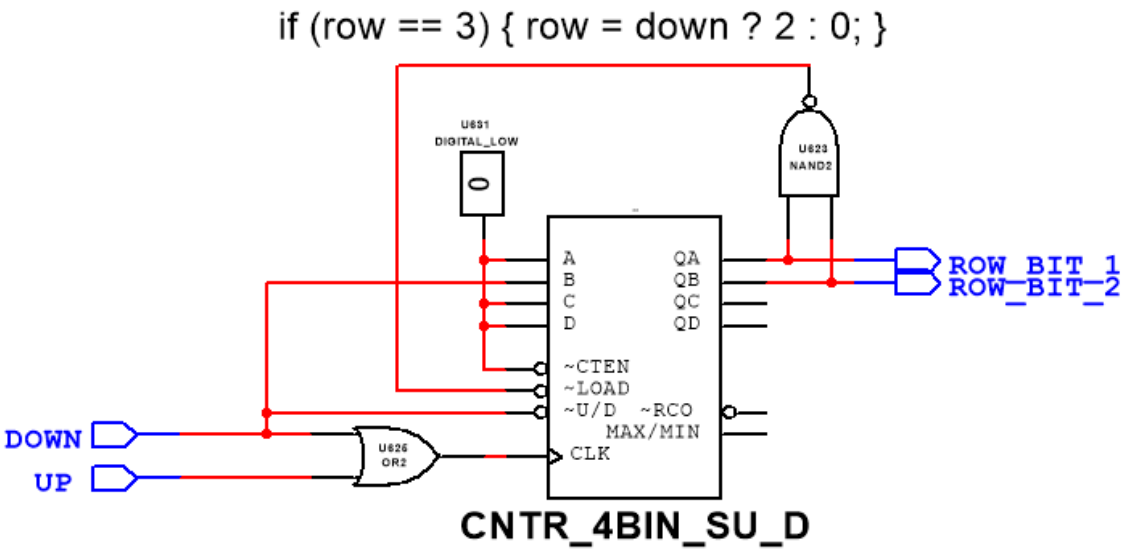
COUNT_1_TO_SQUARE_COORDS

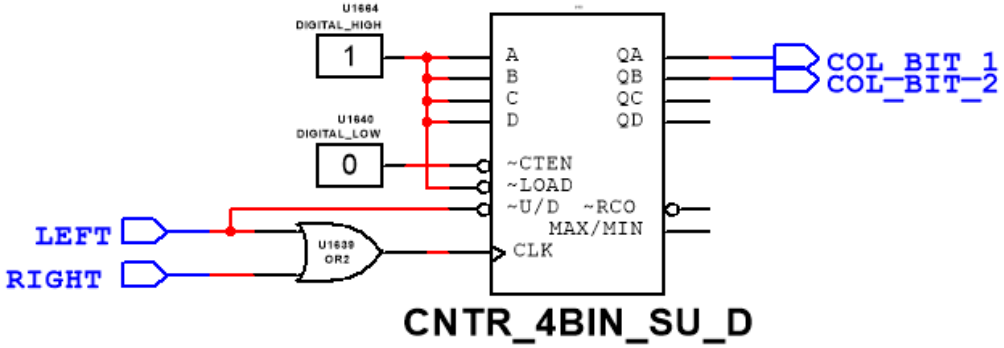


COUNT_1_TO_SQUARE_COORD

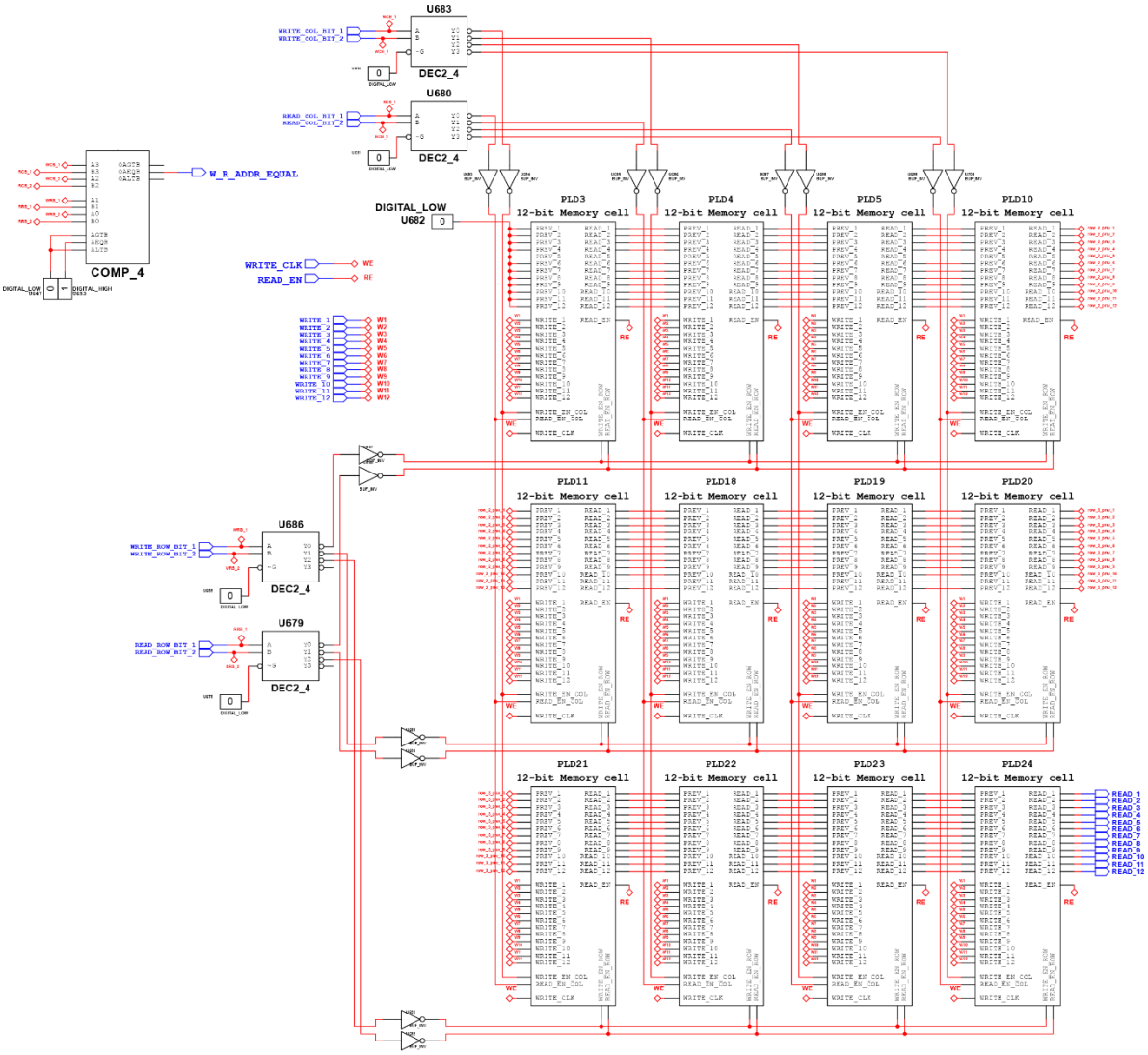


COL_ROW_SELECTOR

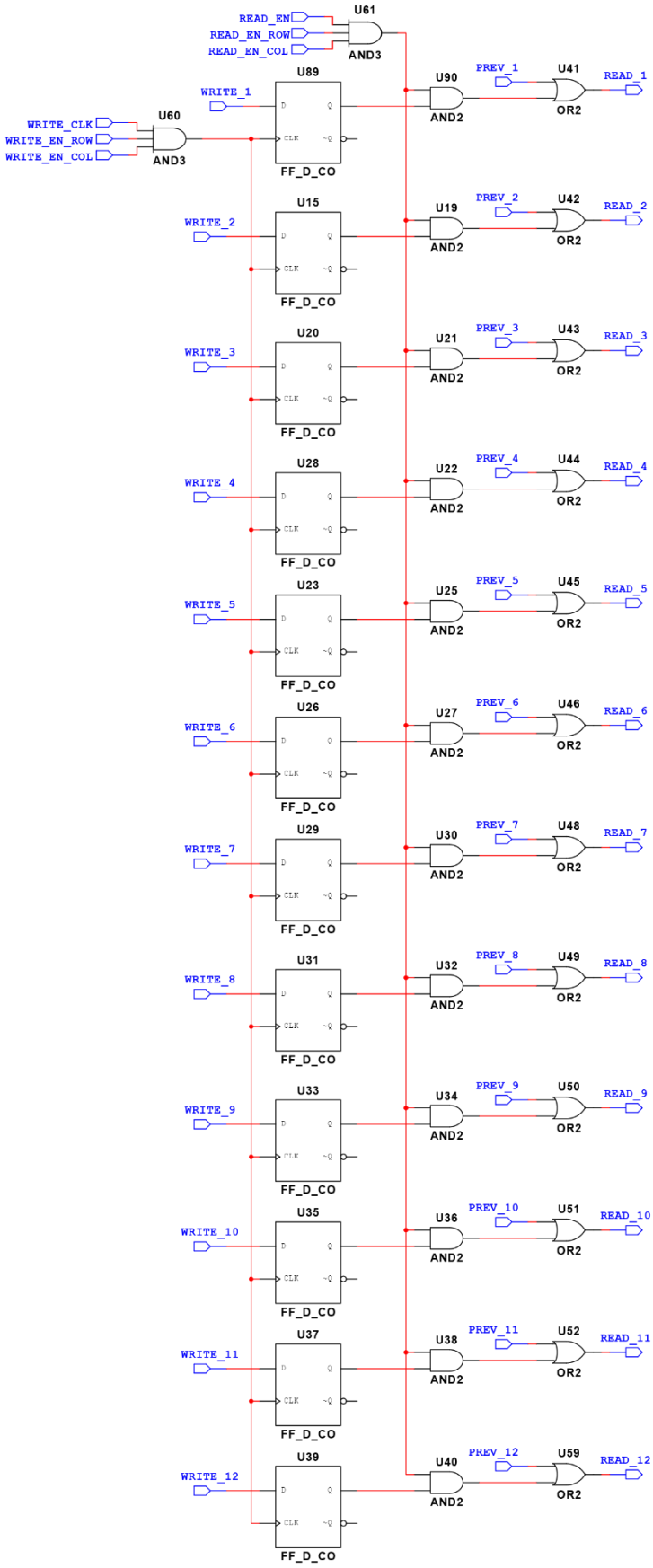




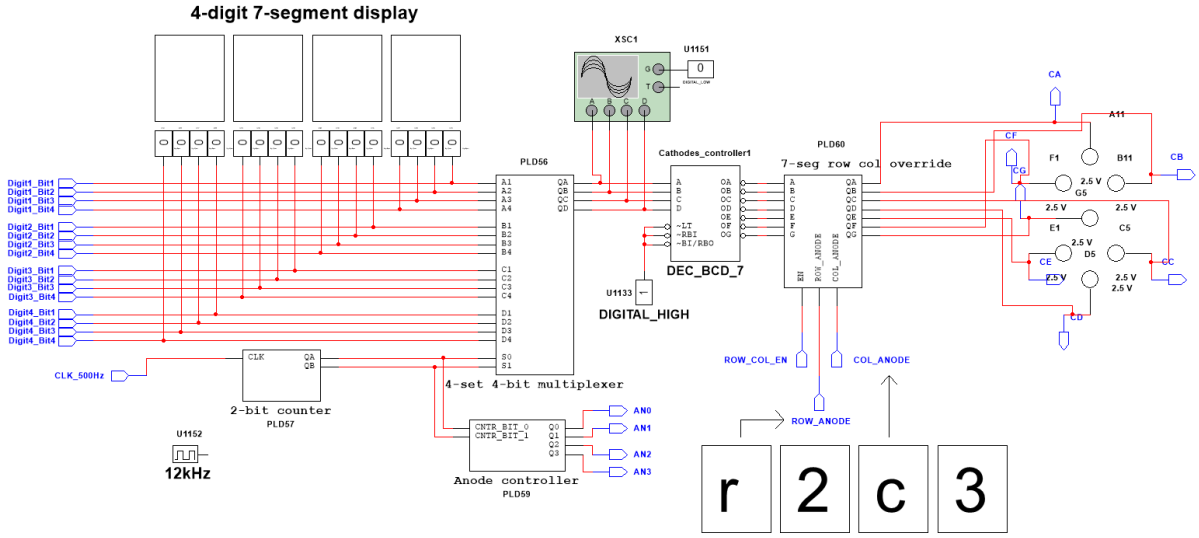
MEMORY


U683
DEC2_4
U680
DEC2_4
COMP_4
W_R_ADDR_EQUAL
DIGITAL_LOW
U682
WRITE_CLK
READ_EN
PLD3
PLD4
PLD5
PLD10
12-bit Memory cell
PLD11
PLD18
PLD19
PLD20
U686
DEC2_4
U679
DEC2_4
PLD21
PLD22
PLD23
PLD24

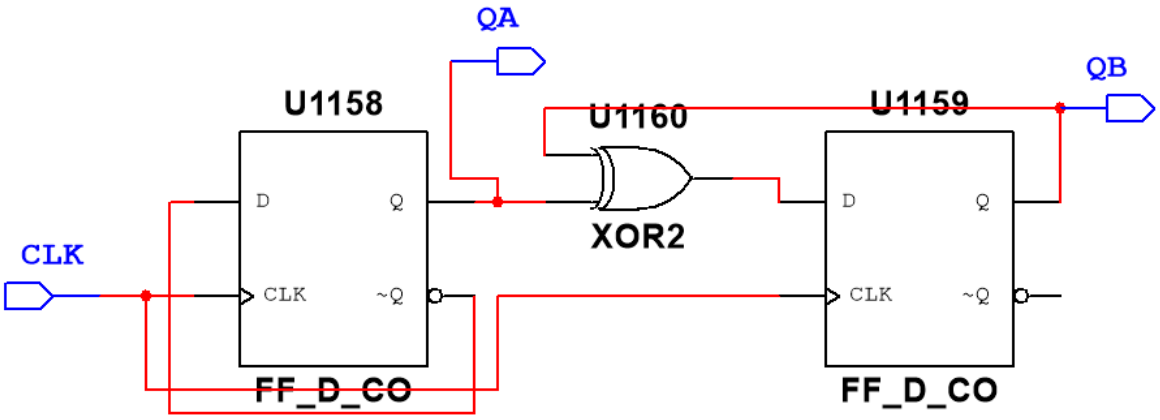
12-bit Memory cell



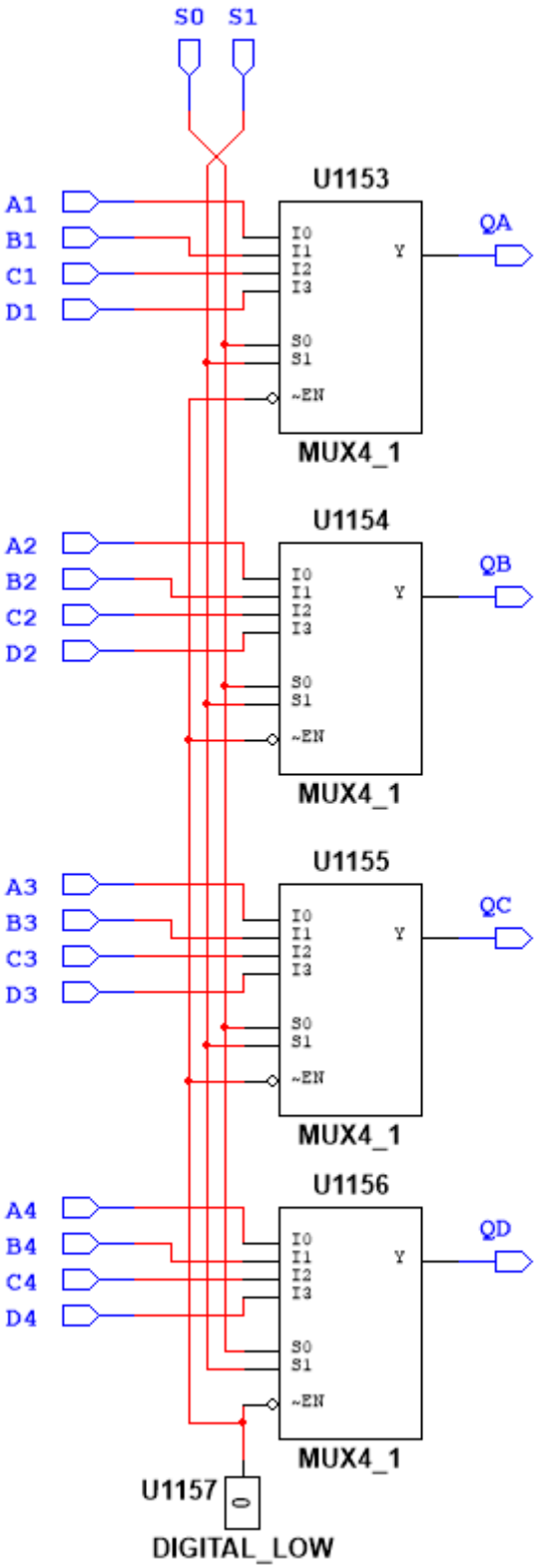
7-segment display driver



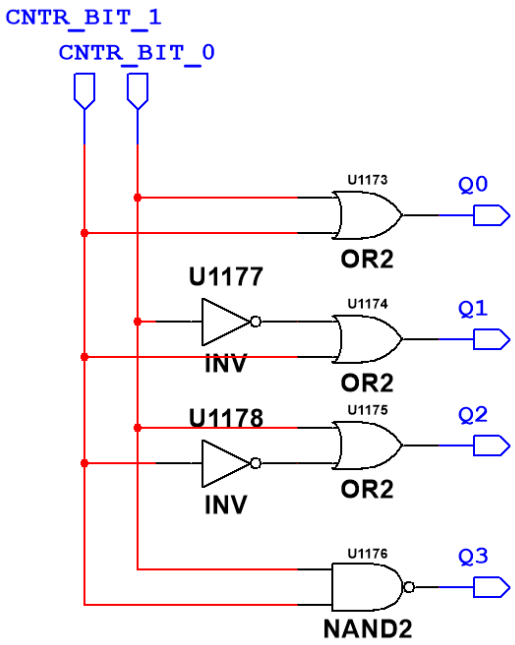
2-bit counter



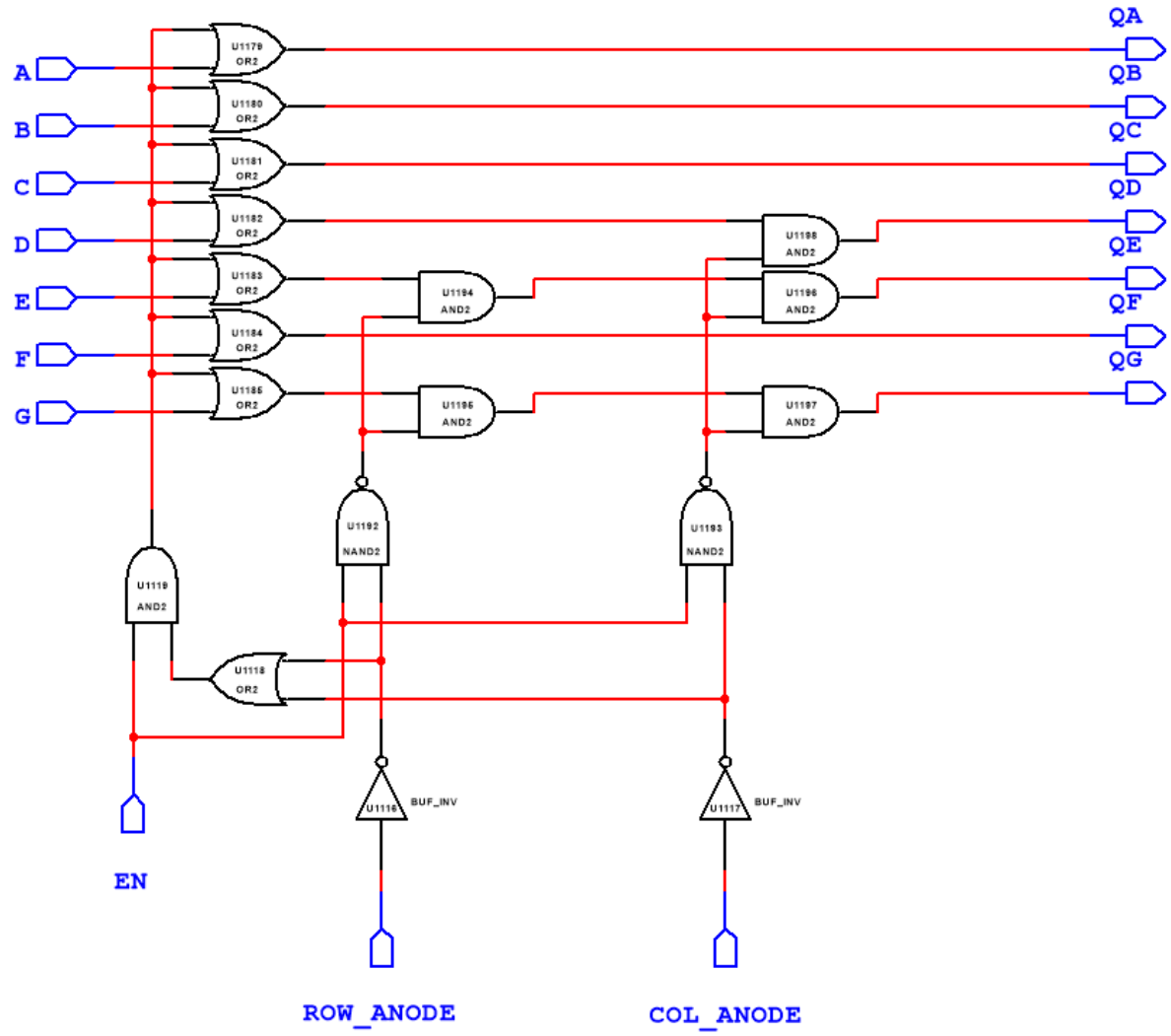
4-set 4-bit multiplexer



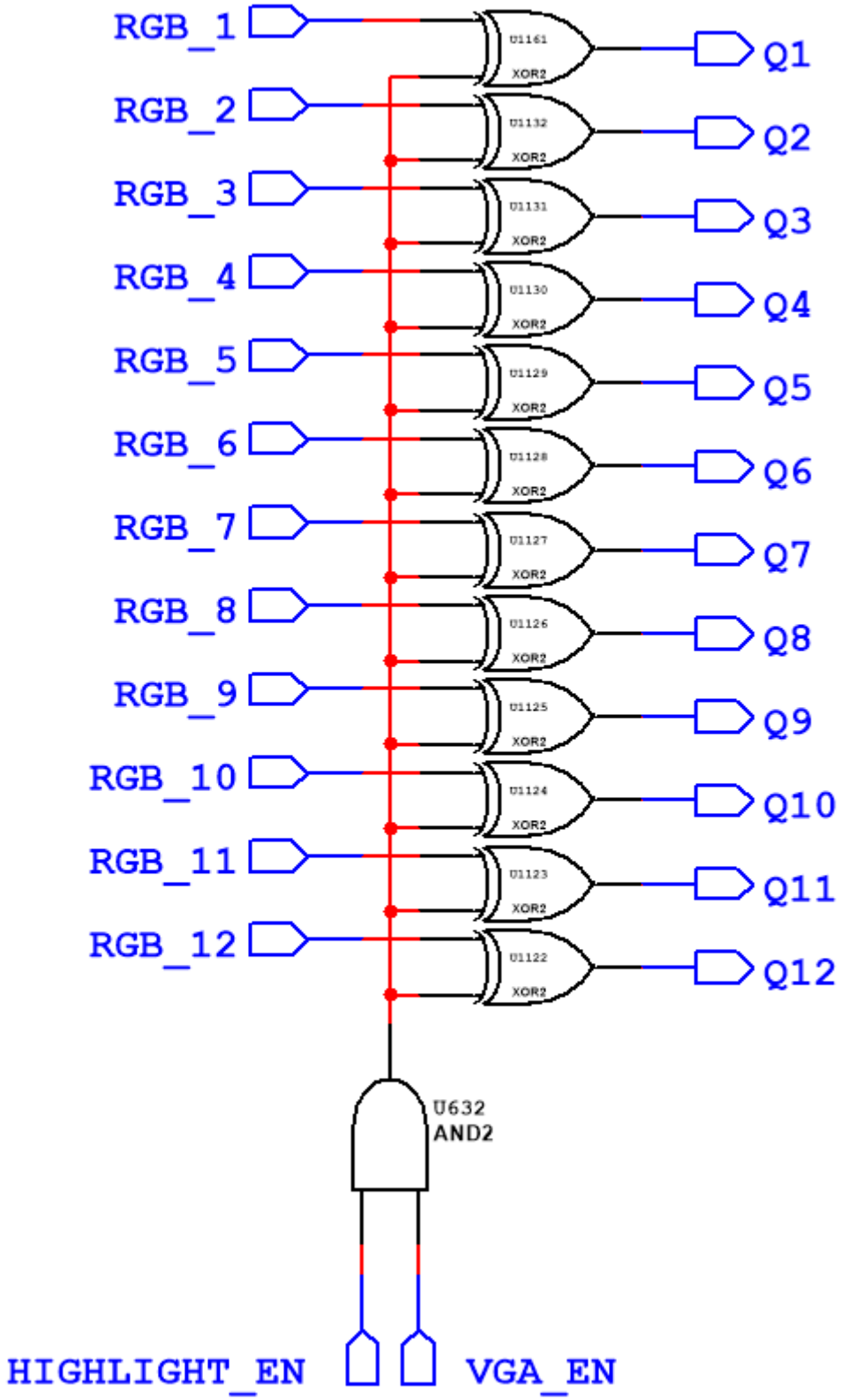
Anode controller



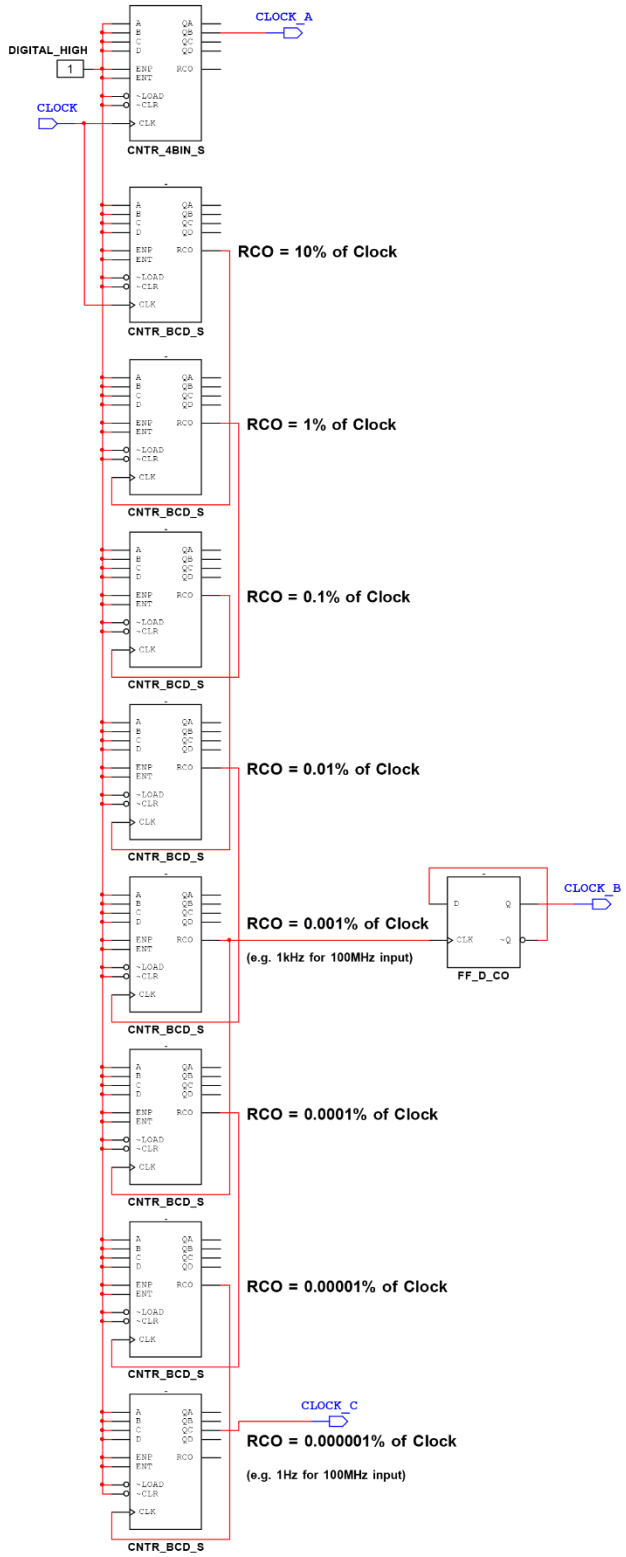
7-seg row col override



SQUARE_HIGHLIGHTER



Clock_divider



7.2 Constraints file contents

```
set_property PACKAGE_PIN W5 [get_ports CLK]
		set_property IOSTANDARD LVCMOS33 [get_ports CLK]
		create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports CLK]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets {CLK_IBUF}]

## Switches
set_property PACKAGE_PIN V17 [get_ports SW0]
		set_property IOSTANDARD LVCMOS33 [get_ports SW0]
set_property PACKAGE_PIN V16 [get_ports SW1]
		set_property IOSTANDARD LVCMOS33 [get_ports SW1]
set_property PACKAGE_PIN W16 [get_ports SW2]
		set_property IOSTANDARD LVCMOS33 [get_ports SW2]
set_property PACKAGE_PIN W17 [get_ports SW3]
		set_property IOSTANDARD LVCMOS33 [get_ports SW3]
set_property PACKAGE_PIN W15 [get_ports SW4]
		set_property IOSTANDARD LVCMOS33 [get_ports SW4]
set_property PACKAGE_PIN V15 [get_ports SW5]
		set_property IOSTANDARD LVCMOS33 [get_ports SW5]
set_property PACKAGE_PIN W14 [get_ports SW6]
		set_property IOSTANDARD LVCMOS33 [get_ports SW6]
set_property PACKAGE_PIN W13 [get_ports SW7]
		set_property IOSTANDARD LVCMOS33 [get_ports SW7]
set_property PACKAGE_PIN V2 [get_ports SW8]
		set_property IOSTANDARD LVCMOS33 [get_ports SW8]
set_property PACKAGE_PIN T3 [get_ports SW9]
		set_property IOSTANDARD LVCMOS33 [get_ports SW9]
set_property PACKAGE_PIN T2 [get_ports SW10]
		set_property IOSTANDARD LVCMOS33 [get_ports SW10]
set_property PACKAGE_PIN R3 [get_ports SW11]
		set_property IOSTANDARD LVCMOS33 [get_ports SW11]
set_property PACKAGE_PIN W2 [get_ports SW12]
		set_property IOSTANDARD LVCMOS33 [get_ports SW12]
set_property PACKAGE_PIN U1 [get_ports SW13]
		set_property IOSTANDARD LVCMOS33 [get_ports SW13]
set_property PACKAGE_PIN T1 [get_ports SW14]
		set_property IOSTANDARD LVCMOS33 [get_ports SW14]
set_property PACKAGE_PIN R2 [get_ports SW15]
		set_property IOSTANDARD LVCMOS33 [get_ports SW15]


## LEDs
set_property PACKAGE_PIN U16 [get_ports LED0]
		set_property IOSTANDARD LVCMOS33 [get_ports LED0]
set_property PACKAGE_PIN E19 [get_ports LED1]
		set_property IOSTANDARD LVCMOS33 [get_ports LED1]
set_property PACKAGE_PIN U19 [get_ports LED2]
		set_property IOSTANDARD LVCMOS33 [get_ports LED2]
set_property PACKAGE_PIN V19 [get_ports LED3]
		set_property IOSTANDARD LVCMOS33 [get_ports LED3]
set_property PACKAGE_PIN W18 [get_ports LED4]
		set_property IOSTANDARD LVCMOS33 [get_ports LED4]
set_property PACKAGE_PIN U15 [get_ports LED5]
		set_property IOSTANDARD LVCMOS33 [get_ports LED5]
set_property PACKAGE_PIN U14 [get_ports LED6]
		set_property IOSTANDARD LVCMOS33 [get_ports LED6]
set_property PACKAGE_PIN V14 [get_ports LED7]
		set_property IOSTANDARD LVCMOS33 [get_ports LED7]
set_property PACKAGE_PIN V13 [get_ports LED8]
		set_property IOSTANDARD LVCMOS33 [get_ports LED8]
set_property PACKAGE_PIN V3 [get_ports LED9]
		set_property IOSTANDARD LVCMOS33 [get_ports LED9]
set_property PACKAGE_PIN W3 [get_ports LED10]
		set_property IOSTANDARD LVCMOS33 [get_ports LED10]
set_property PACKAGE_PIN U3 [get_ports LED11]
		set_property IOSTANDARD LVCMOS33 [get_ports LED11]
set_property PACKAGE_PIN P3 [get_ports LED12]
		set_property IOSTANDARD LVCMOS33 [get_ports LED12]
set_property PACKAGE_PIN N3 [get_ports LED13]
		set_property IOSTANDARD LVCMOS33 [get_ports LED13]
set_property PACKAGE_PIN P1 [get_ports LED14]
		set_property IOSTANDARD LVCMOS33 [get_ports LED14]
set_property PACKAGE_PIN L1 [get_ports LED15]
		set_property IOSTANDARD LVCMOS33 [get_ports LED15]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports CA]
		set_property IOSTANDARD LVCMOS33 [get_ports CA]
```

```
set_property PACKAGE_PIN W6 [get_ports CB]
		set_property IOSTANDARD LVCMOS33 [get_ports CB]
set_property PACKAGE_PIN U8 [get_ports CC]
		set_property IOSTANDARD LVCMOS33 [get_ports CC]
set_property PACKAGE_PIN V8 [get_ports CD]
		set_property IOSTANDARD LVCMOS33 [get_ports CD]
set_property PACKAGE_PIN U5 [get_ports CE]
		set_property IOSTANDARD LVCMOS33 [get_ports CE]
set_property PACKAGE_PIN V5 [get_ports CF]
		set_property IOSTANDARD LVCMOS33 [get_ports CF]
set_property PACKAGE_PIN U7 [get_ports CG]
		set_property IOSTANDARD LVCMOS33 [get_ports CG]

set_property PACKAGE_PIN V7 [get_ports DP]
		set_property IOSTANDARD LVCMOS33 [get_ports DP]

set_property PACKAGE_PIN U2 [get_ports AN0]
		set_property IOSTANDARD LVCMOS33 [get_ports AN0]
set_property PACKAGE_PIN U4 [get_ports AN1]
   set_property IOSTANDARD LVCMOS33 [get_ports AN1]
set_property PACKAGE_PIN V4 [get_ports AN2]
		set_property IOSTANDARD LVCMOS33 [get_ports AN2]
set_property PACKAGE_PIN W4 [get_ports AN3]
		set_property IOSTANDARD LVCMOS33 [get_ports AN3]


##Buttons
set_property PACKAGE_PIN U18 [get_ports BTNC]
		set_property IOSTANDARD LVCMOS33 [get_ports BTNC]
set_property PACKAGE_PIN T18 [get_ports BTNU]
		set_property IOSTANDARD LVCMOS33 [get_ports BTNU]
set_property PACKAGE_PIN W19 [get_ports BTNL]
		set_property IOSTANDARD LVCMOS33 [get_ports BTNL]
set_property PACKAGE_PIN T17 [get_ports BTNR]
		set_property IOSTANDARD LVCMOS33 [get_ports BTNR]
set_property PACKAGE_PIN U17 [get_ports BTND]
		set_property IOSTANDARD LVCMOS33 [get_ports BTND]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTNC_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTNR_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTNU_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTND_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets BTNL_IBUF]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets SW0_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets SW1_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets SW2_IBUF]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets SW3_IBUF]

##Pmod Header JC
##Sch name = JC1
set_property PACKAGE_PIN K17 [get_ports JC0]
		set_property IOSTANDARD LVCMOS33 [get_ports JC0]
##Sch name = JC2
set_property PACKAGE_PIN M18 [get_ports JC1]
		set_property IOSTANDARD LVCMOS33 [get_ports JC1]
##Sch name = JC3
set_property PACKAGE_PIN N17 [get_ports JC2]
		set_property IOSTANDARD LVCMOS33 [get_ports JC2]
##Sch name = JC4
set_property PACKAGE_PIN P18 [get_ports JC3]
		set_property IOSTANDARD LVCMOS33 [get_ports JC3]
##Sch name = JC7
set_property PACKAGE_PIN L17 [get_ports JC4]
		set_property IOSTANDARD LVCMOS33 [get_ports JC4]
##Sch name = JC8
set_property PACKAGE_PIN M19 [get_ports JC5]
		set_property IOSTANDARD LVCMOS33 [get_ports JC5]
##Sch name = JC9
set_property PACKAGE_PIN P17 [get_ports JC6]
		set_property IOSTANDARD LVCMOS33 [get_ports JC6]
##Sch name = JC10
set_property PACKAGE_PIN R18 [get_ports JC7]
		set_property IOSTANDARD LVCMOS33 [get_ports JC7]

##VGA Connector
set_property PACKAGE_PIN G19 [get_ports VGARED0]
		set_property IOSTANDARD LVCMOS33 [get_ports VGARED0]
set_property PACKAGE_PIN H19 [get_ports VGARED1]
		set_property IOSTANDARD LVCMOS33 [get_ports VGARED1]
set_property PACKAGE_PIN J19 [get_ports VGARED2]
```

```
		set_property IOSTANDARD LVCMOS33 [get_ports VGARED2]
set_property PACKAGE_PIN N19 [get_ports VGARED3]
		set_property IOSTANDARD LVCMOS33 [get_ports VGARED3]
set_property PACKAGE_PIN N18 [get_ports VGABLUE0]
		set_property IOSTANDARD LVCMOS33 [get_ports VGABLUE0]
set_property PACKAGE_PIN L18 [get_ports VGABLUE1]
		set_property IOSTANDARD LVCMOS33 [get_ports VGABLUE1]
set_property PACKAGE_PIN K18 [get_ports VGABLUE2]
		set_property IOSTANDARD LVCMOS33 [get_ports VGABLUE2]
set_property PACKAGE_PIN J18 [get_ports VGABLUE3]
		set_property IOSTANDARD LVCMOS33 [get_ports VGABLUE3]
set_property PACKAGE_PIN J17 [get_ports VGAGREEN0]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAGREEN0]
set_property PACKAGE_PIN H17 [get_ports VGAGREEN1]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAGREEN1]
set_property PACKAGE_PIN G17 [get_ports VGAGREEN2]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAGREEN2]
set_property PACKAGE_PIN D17 [get_ports VGAGREEN3]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAGREEN3]
set_property PACKAGE_PIN P19 [get_ports VGAHSYNC]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAHSYNC]
set_property PACKAGE_PIN R19 [get_ports VGAVSYNC]
		set_property IOSTANDARD LVCMOS33 [get_ports VGAVSYNC]
```