VGA MODULE REPORT

December 8, 2017

Daan de Groot
4607414

Emiel van der Meijs
4567528

Contents

1 INTRODUCTION

The following report will provide information on the specifications, structure and design of VGA module used by project group A4 as part of EPO 3. A general overview of the project and the other modules of which it is built up can be found in the previously released design specification report. For this report we will only discuss other modules or general chip behaviour to clarify connections and design decisions. This report starts with describing each component making up the VGA module and will be followed up by looking at several test benches and the corresponding results when testing these components. Finally we take a look at the observed behaviour once implemented on an FPGA. Following this report the design will be implemented on a sea of gates integrated circuit using the OCEAN[1] software package. Information on this will provided in the upcoming final report.

2 BACKGROUND INFORMATION

2.1 VGA protocol

The VGA or video graphics array standard is a display communication protocol used for driving relatively low resolution displays but ideal for our use case due to it's simple and small implementation. Data gets transmitted to the display as an analogue signal over three wires representing the the amount of red green and blue in colour of the respective pixel. The positioning of pixels gets controlled by the horizontal and vertical sync signals, originating from the time of the CRT monitor which used an electron beam deflected by coils to 'draw' an image line after line onto a fluorescent screen. At the end of each line the beam would have to track back to the left of the screen and start again, this time a pixel lower. At the end of each line the horizontal sync signal has to be lowered since this would cause the horizontal coil to discharge and at the end of the frame the vertical sync signal to discharge that coil to start again at the top-left with the next frame. A visual representation of this is given in figure 1 and a table 1 contains the corresponding timing values for the standard resolution of 640 by 480 at a refresh rate of 60 Hz. Today's digital monitors have controllers instead of analogue circuitry, but still use the two sync signals to associate the incoming colour data to the pixels on the screen.

	Horizontal timing		Vertical timing	
Scanline part	Pixels	Time [µs]	Pixels	Time[m s]
Visible area	640	25.42	480	15.25
Front porch	16	0.64	10	0.32
Sync pulse	96	3.81	2	0.064
Back porch	48	1.91	33	1.05
Whole line	800	31.78	525	16.68

Table 1: VGA timings (640x480@60Hz) *from [3]*

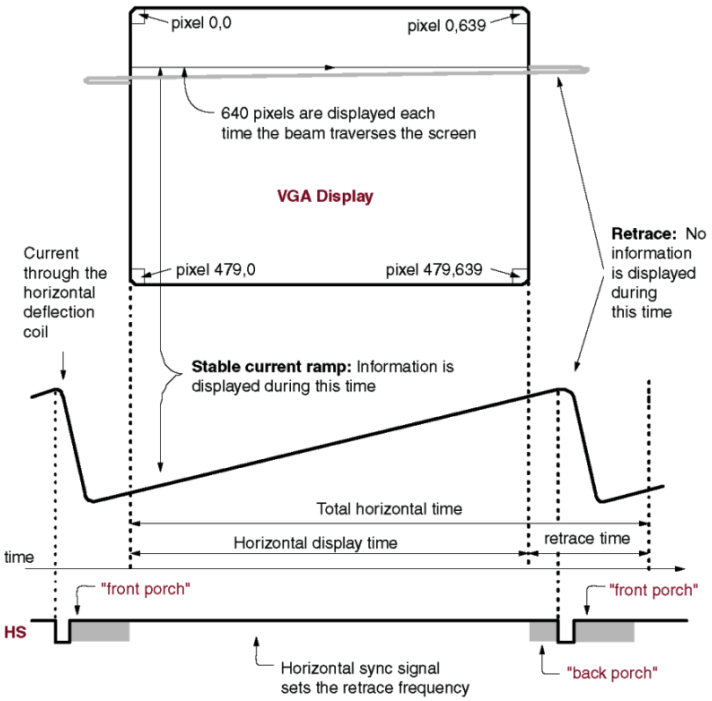

Figure 1: Visualisation of VGA protocol *from [2]*

2.2 FPGA board

For fast prototyping an FPGA board is used. FPGA boards contain many modular logic units, which can be 'programmed' to implement logic functions as desired. Since the connections between these logic blocks are also fully programmable, as long as there are enough logic blocks and the required connections between these blocks are available, any functionality can easily be programmed using only a VHDL description for the design and a pin configuration file to connect VHDL signals to exterior ones. This board also has a VGA port built-in, making it very suitable for prototyping this module.

3 DESIGN SPECIFICATIONS

The VGA module's primary objective is to print out the playing field. A secondary objective is to print out the players current score. They are supposed to be laid out on the screen as drawn up in figure 2. To vary the game difficulty, the amount of blocks on the field should be able to vary from 4x4 to 7x7 by means of a 2 bit signal, varying from binary 0 to 3 respectively, connected to the VGA module. The colours of the blocks making up this playing field are stored in the chip's storage module, from which individual colours stored can be requested. This is done by providing the corresponding address to its input and setting the flag by means of having the set flag signal high during a rising edge of the global clock. When the data sent back by the storage module is valid the flag will be set to a digital zero again. During this time the VGA module is responsible for keeping the requested address available.
To be in sync with the display and comply with standard timings as described in section 2.1 the module uses the clock signal relying on the fact that it will be running at 6.144 MHz. Besides this, the clock will be used to keep communications with other components synchronised. Another example of this is how the score gets advanced when a button is pressed. When a digital 1 is presented to the VGA module by the controller during a rising edge of the clock the score needs to get incremented. When however the game restart is high a new game has been started and the score needs to be reset. Also, because a new field design gets generated during this time, the field must not be drawn to prevent displaying garbled data.
The VGA module will be able to send colours to the display over a three bit bus where a digital 1 corresponds to the inclusion of either red, green or blue in the pixel being addressed at that time. Lastly every register included in the VGA module features a reset, which when brought high, clears all values from the registers and sets FSMs to their idle state.

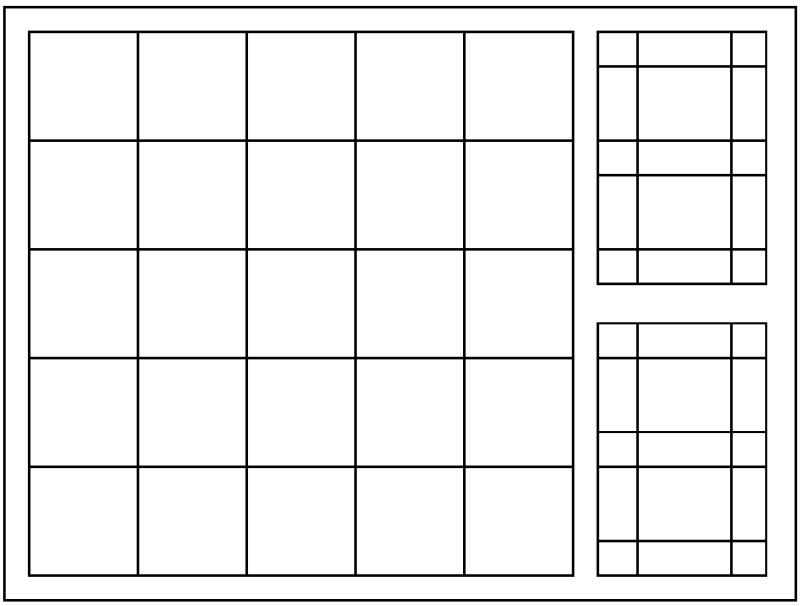

Figure 2: Game layout on a 4 by 3 display

4 DESIGN STRUCTURE

The VGA module consists of multiple blocks all implementing different functions. Therefor these blocks will be written and described separately. The diagram in figure 3 shows how all the blocks are connected to each other and to external signals. In the remainder of this section the functionality of each sub module will be described together with relevant signals.

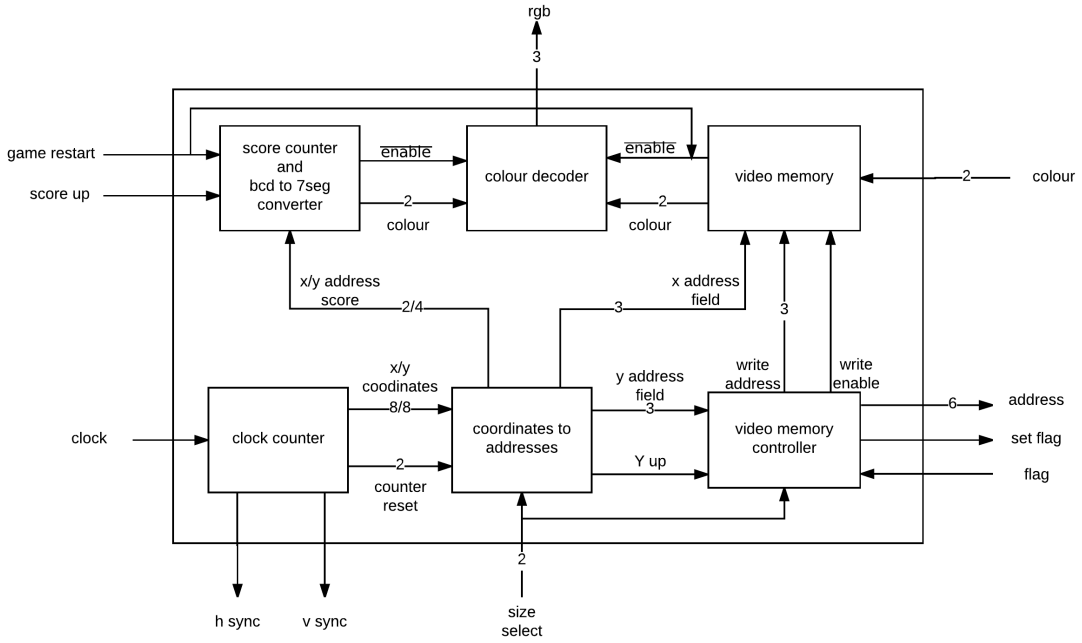

Figure 3: Structure of the VGA module design

4.1 Clock counter

This sub module is responsible of both creating the timing framework onto which the VGA frames can be constructed, as well as creating the synchronisation signals as required by the monitor's VGA control chip to function.

To create the framework, 2 counters are used. These counters, the X and Y counter, track the position of the 'beam' on the screen in the respective directions. The position values relates to a fourth of the value in screen pixels, starting top-left. This maximum of the effective resolution is an effort to shrink bus widths and more importantly a limit caused by the pixel rate for the 640 by 480 resolution at 60 Hz being 25.175 Mpixels/second where the circuit, however, will only have access to a 6.144 MHz clock after production. This causes the colour only being able to change about every 4th pixel. Dividing the horizontal timing pixel values on the left side of table 1 by 4 and adding them together tells us that the maximum value the X counter will contain is 196 which can be represented by an 8 bit binary number.

For the structure of this counter we decided to use a simple structure of cascaded T flip flops and AND gates as is shown in figure 4 for four bits. Having the first connected to a digital 1 makes this counter add 1 to its previous value on every rising edge of the clock. To make the counter reset itself once it reaches the end of the line at value 196 (or 11000100 in binary), given that it will only count upwards and as long as we implement flip flops

with a synchronous reset, the reset signal is simply the AND function of bits 7, 6 and 2.

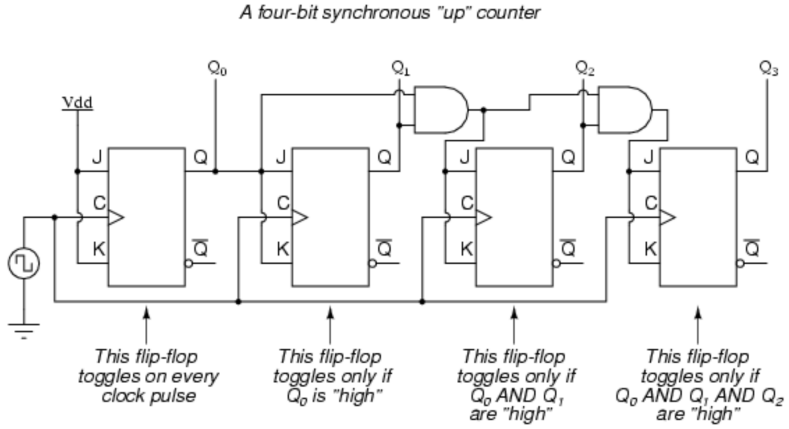

Figure 4: Four bit T flip flop counter *from [4]*

Since the Y counter has to be able to keep track of the number of complete rows drawn every frame, a maximum of 525, the corresponding counter has to be ten bits wide. The structure of this counter is very similar to the one used for the X coordinate with the one main exception that the first flip flop is connected to the reset signal of the X counter, causing this counter only to increment every time while the X counter gets reset. Because this counter doesn't advance every clock pulse, the logic generating the reset signal for this counter will not only be the AND function of bits 9, 3, 2 and 0, but also the X counter's reset signal, causing the counter only to return to zero at the moment it would otherwise have advanced to decimal value 526.
This counter's value now represents the exact row of the screen and so the pixels corresponding to a virtual coordinate would not be square (but instead 4 by 1). To solve this problem only bits 9 downto 2 of the Y counter will be available as output of the component, practically dividing the vector's value by four and so keeping these 'virtual pixels' square and creating the effective resolution of about 160 by 120.

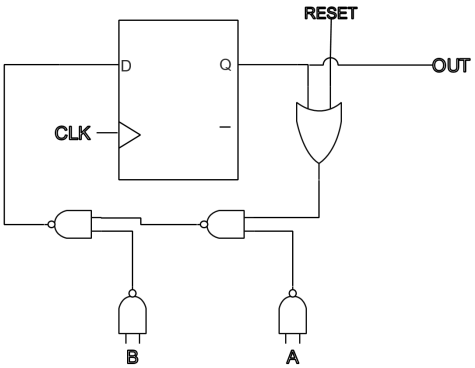

Figure 5: Synchronisation signal FSM

Figure 6: Counter values in relation to synchronisation signals

The generation of the horizontal and vertical synchronisation signals, discussed in section 2.1, is done by two, two state FSMs where the bits stored in the registers directly represent the synchronisation signals. The next value of the horizontal and vertical synchronisation signals is calculated by a combinatorial network based on the current value of the X and Y counter respectively. A visualisation of how this is implemented is shown in figure 5.

Again using the fact the the counters will only count up, the bit set A connected to the first NAND is chosen in such a way that so that at the last counter value before the sync pulse the output of the NAND will be zero for the first time since the counter reset. For B this will be the last counter value that is part of the sync pulse. For the horizontal synchronisation signal these bit are chosen in such a way that the visible part of the screen starts at zero. To the same for the vertical synchronisation signal would require a large fan in NAND B. Instead the start of the synchronisation pulse is shifted towards the last values that the counter attains in such a way that the last Y value of the pulse is zero. This way the logic only has to check whether the counter has reset instead of looking for a specific value. The result however is that the first Y coordinate visible on the screen will be 33, or 8 once re-scaled to a 8 bit vector. The exact way the synchronisation signals are lined up with the counter values is visualised in figure 6.
Lastly this module houses two compactors, one connected to each counter, that create a pulse when the counter's value points to a pixel on the not visible area of the screen. This signal will be used by other components to prevent registers using the X and Y coordinates from running out of sync. This will be explained further in section 4.2. The VHDL code implementing this component can be found in appendix D.2.

4.2 Coordinates to addresses

This component is responsible for for converting the values of both the X- and Y counter, discussed in in the previous section, to values with direct correlation to the memory address storing the colour that has to be displayed. The 6 bit memory addresses used by the storage module are the X and Y coordinate of a square on the playing field added after each other. Thus for example address 9 for the upper left square and address 63 for the bottom right one in case of a 7x7 playing field. The addresses used for the segments of the score work in the same way. Because of the way the address is build up we can reuse the component structure for for both the X and Y part of the address of the blocks on the playing field as well as the segments making up the score. The structure designed for this task is visualised in figure 7. It functions in the following manner.

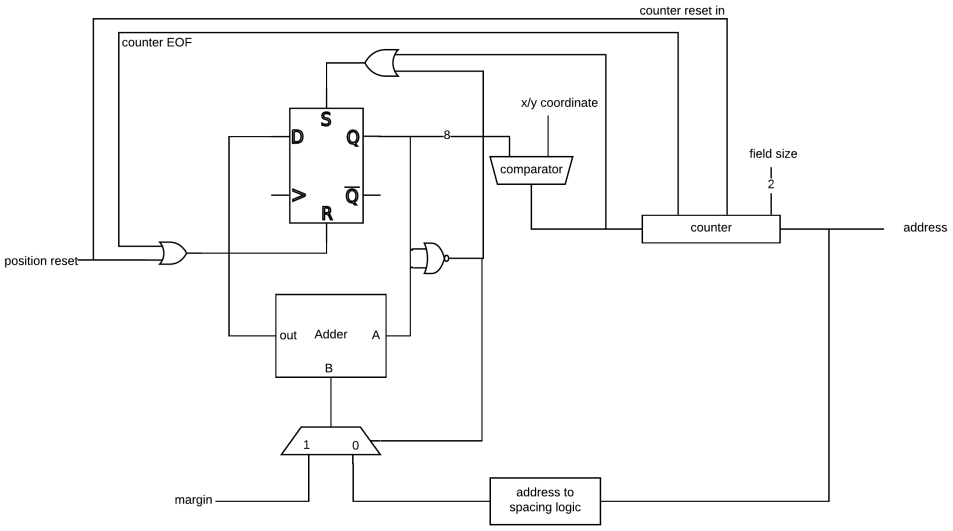

Figure 7: Coordinate to address converter basic structure

The register saves the next value that the incoming counter signal needs to equal to before the address counter gets incremented. The address counter is constructed the same way as the Y counter discussed in section

4.1. The counters used for the field addresses are 3 bits wide and feature a variable reset value depending on the value of the field size signal. The converters for the score use two different counters. The X variant uses a two bit counter and resets after reaching three. The Y variant uses a both a three and a one bit counter the latter counting to five before resetting and incrementing the one bit counter. This way the value of the one bit counter indicates which one of the numbers making up the score is being displayed while the remainder of the address will be the same for the same segments of the different numbers. This is displayed in figure 8.

When a counter gets incremented a new value will be loaded into the register at the same time. When the address counter reaches it's maximum value and resets itself, it will reset the register thus making it impossible for the address to get incremented until also the counter is is connected to has been reset as well. When the register value equals zero the nor gate will output a digital one and so load the margin value into the register. This value will thus determines the coordinate at which the monitor will start drawing.

The address to spacing logic calculates the spacing between the next two coordinates at witch the address gets incremented based on the current address value. For the playing field this will only be a function of the field size. This function is determined by an Espresso script, the code of which can be found in appendix C, and implemented as the dimension conversion entity. In case of the score however the height and width of the segments differs and thus the feedback of the address will be used. The codes of Espresso scrips deriving these function can be found in the same appendix.

This setup causes any area outside the region that has to be drawn to have either the X or Y part of the address zero. This will be used later to enable or disable the colour decoder, discussed in section 4.3. This also allows the X and Y reset signals generated inside the clock counter and discussed at the end of section 4.1 to be the reset of both the address counters and registers of the associated coordinate to address converters. This restores the registers and counters to the state they should be in, in case they got out of sync by accident.

A last thing to note is the fact that the converter used for the Y address of the field also directly outputs the its compactor output. This signal will later get used in the video memory controller further explained in section 4.4. The VHDL code implementing each of the components discussed in this section can be found in appendix D.3.

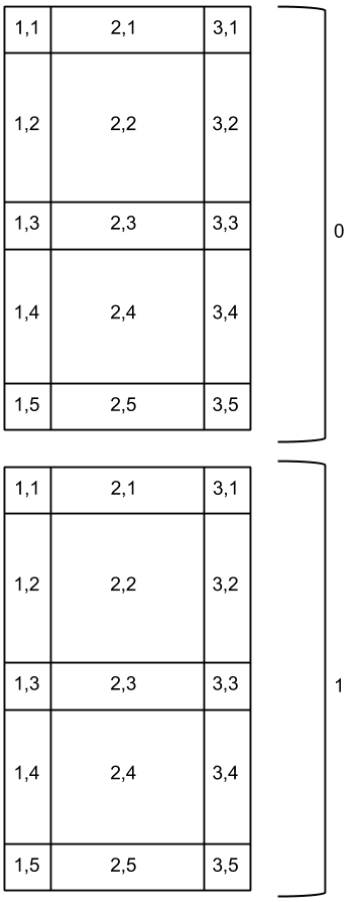

Figure 8: Seven segment coordinate system with to the right Y address bit 3 and inside each segment bit 2 downto 0 and the X address

4.3 Video memory

In the design specification report it has been stated that sending a read request to the storage module for each virtual pixel, since the clock frequency of the final chip is equal to the virtual pixel rate, will result in a read request every clock cycle, thus disabling the capability of the storage to answer requests sent by other modules. To solve this problem the video memory module is implemented, with it's control unit the video memory

controller described in subsection 4.4. This module contains a 7*2-bit register bank, making it possible to store the entire row of 7 2-bit colours which is currently being drawn. This way only a burst of read requests needs to be sent after surpassing the edge between rows of the playing field.
To specify which 2-bit colour is put on the bus connected to the colour decoder module, a decoder connected to the x-address is used. The one-hot output of this 3 to 7 decoder is connected to tri-state buffer pairs which are in turn connected to the outputs of the register bank. To facilitate writing colours into the register bank, another decoder is used. This decoder is connected to an address bit vector and enable bit provided by the video memory controller.
Since all 4 possibilities of the 2-bit colour vector are used as playing fields colours, extra functionality has to be implemented to create a background colour around the playing field. Since for the duration of the front- and back-porch all 3 RGB signals need to be off, the background colour is chosen to be black as well. To implement this a enable signal is wired next to the colour bus, making it possible to disable all three of the RGB signals for the area outside the playing field and porch area. This signal is driven by an OR function of two comparators which compare the current row and column number with '000' and the game reset signal. This last signal is a global user input signal which can be executed to start a new game. When this signal is high, a new map is being generated in the storage module, making it undesirable to display these fast-changing colours and therefor the RGB output is disabled.
The VHDL code implementing this component can be found in appendix D.6.

4.4 Video memory controller

The video memory controller is responsible for loading the colours of the row of blocks that has to be displayed into the video memory. It can request this data from the storage module sequentially by putting the the address of a block on the address line and setting the flag. As a response to the set flag signal being high during an rising edge of the clock the flag signal will go high. The flag will get detected by the storage module and as a response it will determine the corresponding colour, store it in it's output register en clearing the flag. The data will stay stored there until a new request has been made. Important is that the VGA module is responsible for keeping the address available during the clock period used to determine the colour. A visualisation of the protocol with respect to the clock signal is showed in figure 9.

The circuit designed to perform this task is show in figure 10. Important to note is that the x and y register signals get added together to form the 6 bit address connected to the storage module. The x register signal is also connected to the video memory write address. This is very convenient because the address of which the value gets requested is the same as the one it has to get written to in the video memory. Because of this the bus returning from the storage carrying the requested colour can be connected directly to the write bus of video memory. This has the benefit that the video memory controller never has to specifically write any data since it can be seen a side effect of requesting data as long as write enable is set to a digital 1.
The FSM controlled functionality of the circuit is as follows. Every time the Y up signal is high it indicates a change of the Y address on the next rising edge of the clock. This starts the cycle of loading in all colours corresponding to the row with the current Y address. This is done by setting LBL5 to 1 and then alternating between state load_set where the register gets loaded and flag set is high and state flag_wait where write enable is high and the FSM waits for the flag to go low again before returning to state load_set. This continues until during state

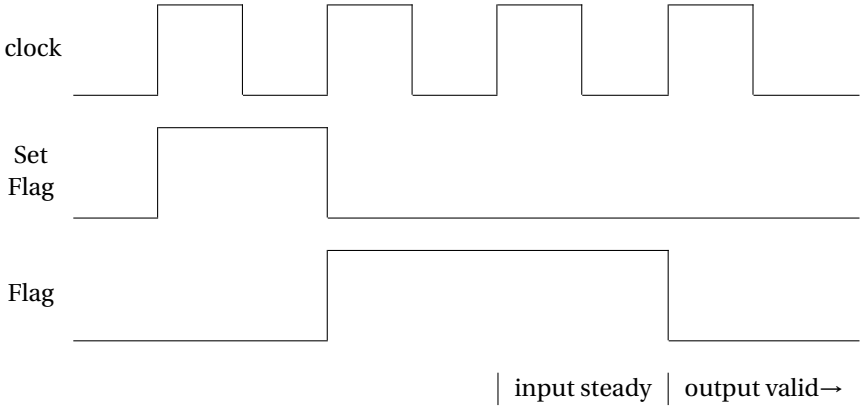

Figure 9: Timings of set flag, flag, address out and data in relative to the clock

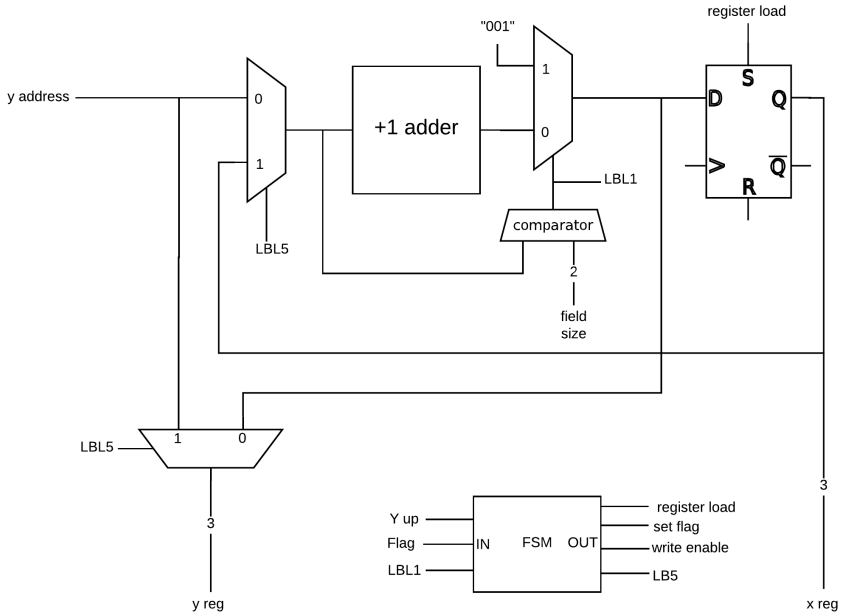

Figure 10: Video memory controller circuit

load_set signal LBL1 is high. This will cause the next state to be the idle state. In this state all the outputs are zero. However, the flag was set and thus a colour will be returned. Since the LBL5 signal changes to zero before the address is used by the storage module the requested address will be the first X address of the next row of blocks.

This is done on purpose and allows this colour being loaded into the video memory as soon as the Y value changes since it is saved in the output register of the storage module. The state diagram of the FSM described is shown in figure 11 with the output values corresponding to each state stated in table 2. As can be seen write enable is made high in the idle state when the next state will be state load_set. This is done to write the previously requested colour that is available on the write bus

	state		
Signal	Idle	load_set	wait_flag
LBL5	0	1	1
set flag	0	1	0
write enable	0	0	1
register load	0	1	0

Table 2: FSM output for each state

to be written to the first address. This technically makes this
FSM a mealy machine but since Y up is a clock synchronous signal and LBL1 a constant during the idle state it shouldn't influence the reliability of the system.
The VHDL code implementing this component can be found in appendix D.7.

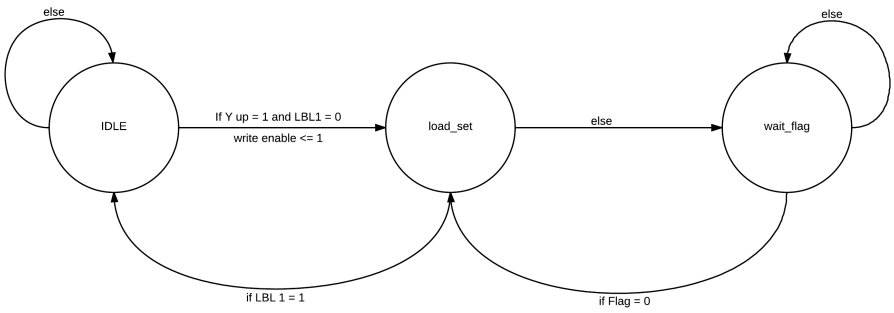

Figure 11: Video memory controller state diagram

4.5 Score counter

The objective of the flood-game is to convert the playing field to a single colour in at the least amount of moves as possible. To track the current amount of moves made a BCD-counter is used. A Binary Coded Decimal counter is a 4-bit counter that resets and increments the following counter when transitioning from the value 9 to 10, making it very suitable for counting decimal numbers. Since it's very unlikely a player would need more than 39 moves to complete the game, the dozen counter is only a 2-bit counter. These counters are implemented in the same way as the Y counter, described in chapter 4.1. The only differences being the width and the self-reset value.

To convert the BCD value to a the seven segment representation that can be displayed on the screen as shown in figure 12, a combinatorial logic block is used. This component determines the state, on or off, for each segment based on the BCD value and outputs this as a thirteen bit vector. The logic functions used for this conversion have been obtained using the espresso logic software tool and manually improving the result it returned. The espresso commands and the corresponding return can be found in appendix C.
To prevent the need to implement this component twice, one for each number, a multiplexer in used to select which decimal we use as input. This multiplexer gets controlled by bit 3 of the incoming Y address. As discussed in section 4.2 this bit indicated the current number being drawn while the remainder in combination with the X address selects the segment.
To select the correct segment the module contains a component that can best be described as specialised multiplexer. It takes as input the

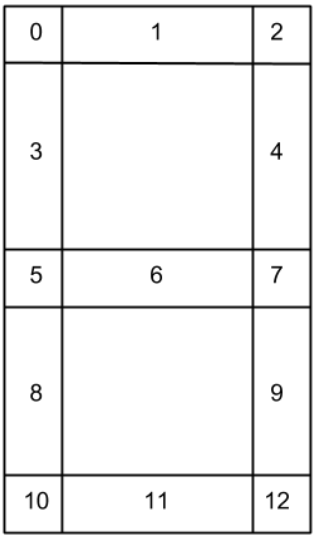

Figure 12: Numbering system for the 7-segment display

aforementioned thirteen bit vector and the score address provided by the
'coordinates to addresses' module. Based on the latter it forwards the state
of the corresponding segment toward the enable pin of colour decoder discussed in the next section. When the address does not correspond to any of the segments a digital one will be sent to the decoder causing it to not draw anything. The colour of the score is determined by the score itself since bit 2 and 1 of the four bit counter are used directly as the colour output.
The VHDL code implementing this component can be found in appendix D.8.

4.6 Colour decoder

Both the video memory and score counter as described in subsections 4.3 and 4.5 output the colours encoded in a 2-bit format as listed in table 3. The colours and the binary values for them are chosen in such a way, that they are easily converted back to the RGB values of the colours. The R and G outputs are connected directly to bit lines 1 and 0 respectively and the VGA's blue pin is connected to both lines through a nor-port, thus only turning the blue output on when both red and green are off. Since both the margins on the screen and front- and back porch areas need to be black, the colour decoder also has enable lines which are 1 for these areas, disabling the respective decoders.
The VHDL code implementing this component can be found in appendix D.4.

code	colour	RGB-value
00	blue	001
01	green	010
10	red	100
11	yellow	110

Table 3: Colour codes as used in this project with their respective RGB counterparts

5 RESULTS

5.1 Simulation by software

To prevent unexpected results while testing the VHDL description on an FPGA board, it is first tested using ModelSim. Here the generated wave forms can be analysed to verify all the RGB and synchronisation timings are according to the VGA standard as described in section 2.1. The values found during simulation are compared with the standard value in table 4. The possible cause of the small discrepancy between the standard and simulated values is the limited precision of the conversion made to the 6.144 MHz clock as well as the test bench clock not being able to run at that exact value. The waveform from which the values are substracted can found in appendix A. The last column of the table will be elaborated on in section 5.2.
The coordinates to address converters have also been tested before implementation. The waveform generated is shown in figure 13. All four structures as described in section 4.2 are connected to the same 8 bit position signal. Wave features that point toward correct operation are the fact that the amount of block addresses traversed correctly correlates to the size select input value while the score addresses do not. The placing and

	Horizontal timing [µs]			Vertical timing [ms]		
Scanline part	Standard	Simulated	Measured	Standard	Simulated	Measured
Sync pulse	3.81	3.89	3.60	0.064	0.064	0.063
Whole line	31.78	31.91	31.20	16.68	16.79	16.60

Table 4: VGA timing comparison between standard timings and measured timings

width of these also seems plausible when comparing it to figure 8. Lastly the difference in margin of the block addresses as explained in section 4.1 is visible. Based on these observations the component is found ready for implementation on the FPGA. Because the design of BCD to seven segment logic module, discussed in section

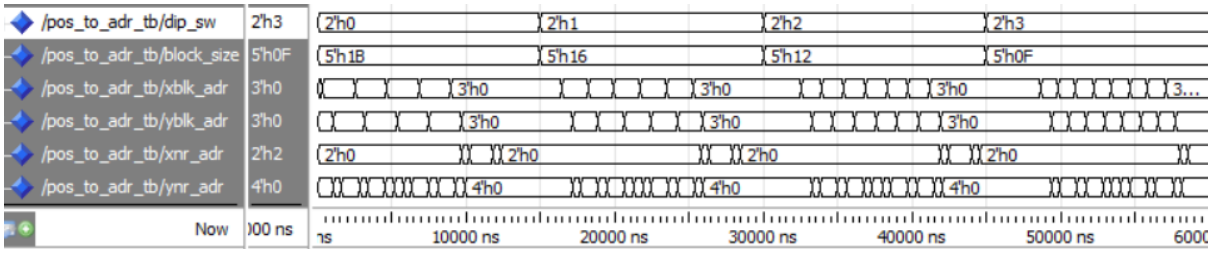

Figure 13: Coordinates to address converter test bench results

4.5, was solely based on a software generated logic function, a test bench was built. The result is shown in figure 14. For clarity every digital 1 is represented in read and a 0 in blue. The output should be compared to figure 12 while keeping in mind that a zero corresponds to a visible segment and vise versa. As can now be inferred the component functions correctly. As final step before FPGA implementation a test bench has been build for the

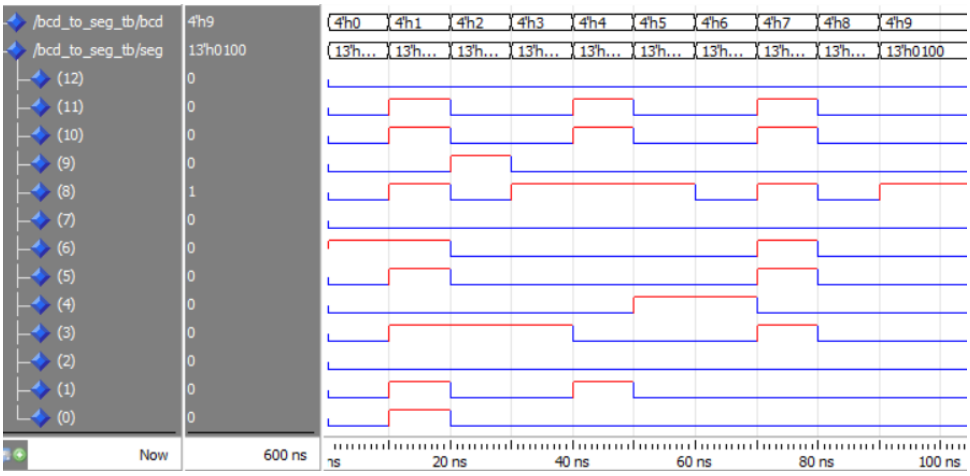

Figure 14: Test bench result for the BCD to 7 segment converter

entire VGA module. Both the actual output to the screen as well as the communication with the storage module are important to examine. The latter being shown in figure 15. It clearly shows how the communication starts at exactly the moment the Y address goes from 0 to 1, then goes through all the necessary addresses depending on the selected field size and finishing by requesting the first value on the next line. When compared to figure 9 it can be concluded that it matches perfectly to the expectations. This also gives conformation that the address is actually stable when it has to be. For verifying the actual output to the monitor the waveform of the outputs

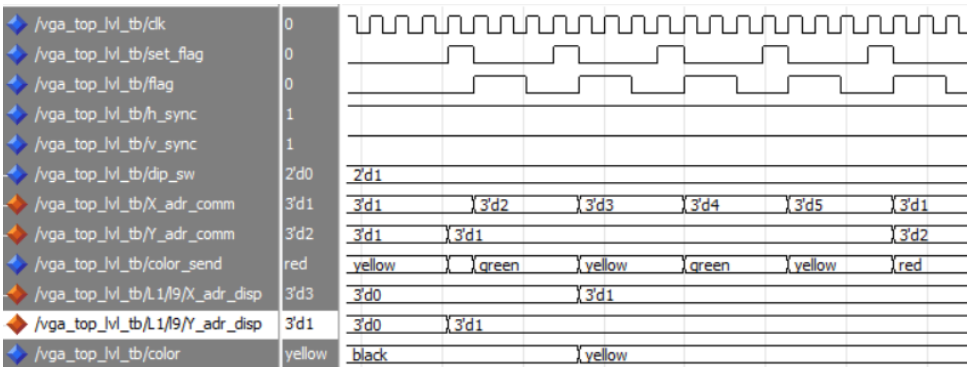

Figure 15: Test bench result showing a the communication between the VGA en storage modules

during the drawing of a single line on the display is studied. The result is shown in figure 16. The seven blocks of equal width will easily get recognised as the playing field. The two smaller lines to the right with the black in between can get identified with the score, which is zero at the moment of simulation. With this simulation looking extremely promising the module can get implemented on the FPGA for further testing.
The code of each of the test benches used can be found in appendix E.

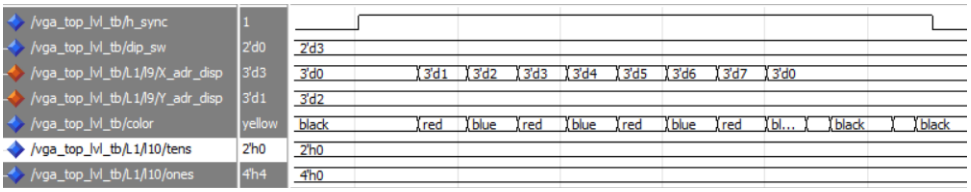

Figure 16: Test bench result showing a row of pixels

5.2 Simulation by FPGA

To further test the VGA module, the hardware description as described in this report is programmed onto an FPGA board. Appendix B shows different sync pulse measurements, which are summed up in table 4 together with the corresponding standard timing values for a resolution of 640*480@60 Hz. The amplitudes of the signals aren't listed here, but these are logical voltage values as generated by the FPGA board and are in the margins of TTL standards.
Looking at the timings and comparing these to the standard timings, these are all quite a bit shorter. This first of all isn't a problem7 since a 5% error margin exists for VGA timings[5]. Secondly, this deviation can be explained by looking at the way the 6 MHz clock is generated on the FPGA. This is done through connecting a factor 8clock divider to the FPGAs 50 MHz clock. This way a 6.25 MHz clock is created instead of the 6.144 MHz clock the VGA module has been designed for.

Figure 17 shows a screen shot of a screen connected to the FPGA programmed with the VGA module description, which functions as required, showing the full array across the screen, with no colour overlap, consistent margins and numbers represent the score saved internally.

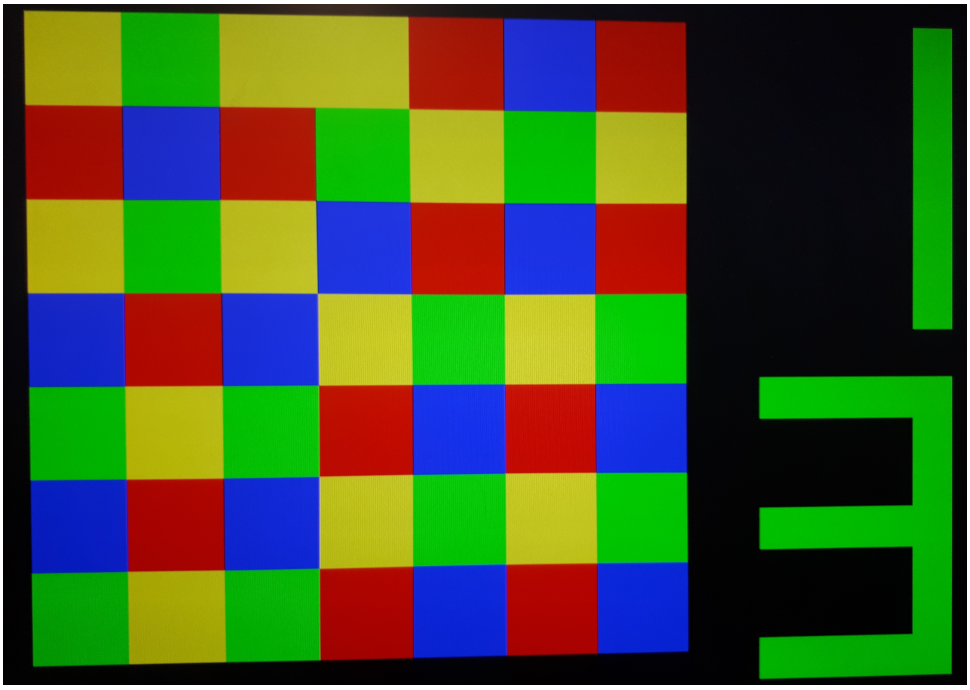

Figure 17: VGA output of an test FPGA programmed with the hardware description as described in this report

6 Conclusion

Due to the limited area of the sea-of-gates chip which will house the final version of the design, only after all modules are fully synthesised it can be determined whether the implementation of a VGA controller as described in this report meets our purposes. That said, the functionality of this VGA module is fully tested in both a software and hardware simulation and has proven to behave as expected and according to the design specifications.
Since printing the score is not an objective as described in the design specification report, this functionality could be removed during the implementation period if the chip space is insufficient to the full design. We can therefor conclude the descriptional implementation of the design to be sufficient.

References

[1] Patrick Groeneveld and Paul Stravers. Ocean: The sea-of-gates design system users manual. `http://cas.et.tudelft.nl/~simon/me/wwman/wwman.html#content`. Accessed: 2017-12-7.

[2] Mark Bowers and Michael Prechel. Asic vga controller. `https://www.markbowers.org/asic-vga/`. Accessed: 2017-12-7.

[3] Vga signal timing. `http://tinyvga.com/vga-timing`. Accessed: 2017-12-7.

[4] Synchronous counters. `https://www.allaboutcircuits.com/textbook/digital/chpt-11/synchronous-counters/`. Accessed: 2017-12-7.

[5] A.J. van Genderen S. de Graaf P. Groeneveld E.A. Hendriks N.P. van der Meijs R. Nouta C. Verhoeven J. Liedorp A. Bakker, A. Frehe. *Manual Lab course Integrated Circuits and Project Design a Chip.* TU Delft.

[6] Github vhdl repository. `https://github.com/chanterheld/VGA_Module`. Accessed: 2017-12-7.

A SIMULATION RESULT

Clock counter

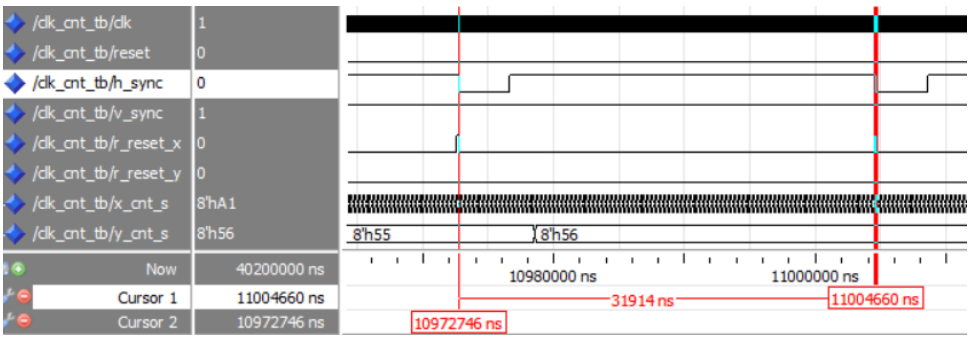

Figure 18: Test bench result showing the horizontal sync period

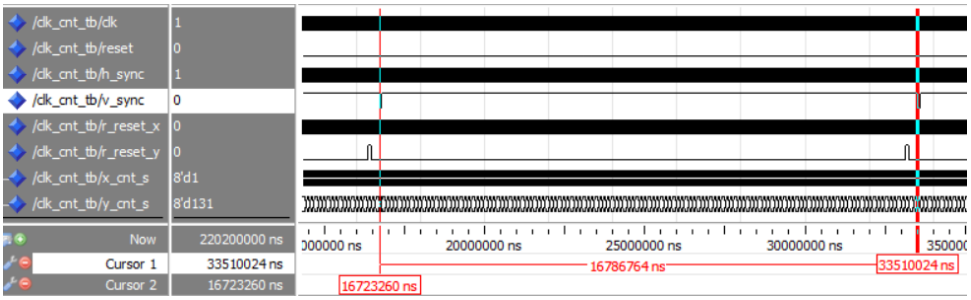

Figure 19: Test bench result showing the vertical sync period

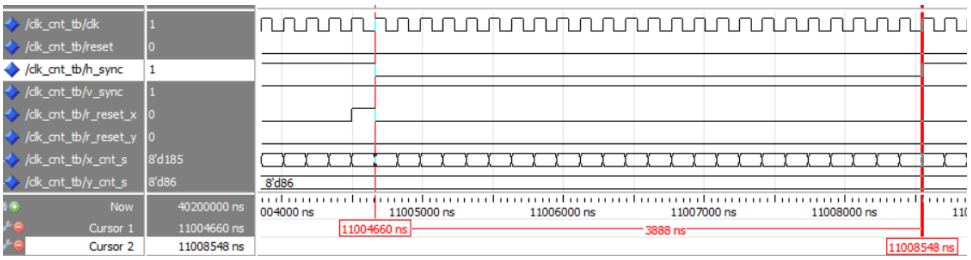

Figure 20: Test bench result showing the horizontal sync pulse width

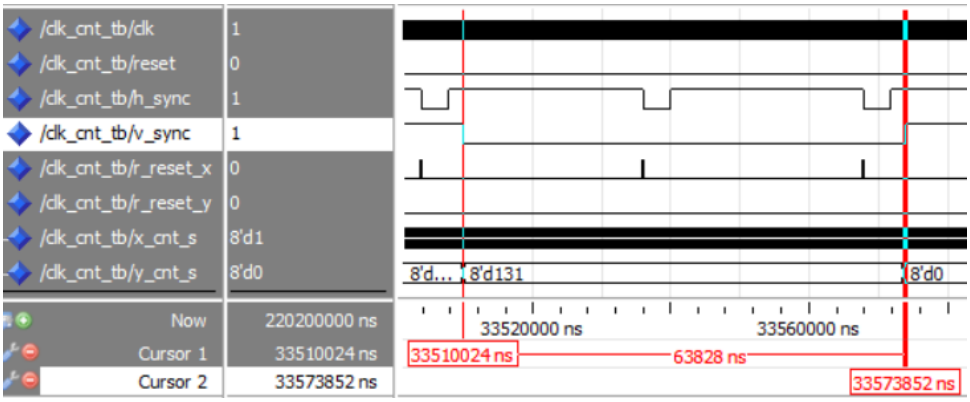

Figure 21: Test bench result showing the vertical sync pulse width

B OSCILLOSCOPE MEASUREMENTS

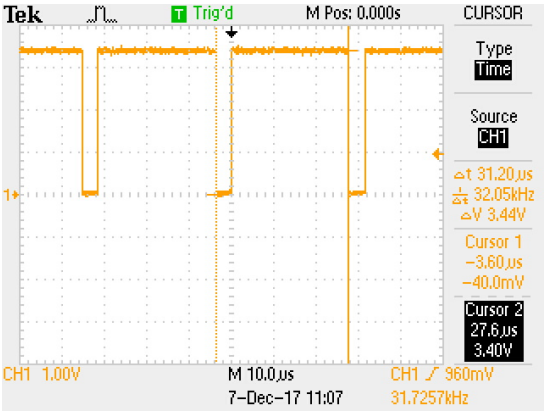

Figure 22: h sync signal period

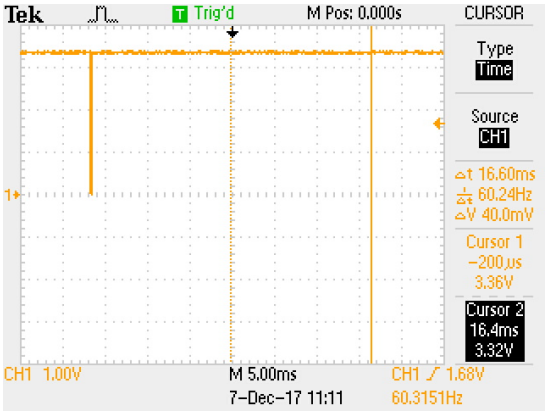

Figure 23: v sync signal period

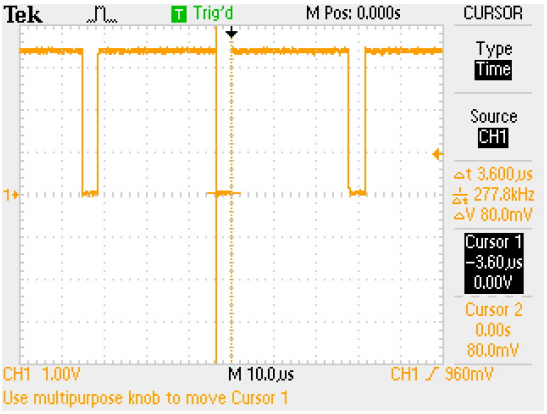

Figure 24: h sync pulse width

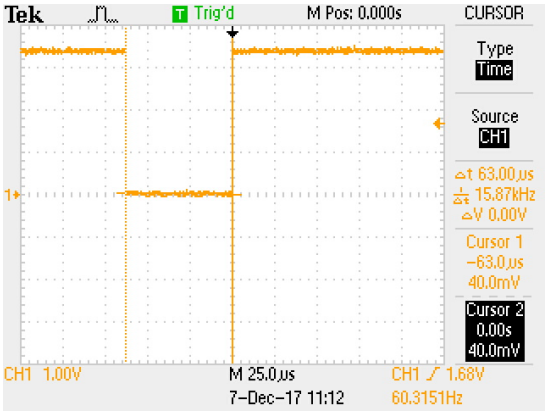

Figure 25: v sync pulse width

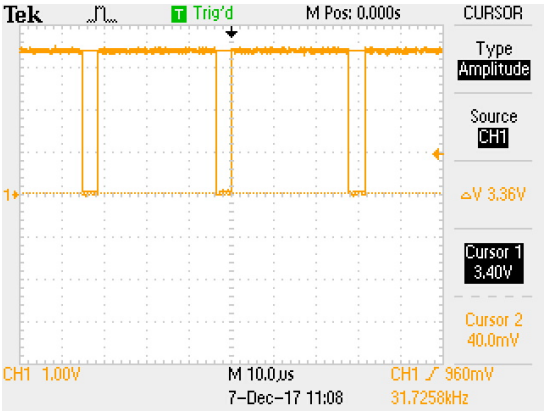

Figure 26: h sync signal amplitude

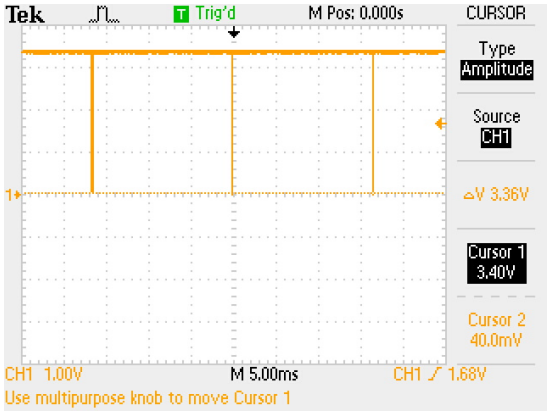

Figure 27: v sync signal amplitude

C ESPRESSO FILES AND RETURNS

BCD to seven segment

```
.i 4
.o 13
.ilb i3 i2 i1 i0
.ob o1 o2 o3 o4 o5 o6 o7 o8 o9 o10 o11 o12 o13
0000 1111110111111
0001 0010100101001
0010 1110111110111
0011 1110111101111
0100 1011111101001
0101 1111011101111
0110 1111011111111
0111 1110100101001
1000 1111111111111
1001 1111111101111
1010 -------------
1011 -------------
1100 -------------
1101 -------------
1110 -------------
1111 -------------
.e

o1 = (i3) | (i2) | (i1) | (!i0);

o2 = (!i2&!i0) | (i3) | (i1) | (i2&i0);

o3 = ();

o4 = (i2&!i0) | (i3) | (i2&!i1) | (!i1&!i0);

o5 = (i1&i0) | (!i2) | (!i1&!i0);

o6 = (i3) | (!i0) | (i2&!i1) | (!i2&i1);

o7 = (i1&!i0) | (i3) | (i2&!i1) | (!i2&i1);

o8 = ();

o9 = (!i2&!i0) | (i1&!i0);

o10 = (!i1) | (i2) | (i0);

o11 = (i1&!i0) | (i3) | (i2&!i1&i0) | (!i2&i1) | (!i2&!i0);

o12 = (i1&!i0) | (i3) | (i2&!i1&i0) | (!i2&i1) | (!i2&!i0);

o13 = ();
```

Dimension converter

```
.i 2
.o 5
.ilb i1 i0
```

```
.ob p4 p3 p2 p1 p0
00 11011
01 10110
10 10010
11 01111
.e
```

Segment width logic

```
.i 2
.o 5
.ilb i1 i0
.ob o4 o3 o2 o1 o0
00 00111
01 10011
10 00111
11 -----
.e
```

Segment height logic

```
.i 3
.o 5
.ilb i2 i1 i0
.ob o4 o3 o2 o1 o0
000 00111
001 01111
010 00111
011 01111
100 00111
101 01000
110 -----
111 -----
.e
```

D VHDL SOURCE FILES

Not all the low-level entities listed in the structural descriptions below have been added to this section of the appendix in an effort to limit file size. These and all other VHDL files can be found here[6]

D.1 Top level

```
library ieee;
use ieee.std_logic_1164.all;

entity vga_top_lvl is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                game_rst: in        std_logic;
                flag        : in        std_logic;
                score_up: in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                data_in        : in        std_logic_vector(1 downto 0);
```

```
                set_flag: out        std_logic;
                h_sync        : out        std_logic;
                v_sync        : out        std_logic;
                rgb        : out        std_logic_vector(2 downto 0);
                address        : out        std_logic_vector(5 downto 0)
        );
end entity;

 architecture structural of vga_top_lvl is
component dim_conv is
        port(        dip_sw                : in        std_logic_vector(1 downto 0);
                block_size        : out        std_logic_vector(4 downto 0)
        );
end component;

component tr_sync is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset_x: out        std_logic;
                r_reset_y: out        std_logic;
                cnt_x        : out        std_logic_vector(7 downto 0);
                cnt_y        : out        std_logic_vector(7 downto 0);
                h_sync        : out        std_logic;
                v_sync        : out        std_logic
        );
end component;


component xpos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0)
        );
end component;

component ypos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0);
                up        : out        std_logic
        );
end component;

component xpos_to_nr_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(1 downto 0)
        );
```

```
end component;

component ypos_to_nr_adr is
        port(        clk         : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(3 downto 0)
        );
end component;

component adr_to_colo_f is
        port(        clk         : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                y_up        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                x_adr        : in        std_logic_vector(2 downto 0);
                y_adr        : in        std_logic_vector(2 downto 0);
                data_in        : in        std_logic_vector(1 downto 0);

                set_flag: out        std_logic;
                e_n        : out        std_logic;
                address        : out        std_logic_vector(5 downto 0);
                color        : out        std_logic_vector(1 downto 0)
        );
end component;

component adr_to_color_sc is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                game_rst: in        std_logic;
                plus_one: in        std_logic;
                x_adr        : in        std_logic_vector(1 downto 0);
                y_adr        : in        std_logic_vector(3 downto 0);

                e_n        : out        std_logic;
                color        : out        std_logic_vector(1 downto 0)

        );
end component;

component rgb_decoder is
        port(        e1_n        : in        std_logic;
                e2_n        : in        std_logic;
                color_1        : in        std_logic_vector(1 downto 0);
                color_2        : in        std_logic_vector(1 downto 0);
                rgb        : out        std_logic_vector(2 downto 0)
        );
end component;

signal r_reset_x, r_reset_y, e_n_field1, e_n_field2, e_n_score, y_adr_up : std_logic;
signal nr_adr_x, color_field, color_score : std_logic_vector(1 downto 0);
signal blk_adr_x, blk_adr_y : std_logic_vector(2 downto 0);
signal nr_adr_y : std_logic_vector(3 downto 0);
signal block_size: std_logic_vector(4 downto 0);
signal x_posi, y_posi : std_logic_vector(7 downto 0);
```

```
begin
l1: tr_sync port map(clk, reset, r_reset_x, r_reset_y, x_posi, y_posi, h_sync, v_sync);
l5: xpos_to_blk_adr port map(clk, reset, r_reset_x, dip_sw, x_posi, block_size, blk_adr_x);
l6: ypos_to_blk_adr port map(clk, reset, r_reset_y, dip_sw, y_posi, block_size, blk_adr_y,
 ↪  y_adr_up);
l7: xpos_to_nr_adr port map(clk, reset, r_reset_x, x_posi, nr_adr_x);
l8: ypos_to_nr_adr port map(clk, reset, r_reset_y, y_posi, nr_adr_y);
l9: adr_to_colo_f port map(clk, reset, flag, y_adr_up, dip_sw, blk_adr_x, blk_adr_y, data_in,
 ↪  set_flag, e_n_field1, address, color_field);
l10: adr_to_color_sc port map(clk, reset, game_rst, score_up, nr_adr_x, nr_adr_y, e_n_score,
 ↪  color_score);
l11: rgb_decoder port map(e_n_field2, e_n_score, color_field, color_score, rgb);
l12: dim_conv port map(dip_sw, block_size);

e_n_field2 <= e_n_field1 or game_rst;
end structural;
```

D.2 Clock counter

```
library ieee;
use ieee.std_logic_1164.all;

entity tr_sync is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset_x: out        std_logic;
                r_reset_y: out        std_logic;
                cnt_x        : out        std_logic_vector(7 downto 0);
                cnt_y        : out        std_logic_vector(7 downto 0);
                h_sync        : out        std_logic;
                v_sync        : out        std_logic
        );
end entity;

 architecture structural of tr_sync is
component clk_cnt_8 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                cnt        : out        std_logic_vector(7 downto 0)
        );
end component;

component up_one_cnt_10 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out        std_logic_vector(9 downto 0)
        );
end component;

signal reset_s_x, intrm_1 : std_logic;
signal cnt_s_x : std_logic_vector(7 downto 0);

signal reset_s_y, intrm_2 : std_logic;
signal cnt_s_y : std_logic_vector(9 downto 0);

signal h_reg, h_next, intrm_h1, intrm_h2, intrm_h3, intrm_h4, intrm_h5 : std_logic;
```

```
signal v_reg, v_next, intrm_v1, intrm_v2, intrm_v3, intrm_v4: std_logic;

begin
--x_gen
l1: clk_cnt_8 port map(clk, reset_s_x, cnt_s_x);

r_reset_x <= '1' when (cnt_s_x = "10100000") else '0';
intrm_1 <= (cnt_s_x(7) and cnt_s_x(6) and cnt_s_x(2));
reset_s_x <= (intrm_1 or reset);
cnt_x <= cnt_s_x;

--y_gen
l2: up_one_cnt_10 port map(clk, reset_s_y, intrm_1, cnt_s_y);

r_reset_y <= '1' when (cnt_s_y(9 downto 2) = "10000000") else '0';
intrm_2 <= (cnt_s_y(9) and cnt_s_y(3) and cnt_s_y(2)and cnt_s_y(0) and intrm_1);
reset_s_y <= (intrm_2 or reset);
cnt_y <= cnt_s_y(9 downto 2);

--sync_gen
intrm_h1 <= (h_reg or reset);
intrm_h2 <= (cnt_s_x(7) nand cnt_s_x(5));
intrm_h3 <= (cnt_s_x(4) nand cnt_s_x(3));
intrm_h4 <= (intrm_h1 nand intrm_h2);
intrm_h5 <= (intrm_h3 or intrm_h2);
h_next <= (intrm_h4 nand intrm_h5);

intrm_v1 <= (v_reg or reset);
intrm_v2 <= (cnt_s_y(9) and cnt_s_y(3));
intrm_v3 <= (intrm_v2 nand cnt_s_y(2));
intrm_v4 <= (intrm_v1 nand intrm_v3);
v_next <= (intrm_v4 nand cnt_s_y(9));


process (clk)
begin
        if (rising_edge(clk)) then
                h_reg <= h_next;
                v_reg <= v_next;
        end if;
end process;
v_sync <= v_reg;
h_sync <= h_reg;
end structural;
```

D.3 Position to addresses

X-position to block column

```
library ieee;
use ieee.std_logic_1164.all;

entity xpos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
```

```
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0)
        );
end entity xpos_to_blk_adr;

 architecture structural of xpos_to_blk_adr is

component r_add_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                sum        : out        std_logic_vector(7 downto 0)
        );
end component;

component blk_cnt is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                cnt_rst        : out        std_logic;
                blk_nr        : out        std_logic_vector(2 downto 0)
        );
end component;

component gated_reg_8 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                load        : in        std_logic;
                t        : in        std_logic_vector(7 downto 0);
                q        : out        std_logic_vector(7 downto 0)
        );
end component;

component comp_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                equal        : out        std_logic
        );
end component;

signal pos_reg, pos_next, mplex_resized : std_logic_vector(7 downto 0);
signal mplex_out : std_logic_vector(4 downto 0);
signal comp_out, reg_nor, reg_load, reg_reset, cnt_reset, reset_s: std_logic;

begin
mplex_out <=        "00101" when (reg_nor = '1') else block_size; --mplex
l3: gated_reg_8        port map(clk, reg_reset, reg_load, pos_next, pos_reg);
l4: r_add_8        port map(pos_reg, mplex_resized, pos_next);
l5: comp_8        port map(pos_reg, posi, comp_out);
reg_nor <=        '1' when (pos_reg = "00000000") else '0'; --comp
l7: blk_cnt        port map(clk, reset_s, comp_out, dip_sw, cnt_reset, address);

mplex_resized <= "000"&mplex_out;
reg_load <= comp_out or reg_nor;
reg_reset <= reset_s or cnt_reset;
reset_s <= reset or r_reset;
end architecture;
```

Y-position to block row

```
library ieee;
use ieee.std_logic_1164.all;

entity ypos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0);
                up        : out        std_logic
        );
end entity ypos_to_blk_adr;

 architecture structural of ypos_to_blk_adr is

component r_add_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                sum        : out        std_logic_vector(7 downto 0)
        );
end component;

component blk_cnt is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                cnt_rst        : out        std_logic;
                blk_nr        : out        std_logic_vector(2 downto 0)
        );
end component;

component gated_reg_8 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                load        : in        std_logic;
                t        : in        std_logic_vector(7 downto 0);
                q        : out        std_logic_vector(7 downto 0)
        );
end component;

component comp_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                equal        : out        std_logic
        );
end component;

signal pos_reg, pos_next, mplex_resized : std_logic_vector(7 downto 0);
signal mplex_out : std_logic_vector(4 downto 0);
signal comp_out, reg_nor, reg_load, reg_reset, cnt_reset, reset_s: std_logic;

begin
mplex_out <=        "01110" when (reg_nor ='1') else block_size; --mplex
```

```
l3: gated_reg_8        port map(clk, reg_reset, reg_load, pos_next, pos_reg);
l4: r_add_8         port map(pos_reg, mplex_resized, pos_next);
l5: comp_8         port map(pos_reg, posi, comp_out);
reg_nor <=         '1' when (pos_reg = "00000000") else '0'; --comp
l7: blk_cnt         port map(clk, reset_s, comp_out, dip_sw, cnt_reset, address);

mplex_resized <= "000"&mplex_out;
reg_load <= comp_out or reg_nor;
reg_reset <= reset_s or cnt_reset;
reset_s <= reset or r_reset;
up <= comp_out;
end architecture;
```

X-position to score column

```
library ieee;
use ieee.std_logic_1164.all;

entity xpos_to_nr_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(1 downto 0)
        );
end entity xpos_to_nr_adr;

 architecture structural of xpos_to_nr_adr is
component r_add_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                sum        : out        std_logic_vector(7 downto 0)
        );
end component;

component nr_cnt_x is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                reg_rst        : out        std_logic;
                blk_nr        : out        std_logic_vector(1 downto 0)
        );
end component nr_cnt_x;

component gated_reg_8 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                load        : in        std_logic;
                t        : in        std_logic_vector(7 downto 0);
                q        : out        std_logic_vector(7 downto 0)
        );
end component;

component comp_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                equal        : out        std_logic
```

```
        );
end component;

signal pos_reg, pos_next, mplex_out, block_size : std_logic_vector(7 downto 0);
signal address_s: std_logic_vector(1 downto 0);
signal comp_out, reg_nor, reg_load, reg_reset, cnt_reset, reset_s: std_logic;

begin
mplex_out <=        "01111010" when (reg_nor = '1') else block_size;
l3: gated_reg_8        port map(clk, reg_reset, reg_load, pos_next, pos_reg);
l4: r_add_8         port map(pos_reg, mplex_out, pos_next);
l5: comp_8         port map(pos_reg, posi, comp_out);
reg_nor <=          '1' when (pos_reg = "00000000") else '0';
l7: nr_cnt_x        port map(clk, reset_s, comp_out, cnt_reset, address_s);

reg_load <= comp_out or reg_nor;
reg_reset <= reset_s or cnt_reset;
reset_s <= reset or r_reset;
address <= address_s;
block_size <= "000"&address_s(0)&'0'&not(address_s(0))&"11";
end architecture;
```

Y-position to score row

```
library ieee;
use ieee.std_logic_1164.all;

entity ypos_to_nr_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(3 downto 0)
        );
end entity ypos_to_nr_adr;

 architecture structural of ypos_to_nr_adr is
component r_add_8 is
        port(        a        : in        std_logic_vector(7 downto 0);
                b        : in        std_logic_vector(7 downto 0);
                sum        : out        std_logic_vector(7 downto 0)
        );
end component;

component mplex2t1_5 is
        port(        a        : in        std_logic_vector(4 downto 0);
                b        : in        std_logic_vector(4 downto 0);
                sel        : in        std_logic;
                q        : out        std_logic_vector(4 downto 0)
        );
end component;

component nr_cnt_y is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                reg_rst        : out        std_logic;
```

```
                blk_nr          : out         std_logic_vector(3 downto 0)
        );
end component;

component gated_reg_8 is
        port(           clk          : in          std_logic;
                reset          : in          std_logic;
                load           : in          std_logic;
                t          : in          std_logic_vector(7 downto 0);
                q          : out         std_logic_vector(7 downto 0)
        );
end component;

component comp_8 is
        port(           a          : in          std_logic_vector(7 downto 0);
                b          : in          std_logic_vector(7 downto 0);
                equal          : out         std_logic
        );
end component;

signal pos_reg, pos_next, mplex_resized  : std_logic_vector(7 downto 0);
signal mplex_out, block_size: std_logic_vector (4 downto 0);
signal address_s: std_logic_vector(3 downto 0);
signal comp_out, reg_nor, reg_load, reg_reset, cnt_reset, reset_s, intrm: std_logic;

begin
mplex_out <=         "01110" when (reg_nor = '1') else block_size; --mplex
l3: gated_reg_8         port map(clk, reg_reset, reg_load, pos_next, pos_reg);
l4: r_add_8         port map(pos_reg, mplex_resized, pos_next);
l5: comp_8         port map(pos_reg, posi, comp_out);
reg_nor <=         '1' when (pos_reg = "00000000") else '0'; --comp
l7: nr_cnt_y         port map(clk, reset_s, comp_out, cnt_reset, address_s);

mplex_resized <= "000"&mplex_out;
reg_load <= comp_out or reg_nor;
reg_reset <= reset_s or cnt_reset;
reset_s <= reset or r_reset;
address <= address_s;
intrm <= (address_s(0) nand address_s(2));
block_size <= '0'&address_s(0)&intrm&intrm&intrm;
end architecture;
```

Block address counter

```
library ieee;
use ieee.std_logic_1164.all;

entity blk_cnt is
        port(           clk          : in          std_logic;
                reset          : in          std_logic;
                up_one          : in          std_logic;
                dip_sw          : in          std_logic_vector(1 downto 0);
                cnt_rst          : out         std_logic;
                blk_nr          : out         std_logic_vector(2 downto 0)
        );
end entity blk_cnt;
```

```
 architecture structural of blk_cnt is
component up_one_cnt_3 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out        std_logic_vector(2 downto 0)
        );
end component;

signal reset_s, intrm_1, comp_out  : std_logic;
signal cnt_s, grid_size : std_logic_vector(2 downto 0);
begin
l1: up_one_cnt_3        port map(clk, reset_s, up_one, cnt_s);
comp_out <=                '1' when (cnt_s = '1'&dip_sw) else '0'; --comp

intrm_1 <= (comp_out and up_one);
reset_s <= (intrm_1 or reset);
blk_nr <= cnt_s;
cnt_rst <= intrm_1;
end structural;
```

Score column counter

```
library ieee;
use ieee.std_logic_1164.all;

entity nr_cnt_x is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                reg_rst        : out        std_logic;
                blk_nr        : out        std_logic_vector(1 downto 0)
        );
end entity nr_cnt_x;


 architecture structural of nr_cnt_x is
component up_one_cnt_2 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out        std_logic_vector(1 downto 0)
        );
end component;

signal reset_s, intrm_1, comp_out  : std_logic;
signal cnt_s : std_logic_vector(1 downto 0);
begin
l1: up_one_cnt_2        port map(clk, reset_s, up_one, cnt_s);
comp_out <=                '1' when (cnt_s = "11") else '0'; --comp

intrm_1 <= (comp_out and up_one);
reset_s <= (intrm_1 or reset);
blk_nr <= cnt_s;
reg_rst <= intrm_1;
end structural;
```

Score row counter

```
library ieee;
use ieee.std_logic_1164.all;

entity nr_cnt_y is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                up_one        : in        std_logic;
                reg_rst        : out        std_logic;
                blk_nr        : out        std_logic_vector(3 downto 0)
        );
end entity nr_cnt_y;

 architecture structural of nr_cnt_y is
component up_one_cnt_3 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out        std_logic_vector(2 downto 0)
        );
end component;

component t_ff is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                q        : out        std_logic
        );
end component;

signal reset_s, intrm_1, comp_out, nr_select  : std_logic;
signal cnt_s : std_logic_vector(2 downto 0);
begin
l1: up_one_cnt_3        port map(clk, reset_s, up_one, cnt_s);
comp_out <=                '1' when (cnt_s = "101") else '0'; --comp
l3: t_ff                port map(clk, reset, intrm_1, nr_select);

intrm_1 <= (comp_out and up_one);
reset_s <= (intrm_1 or reset);
blk_nr <= (nr_select&cnt_s);
reg_rst <= (intrm_1 and nr_select);
end structural;
```

Field dimensions to block size conversion

```
library ieee;
use ieee.std_logic_1164.all;

entity dim_conv is
        port(        dip_sw                : in        std_logic_vector(1 downto 0);
                block_size        : out        std_logic_vector(4 downto 0)
        );
end entity dim_conv;

 architecture behav of dim_conv is
signal intrm: std_logic;
begin
```

```
intrm <= (dip_sw(1) xnor dip_sw(0));
block_size(4) <= dip_sw(1) nand dip_sw(0);
block_size(3) <= intrm;
block_size(2) <= dip_sw(0);
block_size(1) <= '1';
block_size(0) <= intrm;
end behav;
```

D.4 Colour decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity rgb_decoder is
        port(        e1_n        : in        std_logic;
                e2_n        : in        std_logic;
                color_1        : in        std_logic_vector(1 downto 0);
                color_2        : in        std_logic_vector(1 downto 0);
                rgb        : out        std_logic_vector(2 downto 0)
        );
end rgb_decoder;

architecture behavior of rgb_decoder is
begin

process(e1_n, e2_n, color_1, color_2)
begin
        rgb <= "000";
        if(e1_n = '0') then
                rgb(2) <= color_1(1);
                rgb(1) <= color_1(0);
                rgb(0) <= color_1(1) nor color_1(0);
        elsif(e2_n='0') then
                rgb(2) <= color_2(1);
                rgb(1) <= color_2(0);
                rgb(0) <= color_2(1) nor color_2(0);
        end if;
end process;
end behavior;
```

D.5 Video memory & controller top level

```
library ieee;
use ieee.std_logic_1164.all;

entity adr_to_colo_f is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                y_up        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                x_adr        : in        std_logic_vector(2 downto 0);
                y_adr        : in        std_logic_vector(2 downto 0);
                data_in        : in        std_logic_vector(1 downto 0);

                set_flag: out        std_logic;
                e_n        : out        std_logic;
```

```
                address         : out         std_logic_vector(5 downto 0);
                color         : out         std_logic_vector(1 downto 0)
        );
end adr_to_colo_f;




architecture structural of adr_to_colo_f is
component color_sel is
        port(         clk         : in         std_logic;
                reset         : in         std_logic;
                x_adr         : in         std_logic_vector(2 downto 0);
                wr_adr         : in         std_logic_vector(2 downto 0);
                wr_bus         : in         std_logic_vector(1 downto 0);
                wr_e         : in         std_logic;

                clr_out         : out         std_logic_vector(1 downto 0)
        );
end component;

component vga_reg_upd is
        port(         clk         : in         std_logic;
                reset         : in         std_logic;
                flag         : in         std_logic;
                dip_sw         : in         std_logic_vector(1 downto 0);
                y_adr         : in         std_logic_vector(2 downto 0);
                y_up         : in         std_logic;

                set_flag: out         std_logic;
                write_en: out         std_logic;
                reg_x         : out         std_logic_vector(2 downto 0);
                reg_y         : out         std_logic_vector(2 downto 0)
        );
end component;

component comp_3 is
        port(         a         : in         std_logic_vector(2 downto 0);
                b         : in         std_logic_vector(2 downto 0);
                equal         : out         std_logic
        );
end component;

signal wr_e, x_nor, y_nor: std_logic;
signal reg_x, reg_y, wr_adr : std_logic_vector(2 downto 0);
begin
l1: color_sel port map(clk, reset, x_adr, wr_adr, data_in, wr_e, color);
l2: vga_reg_upd port map(clk, reset, flag, dip_sw, y_adr,  y_up, set_flag, wr_e, reg_x, reg_y);
x_nor <= '1' when (x_adr = "000") else '0'; --comp
y_nor <= '1' when (y_adr = "000") else '0'; --comp

e_n <= x_nor or y_nor;
wr_adr <= reg_x;
address <= reg_x&reg_y;
end structural;
```

D.6 Video memory

temporary row storage

```
library ieee;
use ieee.std_logic_1164.all;

entity color_sel is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                x_adr        : in        std_logic_vector(2 downto 0);
                wr_adr        : in        std_logic_vector(2 downto 0);
                wr_bus        : in        std_logic_vector(1 downto 0);
                wr_e        : in        std_logic;

                clr_out        : out        std_logic_vector(1 downto 0)
        );
end color_sel;



architecture structural of color_sel is
component c_reg is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                re        : in        std_logic_vector(6 downto 0);
                we        : in        std_logic_vector(6 downto 0);
                write_bus: in        std_logic_vector(1 downto 0);
                read_bus: out        std_logic_vector(1 downto 0)
        );
end component;

component decoder_3to7_e is
        port(        enable        : in        std_logic;
                x        : in        std_logic_vector(2 downto 0);
                y        : out        std_logic_vector(6 downto 0)
        );
end component;

component decoder_3to8 is
  port(
    x  :  in  std_logic_vector(2 downto 0);
    y  :  out  std_logic_vector(7 downto 0)
        );
end component;

signal read_e: std_logic_vector(7 downto 0);
signal write_e: std_logic_vector(6 downto 0);

begin
l1: c_reg port map(clk, reset, read_e(7 downto 1), write_e, wr_bus, clr_out);
l2: decoder_3to7_e port map(wr_e, wr_adr, write_e);
l3: decoder_3to8 port map(x_adr, read_e);


end structural;
```

Register bank

```
library ieee;
use ieee.std_logic_1164.all;

entity c_reg is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                re        : in        std_logic_vector(6 downto 0);
                we        : in        std_logic_vector(6 downto 0);
                write_bus: in        std_logic_vector(1 downto 0);
                read_bus: out        std_logic_vector(1 downto 0)
        );
end c_reg;

architecture structural of c_reg is
component gated_reg_2 is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                load        : in        std_logic;
                t        : in        std_logic_vector(1 downto 0);
                q        : out        std_logic_vector(1 downto 0)
        );
end component;

component tr_buf_2 is
        port(       t        : in        std_logic_vector(1 downto 0);
                en        : in        std_logic;
                q        : out        std_logic_vector(1 downto 0)
        );
end component;

signal ff_out: std_logic_vector(13 downto 0);
begin
reg_gen:
for i in 0 to 6 generate
        ll_1: gated_reg_2 port map(clk, reset, we(i), write_bus, ff_out(2*i+1 downto 2*i));
        ll_2: tr_buf_2 port map(ff_out(2*i+1 downto 2*i), re(i),read_bus);
end generate ;
end structural;
```

D.7 Video memory controller

Video memory controller top level

```
library ieee;
use ieee.std_logic_1164.all;

entity vga_reg_upd is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                y_adr        : in        std_logic_vector(2 downto 0);
                y_up        : in        std_logic;

                set_flag: out        std_logic;
                write_en: out        std_logic;
```

```
                reg_x        : out        std_logic_vector(2 downto 0);
                reg_y        : out        std_logic_vector(2 downto 0)
        );
end entity vga_reg_upd;


architecture structural of vga_reg_upd is
component vga_reg_upd_fsm is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                lst_blk : in        std_logic;
                adr_up        : in        std_logic;

                plex_sel: out        std_logic;
                set_flag: out        std_logic;
                write_en: out        std_logic;
                reg_l        : out        std_logic
        );
end component;

component one_adder_3 is
        port(        a        : in        std_logic_vector(2 downto 0);
                sum        : out        std_logic_vector(2 downto 0)
        );
end component;

component mplex2t1_3 is
        port(        a        : in        std_logic_vector(2 downto 0);
                b        : in        std_logic_vector(2 downto 0);
                sel        : in        std_logic;
                q        : out        std_logic_vector(2 downto 0)
        );
end component;

component gated_reg_3 is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                load        : in        std_logic;
                t        : in        std_logic_vector(2 downto 0);
                q        : out        std_logic_vector(2 downto 0)
        );
end component;

signal reg_s, adder_in, adder_out, nxt_row : std_logic_vector(2 downto 0);
signal lb5, lst_blk, reg_r, reg_l: std_logic;
begin
l1: mplex2t1_3 port map(y_adr, reg_s, lb5, adder_in);
l4: mplex2t1_3 port map(nxt_row, y_adr, lb5, reg_y);
l2: one_adder_3 port map(adder_in, adder_out);
nxt_row <= "001" when (lst_blk = '1') else adder_out;
lst_blk <= '1' when (adder_in = '1'&dip_sw) else '0';
l7: gated_reg_3 port map(clk, reset, reg_l, nxt_row, reg_s);
l10: vga_reg_upd_fsm port map(clk, reset, flag, lst_blk, y_up, lb5, set_flag, write_en, reg_l);

reg_x <= reg_s;
end architecture;
```

Temporary row storage update FSM

```
library ieee;
use ieee.std_logic_1164.all;

entity vga_reg_upd_fsm is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                lst_blk : in        std_logic;
                adr_up        : in        std_logic;

                plex_sel: out        std_logic;
                set_flag: out        std_logic;
                write_en: out        std_logic;
                reg_l        : out        std_logic
        );
end entity vga_reg_upd_fsm;

architecture behav of vga_reg_upd_fsm is
type fsm_state is (idle, load_set, flag_wait);
signal state, next_state : fsm_state;
begin
reg: process (clk)
begin
        if (rising_edge(clk)) then
                if (reset ='1') then
                        state <= idle;
                else
                        state <= next_state;
                end if;
        end if;
end process;

comb: process(state, flag, lst_blk, adr_up)
begin
next_state <= state;
case state is
        when idle =>
                plex_sel <= '0';
                set_flag <= '0';
                reg_l <= '0';
                write_en <= '0';
                if ((adr_up = '1') and (lst_blk = '0')) then
                        next_state <= load_set;
                        write_en <= '1';
                end if;

        when load_set =>
                plex_sel <= '1';
                set_flag <= '1';
                reg_l <= '1';
                write_en <= '0';
                if (lst_blk = '1') then
                        next_state <= idle;
                else
                        next_state <= flag_wait;
                end if;
```

```
        when flag_wait =>
                plex_sel <= '1';
                set_flag <= '0';
                reg_l <= '0';
                write_en <= '1';
                if (flag = '0') then
                        next_state <= load_set;
                end if;

end case;
end process;
end behav;
```

D.8 Score counter

Score counter top level

```
library ieee;
use ieee.std_logic_1164.all;

entity adr_to_color_sc is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                game_rst: in        std_logic;
                plus_one: in        std_logic;
                x_adr        : in        std_logic_vector(1 downto 0);
                y_adr        : in        std_logic_vector(3 downto 0);

                e_n        : out        std_logic;
                color        : out        std_logic_vector(1 downto 0)

        );
end entity adr_to_color_sc;


architecture structural of adr_to_color_sc is
component score_cnt is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                game_rst: in        std_logic;
                plus_one: in        std_logic;
                tens        : out        std_logic_vector(1 downto 0);
                ones        : out        std_logic_vector(3 downto 0)
        );
end component;

component mplex2t1_4 is
        port(       a        : in        std_logic_vector(3 downto 0);
                b        : in        std_logic_vector(3 downto 0);
                sel        : in        std_logic;
                q        : out        std_logic_vector(3 downto 0)
        );
end component;

component bcd_to_seg is
        port(       bcd        : in        std_logic_vector(3 downto 0);
```

```
                seg         : out         std_logic_vector(12 downto 0)
        );
end component;

component num_decoder is
        port(         x_adr         : in         std_logic_vector(1 downto 0);
                y_adr         : in         std_logic_vector(2 downto 0);
                seg         : in         std_logic_vector(12 downto 0);
                e_n         : out         std_logic
        );
end component;

signal tens: std_logic_vector(1 downto 0);
signal tens_resized, ones, bcd: std_logic_vector(3 downto 0);
signal seg: std_logic_vector(12 downto 0);
begin
l1: score_cnt port map(clk, reset, game_rst, plus_one, tens, ones);
l2: mplex2t1_4 port map(tens_resized, ones, y_adr(3), bcd);
l3: bcd_to_seg port map(bcd, seg);
l4: num_decoder port map(x_adr, y_adr(2 downto 0), seg, e_n);

tens_resized <= "00"&tens;
color <= ones(2 downto 1);
end structural;
```

Score address to segment converter

```
library ieee;
use ieee.std_logic_1164.all;

entity num_decoder is
        port(         x_adr         : in         std_logic_vector(1 downto 0);
                y_adr         : in         std_logic_vector(2 downto 0);
                seg         : in         std_logic_vector(12 downto 0);
                e_n         : out         std_logic
        );
end entity num_decoder;

architecture behav of num_decoder is
signal mux_in : std_logic_vector(4 downto 0);
begin
mux: process(mux_in, seg)
begin
        case mux_in is
                when "01001" => e_n <= seg(0);
                when "10001" => e_n <= seg(1);
                when "11001" => e_n <= seg(2);
                when "01010" => e_n <= seg(3);
                when "11010" => e_n <= seg(4);
                when "01011" => e_n <= seg(5);
                when "10011" => e_n <= seg(6);
                when "11011" => e_n <= seg(7);
                when "01100" => e_n <= seg(8);
                when "11100" => e_n <= seg(9);
                when "01101" => e_n <= seg(10);
                when "10101" => e_n <= seg(11);
                when "11101" => e_n <= seg(12);
```

```
                when others => e_n <= '1';
        end case;
end process;

mux_in <= x_adr&y_adr;
end behav;
```

Score counter

```
library ieee;
use ieee.std_logic_1164.all;

entity score_cnt is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                game_rst: in        std_logic;
                plus_one: in        std_logic;
                tens        : out       std_logic_vector(1 downto 0);
                ones        : out       std_logic_vector(3 downto 0)
        );
end entity score_cnt;

architecture structural of score_cnt is
component up_one_cnt_2 is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out       std_logic_vector(1 downto 0)
        );
end component;

component up_one_cnt_4 is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                t        : in        std_logic;
                cnt        : out       std_logic_vector(3 downto 0)
        );
end component;

signal ones_r, tens_r, ones_max: std_logic;
signal ones_s: std_logic_vector(3 downto 0);
begin
l1: up_one_cnt_2 port map(clk, tens_r, ones_max, tens);
l2: up_one_cnt_4 port map(clk, ones_r, plus_one, ones_s);

ones_max <= ones_s(3) and ones_s(0) and plus_one;
tens_r <= reset or game_rst;
ones_r <= ones_max or tens_r;
ones <= ones_s;
end structural;
```

BCD to 7-segment converter

```
library ieee;
use ieee.std_logic_1164.all;

entity bcd_to_seg is
```

```
        port(        bcd        : in        std_logic_vector(3 downto 0);
                seg        : out        std_logic_vector(12 downto 0)
        );
end bcd_to_seg;

architecture behavior of bcd_to_seg is
signal seg_s : std_logic_vector(12 downto 0);
begin
seg_s(0) <= seg_s(1) and seg_s(3);
seg_s(1) <= not((bcd(2) xnor bcd(0)) or bcd(3) or bcd(1));
seg_s(2) <= '0';
seg_s(3) <= not(((bcd(1) nand bcd(0)) and bcd(2)) or bcd(3) or (bcd(1) nor bcd(0)));
seg_s(4) <= (bcd(1) xor bcd(0)) and bcd(2);
seg_s(5) <= seg_s(3) and seg_s(6) and seg_s(8); --seg(3) or seg(6) or seg(8)
seg_s(6) <= not((bcd(2) xor bcd(1)) or bcd(3) or (not(bcd(1)) nor bcd(0)));
seg_s(7) <= '0';
seg_s(8) <= not((bcd(2) nor bcd(0)) or (not(bcd(1)) nor bcd(0)));
seg_s(9) <= not(bcd(2) or not(bcd(1)) or bcd(0));
seg_s(10) <= not(bcd(3) or (not(bcd(1)) nor bcd(0)) or (bcd(2) xor (bcd(0) nand not(bcd(1)))));
seg_s(11) <= seg_s(10);
seg_s(12) <= '0';
seg <= seg_s;
end behavior;
```

E VHDL TESTBENCHES

BCD to segment converter

```
--100 ns
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity bcd_to_seg_tb  is
end entity bcd_to_seg_tb;

architecture structural of bcd_to_seg_tb is

component bcd_to_seg is
        port(        bcd        : in        std_logic_vector(3 downto 0);
                seg        : out        std_logic_vector(12 downto 0)
        );
end component bcd_to_seg;
signal bcd : std_logic_vector(3 downto 0);
signal seg : std_logic_vector (12 downto 0);

begin

  bcd <=         "0000" after 0 ns,
                "0001" after 10 ns,
                "0010" after 20 ns,
                "0011" after 30 ns,
                "0100" after 40 ns,
                "0101" after 50 ns,
                "0110" after 60 ns,
                "0111" after 70 ns,
```

```
                "1000" after 80 ns,
                "1001" after 90 ns;

L01: bcd_to_seg port map (bcd, seg);
end architecture structural;
```

Position to address converter

```
--60 us
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity pos_to_adr_tb is
end entity;

architecture behav of pos_to_adr_tb is
component xpos_to_nr_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(1 downto 0)
        );
end component;

component xpos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0)
        );
end component;

component ypos_to_blk_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                posi        : in        std_logic_vector(7 downto 0);
                block_size        : in        std_logic_vector(4 downto 0);
                address        : out        std_logic_vector(2 downto 0);
                up        : out        std_logic
        );
end component;

component ypos_to_nr_adr is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset        : in        std_logic;
                posi        : in        std_logic_vector(7 downto 0);
                address        : out        std_logic_vector(3 downto 0)
        );
end component;
```

```
component dim_conv is
        port(        dip_sw                : in        std_logic_vector(1 downto 0);
                block_size        : out        std_logic_vector(4 downto 0)
        );
end component;

signal posi  : std_logic_vector(7 downto 0);
signal pos_gen, clk,reset, y_up : std_logic;
signal dip_sw, xnr_adr: std_logic_vector(1 downto 0);
signal xblk_adr, yblk_adr: std_logic_vector(2 downto 0);
signal ynr_adr: std_logic_vector(3 downto 0);
signal block_size: std_logic_vector(4 downto 0);
begin

clk         <=        '0' after 0 ns,
                '1' after 10 ns when clk /= '1' else '0' after 10 ns;

pos_gen <=         '0' after 0 ns,
                '1' after 40 ns when pos_gen /= '1' else '0' after 40 ns;

reset        <=         '1' after 0 ns,
               '0' after 50 ns;

dip_sw <=         "00" after 0 ns,
               "01" after 15000 ns,
               "10" after 30000 ns,
               "11" after 45000 ns;

L1: xpos_to_blk_adr port map(clk, reset, reset, dip_sw, posi, block_size, xblk_adr);
L2: ypos_to_blk_adr port map(clk, reset, reset, dip_sw, posi, block_size, yblk_adr, y_up);
L3: xpos_to_nr_adr port map(clk, reset, reset, posi, xnr_adr);
L4: ypos_to_nr_adr port map(clk, reset, reset, posi, ynr_adr);
L5: dim_conv port map(dip_sw, block_size);


cnt: process (pos_gen)
variable tmp : integer range 0 to 200 := 0;
begin
        if (rising_edge(pos_gen)) then
                if tmp = 200 then
                        tmp := 0;
                else
                        tmp := tmp + 1;
                end if;
        end if;

posi <= std_logic_vector(to_unsigned(tmp,8));
end process;

end behav;
```

Video memory update

```
-- 70 ms
library ieee;
use ieee.std_logic_1164.all;
```

```
entity vga_reg_upd_tb is
end entity;

architecture behav of vga_reg_upd_tb is

component vga_reg_upd is
        port(       clk        : in        std_logic;
                reset        : in        std_logic;
                flag        : in        std_logic;
                dip_sw        : in        std_logic_vector(1 downto 0);
                y_adr        : in        std_logic_vector(2 downto 0);
                val_l        : in        std_logic_vector(2 downto 0);
                val_r        : in        std_logic_vector(2 downto 0);

                load_val: out        std_logic;
                hold_val: out        std_logic;
                set_flag: out        std_logic;
                reg_x        : out        std_logic_vector(2 downto 0);
                reg_y        : out        std_logic_vector(2 downto 0)

        );
end component;

signal clk, reset, flag, flag_next, clr_flag, load_val, hold_val, set_flag : std_logic;
signal y_adr, val_l, val_r, reg_x, reg_y : std_logic_vector(2 downto 0);
signal dip_sw : std_logic_vector(1 downto 0);

begin
l1: vga_reg_upd port map(clk, reset, flag, dip_sw, y_adr, val_l, val_r, load_val, hold_val,
 ↪  set_flag, reg_x, reg_y);

clk         <=        '0' after 0 ns,
                '1' after 10 ns when clk /= '1' else '0' after 10 ns;

reset        <=         '1' after 0 ns,
                '0' after 50 ns;

dip_sw        <=         "11" after 0 ns;

y_adr        <=         "010" after 0 ns,
                "011" after 100 ns,
                "111" after 1100 ns;


val_l        <=         "011" after 0 ns;

val_r        <=        "001" after 0 ns,
                "000" after 1700 ns;

process (clk, set_flag, clr_flag, flag)
begin
        if flag = '1' then
                flag_next <= '1';
        else
                flag_next <= '0';
        end if;
```

```
        if set_flag = '1' then
                flag_next <= '1';
        elsif clr_flag = '1' then
                flag_next <= '0';
        end if;
        if (rising_edge(clk)) then
                flag <= flag_next;
        end if;

end process;

clr_flag <= flag after 60 ns;
end behav;
```

Clock counter

```
-- 40 ms
library IEEE;
use IEEE.std_logic_1164.all;

entity clk_cnt_tb is
end entity clk_cnt_tb;

architecture behav of clk_cnt_tb is
component tr_sync is
        port(        clk        : in        std_logic;
                reset        : in        std_logic;
                r_reset_x: out        std_logic;
                r_reset_y: out        std_logic;
                cnt_x        : out        std_logic_vector(7 downto 0);
                cnt_y        : out        std_logic_vector(7 downto 0);
                h_sync        : out        std_logic;
                v_sync        : out        std_logic
        );
end component;

signal clk, reset,  h_sync, v_sync,r_reset_x ,r_reset_y: std_logic;
signal x_cnt_s, y_cnt_s: std_logic_vector(7 downto 0);

begin
clk         <=         '1' after 0 ns,
                '0' after 81380 ps when clk /= '0' else '1' after 81380 ps; --6144015 Hz +-
                 ↪  6.144MHz
reset <=         '1' after 0 ns,
                '0' after 200 ns;

l5: tr_sync        port map(clk, reset, r_reset_x, r_reset_y, x_cnt_s, y_cnt_s, h_sync,
 ↪  v_sync);
end architecture;
```