

Wilbur Lewis-s3529819, Manar Yalid-3496165



School of Electrical and Computer Engineering

EEET2162 Advanced Digital Design 1

Major Project - Technical Report

HDMI processing and overlay

(on-screen display) using the Cyclone V

Lecturer: Dr Glenn Matthews

Tutor: Christopher Harrison

Students:

Manar Yalid (s3496165)

Wilbur Lewis (s3529819)

Submission Due Date:

05/06/2019

Wilbur Lewis-s3529819, Manar Yalid-3496165

Table of content: -

Introduction	3
Discussion	3
Methodology	6
Conclusion.....	9
References.....	10
Appendixes.....	11

Introduction:

The main aim of this project is to develop a Verilog HDL design that utilises the HDMI transmitter to generate an on-screen display. The design that was developed uses a static image that is saved in a ROM block and by deploying the I2C protocol and the 24-bit data interface for the HDMI transmitter on the FPGA to output the static image on display. The DE10 Nano development board was employed in the design and all Verilog HDL code, and pins assignment were implemented and tested using the Quartus toolchain [1].

The project consists of two main parts that need to be configured in order to output the image on screen display. First part is configuring the I2C, which consist of two signals serial clock known as SCL and serial data known as SDA. The second part is getting video data by utilising the 24-bit data interface on the HDMI chip [2].

Discussion:

Part one: This part only explains the configuration of the I2C, which is used to write a list of commands to drive out the HDMI transmitter to wake up.

I2C is an inter-integrated circuit protocol that is used for connecting IC's on the same circuit board. It was developed by Philips back 1982, it is a two-wire interface and works on lower clock speeds compared to other protocols such as the SPI. The communication of the I2C is done in between the master and slave. I2C has two lines the serial clock (SCL) and serial data (SDA), SCL is always driven by the master as an output, and the SDA can be driven by either master or slave, therefore, its bidirectional(Figure 1) so the idea after the i2c is a protocol that has two lines with one device being the master and multiple slave devices [3].

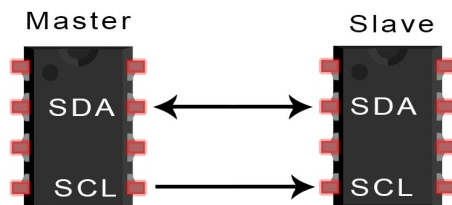


Figure 1 I2C master-slave[4]

Figure 2 demonstrates the communication with 7-bit I2C address which has a start condition and a stop condition then the next part is when the data are being sent to begin with an 8 bits transaction, where the address is 7 bits followed with one read-write bit therefore during interactions the data can be read or written into it.

Wilbur Lewis-s3529819, Manar Yalid-3496165

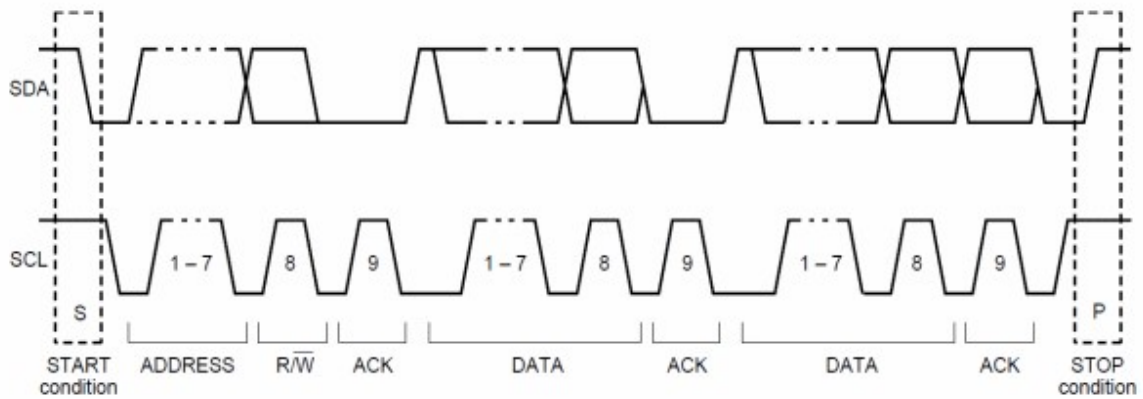


Figure 2 communication with I2C(7-bit) [5]

In this project, the data are being written to the I2C no matter what is inside, and this is fixed information in the design. After the data are sent the bidirectional data bus come, to play in which the master goes back to high and if something is listening to it will pull the bus low, and this will drive either the ACK (acknowledged) or NCK (not acknowledged). This type of interaction occurs for every bit [6].

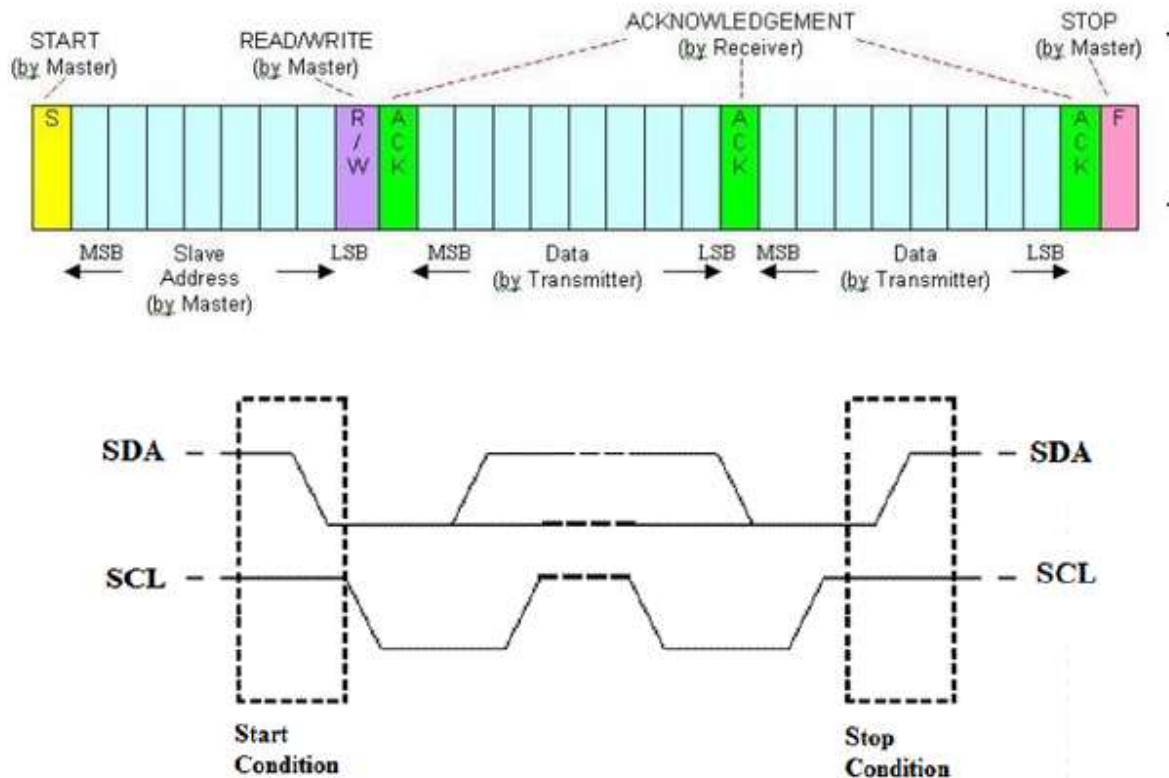


Figure 3 I2C communication protocol [7]

Wilbur Lewis-s3529819, Manar Yalid-3496165

Therefore for every time, the interaction finishes the slave talks back and replies with an acknowledge, then continuing with sending the data. Figure 3 demonstrates the I2C protocol the first two bytes represents the address register that will be writing in, the last byte represents the value that will be written to it.

A list of commands that contains the address and the value will be written using the I2C in order to configure the HDMI transmitter. The hardware user guide revision B was used to complete the setup of the HDMI chip, a ROM was created using Quartus that contain all the commands with address and their values, and all was needed is to iterate through the ROM by sending one by one and at the end the program stops and goes to sleep forever. After the chip was on the next part can be started. A state machine was created that starts with an ideal state then iterating through and sending the next address. The process was preloading the address and values into the register [8].

Part two: video/image configuration will be explained in this part

Resolution of $640 * 480$ will be used, and by having a frame that contains the pixels. All need to be done is to go through each pixel until the whole frame is finished. Each pixel is R or G or B (red, green, blue). The image frame will be saved in a ROM file then outputted into the video generator.

On the frame, the program starts from the zero pixel and for every clock edge output the next pixel something similar to a large shift register. Therefore, advancing through frame must output the next pixel. Thus, starting from pixel 0 and clocking until pixel 639 is reached that is the end of the line.

Next the program goes to the next line then all the frame pixel are being outputted as a constant stream of data, that is an issue as the program does not have a way to tell when the end of the first line and does not distinguish which one pixel 0 or pixel 639, so literally what is being generated is constant set of pixels that has information such as being R or G or B but no way to tell which pixel is the first or last pixel of that raw, and this is where the control signals come in, e.g. the output enabled signals (vertical sync digital signal (H-sync) and horizontal sync digital signal (V-sync)).

Output enabled theory is if the output is enabled, then the pixel data is on and if the output is not enabled, then the pixel data is off.

H-sync and V-sync are some sort of virtual space where no image data are being displayed, and this is referred to as the blanking time where essentially the output is black 0 in this case the output enabled is off, and no pixel data are shown. the H-sync is the blank space located after pixel 639 horizontally figure 4, whereas V-sync is the blank space located after pixel 479 vertically figure 4[9]

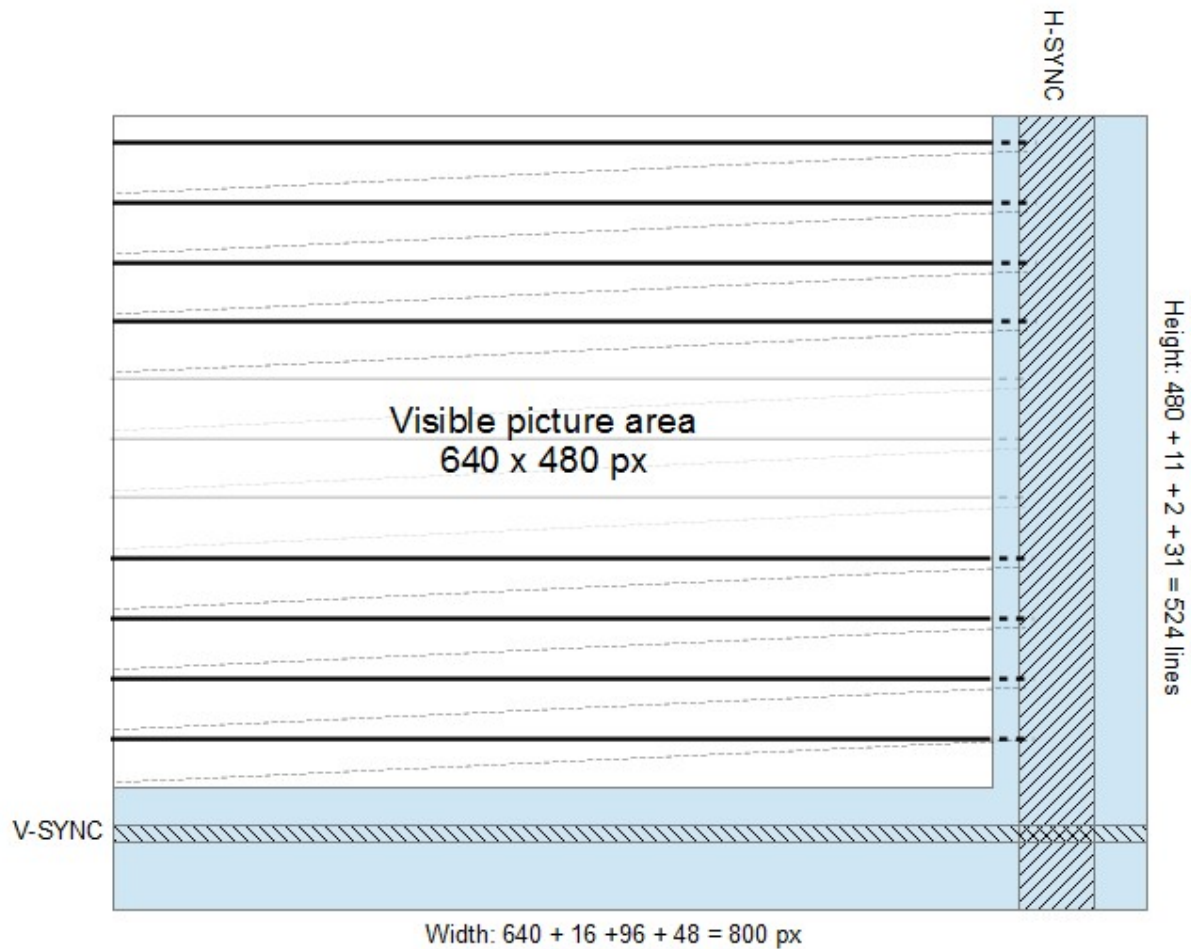


Figure 4 Control signals[10]

Methodology:

I2C

400kHz clock

To make the I2C communication to the HDMI module, we must use a PLL as well as a counter to get a clock frequency of 400kHz. This 400kHz will run our state machine, which will transmit I2C data at a rate of 100kHz.

The PLL we have used gives a frequency of 8MHz. This frequency was chosen as it is easy to reduce to 400kHz (5MHz doesn't divide perfectly). We used a variable counter made for the previous lab.

$8\text{MHz}/400\text{kHz} = 20$. Therefore, one period of 400kHz should equal 20 periods of the 8MHz clock. Using this knowledge, a state machine is made which sets the state of the 400kHz clock, as shown in figure 5.

Wilbur Lewis-s3529819, Manar Yalid-3496165

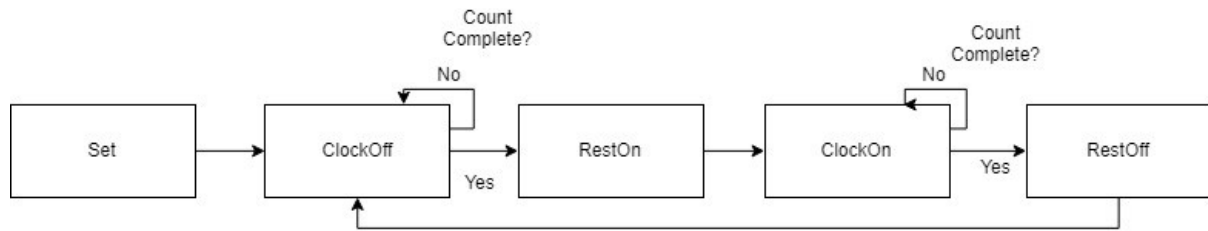


Figure 5 State machine

For this state machine, it is evaluated at the rising edge of the 8MHz clock. If not specified, it means the positive edge causes the state machine to proceed to the next state specified by the arrow. RestOn and ClockOn are when the 400kHz clock is set to high and RestOff, and ClockOff is when the clock is set to low. RestOn and RestOff are used to reset the counter for the states ClockOn and ClockOff. A counter with a duration of 6 counts is used to achieve a clock of 400kHz. This was chosen as half the period of the 400kHz clock is ten clock periods of the 8MHz clock. There is also the time needed to change between these states of resetting the timer and hence why a count of 6 is chosen. This was simulated and found to be correctly made.

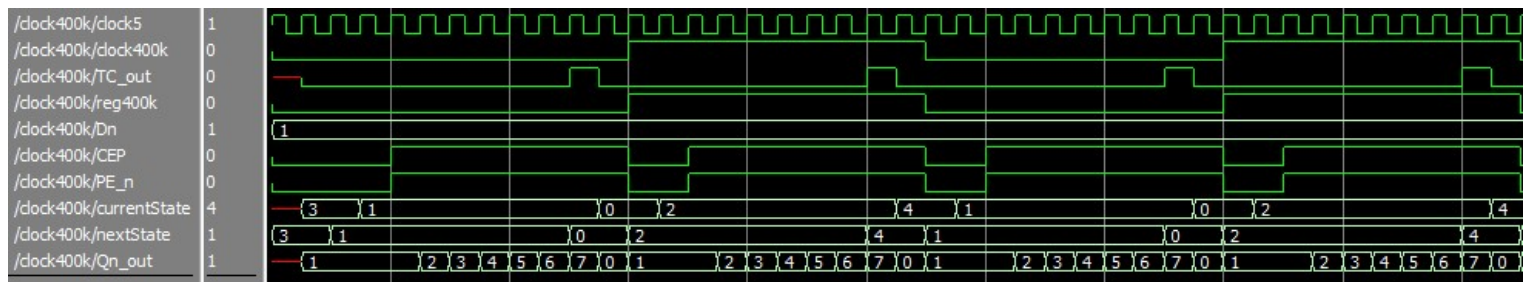


Figure 6 Module sim (400 KHz clock)

As shown in figure 6 there are ten clock periods in half a period of the 400kHz clock.

ROM and Increment

Each time data is written to the HDMI module it will first send the slave address (0x72) as mentioned, and then the control register address and then the data to be written to the control register. Therefore, ROM is made with a data size of 16 bit, which will hold the control register address and the data to be written to the address. We have made an increment module which increments to the next address in the ROM table, allowing us to select the next control register and slave address to be sent when needed. This was correctly made and simulated as shown in figure 7



Figure 7 module sim- Increment module

0x9083 is the first data byte of information in the ROM at address 0x00. The increment on a positive edge increments the address to be accessed. The data at address 0x01 is 0100.

Wilbur Lewis-s3529819, Manar Yalid-3496165

Shift register

Also, what is needed is a shift register. Since one bit is sent at a time in our state machine and the most significant bit is sent first a shift register has been made which will shift a one bit to the left. Then the most significant bit is only sent. This is demonstrated in table 1, with the most significant bit highlighted in blue.

1	0	0	0	1	1	0	0
0	0	0	1	1	0	0	0

Table 1 shift register

As shown the byte was shifted one bit to the left.

This shift register also has a 'shiftComplete' output which tells us when all the bits have been shifted out.

I2C state machine

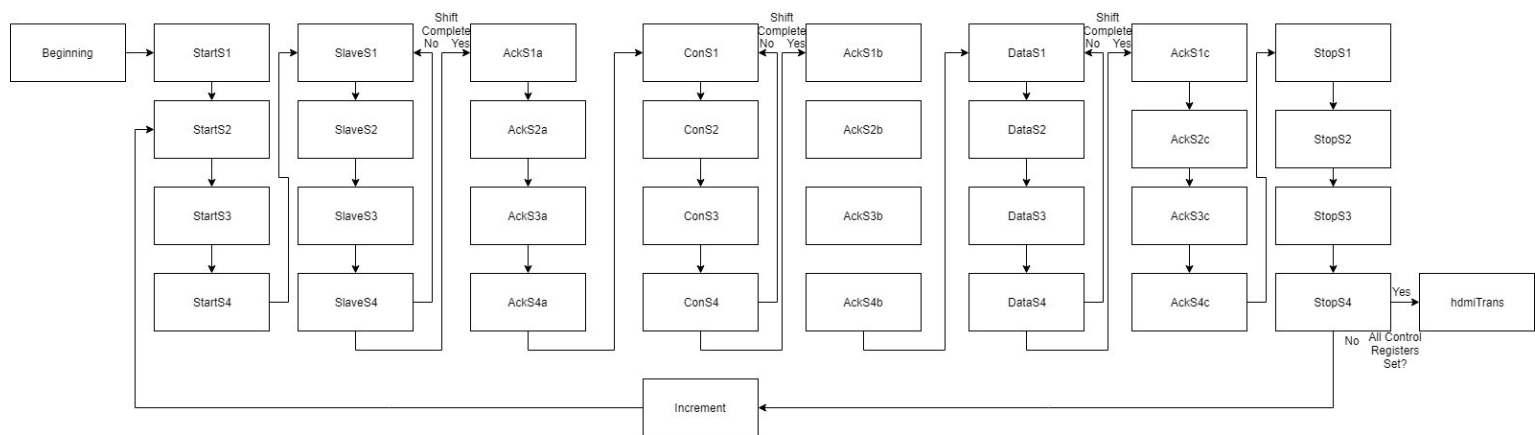


Figure 8 I2C state machine

A bigger version of this can be found in the appendix.

This state machine is used to drive the transmission at a rate of 100kHz. The 400kHz clock is used to do this. Each state transition occurs due to the positive clock edge of the 400kHz clock. One period of the 100kHz clock is defined by four states of the state machine. This allows us to have control over the waveforms. This is best explained with the states (startS1-startS4) which sends a start signal.



Figure 9 Module sim – start transfer

Wilbur Lewis-s3529819, Manar Yalid-3496165

As shown for startS1 SDA is set to 1 and SCL is set to 0. For next state we set SDA and SCL. This allows us to have a negative edge of the SDA have way through the peak of SCL. This is how to initial the transfer of information over I2C.

Once the start sequence has been set, slaveS1-slaveS4 allow us to send one bit of the slave address (0x72) the most significant bit of the slave address is sent during slaveS1-slaveS4 and then in slaveS4 the shift register is used to shift the address to get the next bit. This will continue until all bits for the slave address have been sent. The 'shiftComplete' from the shift register will cause the state to go from slaveS4 to ackS1a. ackS1a-ackS4a have SDA set to z as this is where the HDMI module will send a acknowledge bit. As shown below in figure 10 the slave address 0x72 (01110010) is sent correctly.

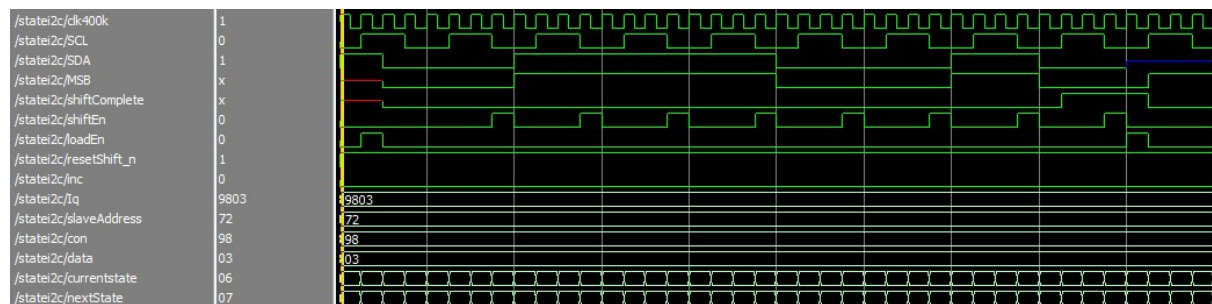


Figure 10 Module sim – slave address sent

ConS1-ConS4 and DataS1-DataS4 work in the same fashion however they send the control address and the data for the control address respectively. In the fourth acknowledge states (ackS4a,ackS4b,ackS4c) the new value to be sent is put in the shift register. In ackS4a for example, the load register puts the control register address into the shift register so it can be sent in the following ConS1-ConS4 states.

In the StopS4 state it will check if all the control registers have been sent. If it hasn't the next control register will be taken from the ROM using the increment module as explained earlier and the sequence will be repeated (setting another control register of the HDMI module). If all registers are set the HDMI transmission will be enabled.

Wilbur Lewis-s3529819, Manar Yalid-3496165

HDMI

Counters

To send with HDMI we use a PLL loop to get 25MHz. With this we use counters as shown below in figure 11.

```
myCount #(10,799) countH (
    .CEP(reset_n), // Count enable
    .PE_n(reset_n), // Parallel load enable
    .Dn(10'b0000000001), // Parallel load value
    .clock50(clk25), // System clock

    .Qn_out(Hcount), // Current count value
    .TC_out(TC_H) // Count Complete
);

myCount #(10,524) countV (
    .CEP(reset_n), // Count enable
    .PE_n(reset_n), // Parallel load enable
    .Dn(10'b0000000000), // Parallel load value
    .clock50(TC_H), // System clock

    .Qn_out(Vcount), // Current count value
    .TC_out(TC_V) // Count Complete
);
```

Figure 11 Vertical and Horizontal Counters

The pixel horizontal width that we are using is 800. Therefore, a counter (countH) is made with a size of 10 and will count to 799 using the 25MHz clock. 799 is chosen as this will make 800 including 0. This counter has a count complete output which is fed to the vertical count which will count the vertical pixels. Each time one count is completed by countH this means one row has been output. This means the next row needs to be set and so the countV is incremented to signify a new row. Once these counters complete, they will wrap around and continue counting so they can be used to set our outputs to the HDMI module: Vsync, Hsync, DataEnable (DE) and Data.

Vsync, Hsync and DataEnable (DE)

Vsync, Hsync and DataEnable (DE) is set by the following always block fed by the 25MHz clock containing an if else chain in figure 12.

Wilbur Lewis-s3529819, Manar Yalid-3496165

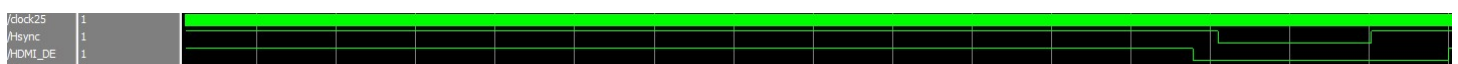
```
always@(posedge(clk25))
begin
    if(vcount >= 491) //verticle back porch
    begin
        DE = 0;
        Vsync = 1;
        if(Hcount >= 751) //horizontal back porch
        Hsync = 1;
        else if (Hcount >= 655) //horizontal sync
        Hsync = 0;
        else //horizontal front porch-active
        Hsync = 1;
    end
    else if (Vcount >=489) //verticle sync
    begin
        DE = 0;
        Vsync = 0;
        if(Hcount >= 751) //horizontal back porch
        Hsync = 1;
        else if (Hcount >= 655) //horizontal sync
        Hsync = 0;
        else //horizontal front porch-active
        Hsync = 1;
    end
    else if (Vcount >=480) //verticle front porch
    begin
        DE = 0;
        Vsync = 1;
        if(Hcount >= 751) //horizontal back porch
        Hsync = 1;
        else if (Hcount >= 655) //horizontal sync
        Hsync = 0;
        else //horizontal front porch-active
        Hsync = 1;
    end
    else if(Hcount >= 751) //verticle active, horizontal back porch
    begin
        DE = 0;
        Hsync = 1;
        Vsync = 1;
    end
    else if(Hcount >= 655) //verticle active, horizontal sync
    begin
        DE = 0;
        Hsync = 0;
        Vsync = 1;
    end
    else if(Hcount >= 639) //verticle active, horizontal front porch
    begin
        DE = 0;
        Hsync = 1;
        Vsync = 1;
    end
    else //verticle active, horizontal active
    begin
        DE = 1;
        Hsync = 1;
        Vsync = 1;
    end
end
```

Figure 12 Vsync, Hsync and DataEnable (DE) set according to Hcount and Vcount

Each if statement checks counters to find which region we are in. For example, the first if statement checks if the vertical count is greater than 491. If it is this means we are in the vertical back porch region and so DE is set to 0 and Vsync is set to 1. It then checks if we are in the horizontal back porch, horizontal Sync, horizontal front port or horizontal active. Hsync is set accordingly.

We then check if we are in the vertical sync region and the horizontal location and set DE to 0, Vsync to 0 and Hsync depending on the vertical count. This continues until DE, Hsync and Vsync are set correctly at the appropriate times.

This was simulated and found to be operating correctly. This is shown below where Hsync and DE are set due to the clock counting



Wilbur Lewis-s3529819, Manar Yalid-3496165

Figure 13 Hsync and DataEnable (DE) set correctly using 25MHz clock

Data Transfer

With the Vsync and Hsync set correctly we can set the data. This is done with the use of a ROM table to hold our data as well as an always block fed by the 25MHz clock.

```
reg [5:0] Iaddress;
wire [7:0] Iq;

red33 redy (
    .aclr(1'b0),
    .address(Iaddress),
    .clock(clk50),
    .rden(1'b1),
    .q(Iq)
);

always@(posedge(clk25), negedge(reset_n))
begin
    if(reset_n == 1'b0)
    begin
        D <= 0;
        Iaddress <= 0;
    end
    else if(Hcount <= 13 && Vcount <= 13)
    begin
        D [23:16] <= Iq;
        D [15:0] <= 0;
        Iaddress <= Iaddress+1'b1;
    end
    else if(Vcount >= 13)
    begin
        D <= 0;
        Iaddress <= 0;
    end
    else
    begin
        Iaddress <= Iaddress;
        D <= 0;
    end
end

assign Data =_D;
```

Figure 14 HDMI Data set

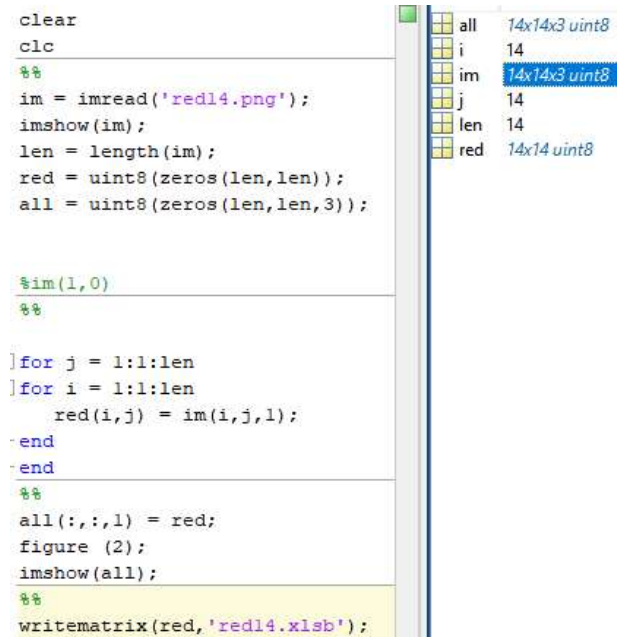
Iaddress is used to select the data from the ROM which for this example is red33. Iq outputs the data from the ROM that is selected. Knowing dimensions of the ROM we can set it so that Iaddress is incremented correctly and the correct pixel is set at the right time. In this example a very small ROM is used to hold the data for an image. It was found to output correctly as shown below. This is an image from the top right corner of the screen as shown the red pixels are lit up. This method will be implemented to do much more and to display text for the demonstration.



Image

Wilbur Lewis-s3529819, Manar Yalid-3496165

Data for the image was gathered using matlab. A script was written to separate the RGB data. In the example below the data for red is collected and output to an excel file.



```
clear
clc
%%
im = imread('red14.png');
imshow(im);
len = length(im);
red = uint8(zeros(len,len));
all = uint8(zeros(len,len,3));

%im(1,0)
%%

for j = 1:1:len
for i = 1:1:len
    red(i,j) = im(i,j,1);
end
end
%%
all(:, :, 1) = red;
figure (2);
imshow(all);
%%
writematrix(red, 'red14.xlsb');
```

all	14x14x3 uint8
i	14
im	14x14x3 uint8
j	14
len	14
red	14x14 uint8

This image is only 14 by 14 bits. The excel file converts the decimal into hex and the hex can be pasted into a hex file. The hex file is then converted to intel hex as shown in the lecture.

Conclusion:

In conclusion, this report examined how to utilise the HDMI transmitter to generate an on the on-screen image. Additionally, exploring the I2C protocol and how the two lines (SDA, SCL) are driven. Moreover, the type of commands that need to be written into the HDMI in order to get the chip to wake up. Also discussing how the image frame is read and the type of control signals, vertical sync digital signal (H-sync) and horizontal sync digital signal (V-sync)) and how these signals are used to distinguish between the pixels and know when is the end of the and where to start.

Furthermore generating PLL and a counter to bring the clock frequency to 400 kHz. The 400KHz was used to drive the state machine. All this was simulated using the module sim. Every time the data is written to the HDMI transmitter, firstly the slave address(0x72) is being sent, then the control register address and finally the data that needs to be written into the control register. A shift register that shifts one bit to the left since the data were needed to be send as one bit at a time. Each One period of the 100kHz transmutation rate is defined by four states of the state machine.

Wilbur Lewis-s3529819, Manar Yalid-3496165

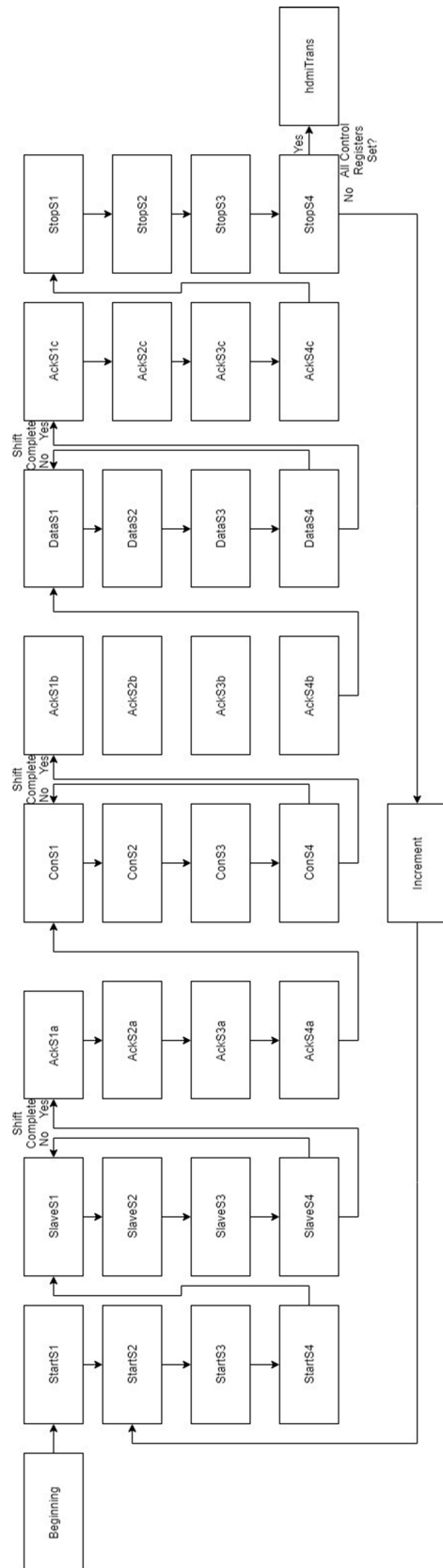
References:

- [1] G. Matthews, *MAJOR PROJECT-a) HDMI processing and overlay (on-screen display) using the Cyclone V*. pp. 1,2.
- [2] "Dual Port, Xpressview, 3 GHz HDMI Receiver, Data Sheet ADV7619", 2019. [Online]. Available: <https://www.analog.com/media/en/technical-documentation/data-sheets/ADV7619.pdf>. [Accessed: 06-Jun- 2019].
- [3] *UM10204 I2C-bus specification and user manual*. NXP Semiconductors, 2014.
- [4] RF Wireless World, *SCL and SDA*. 2019.
- [5] electrical engineering, *Communication with 7-bit I2C Addresses*. 2019.
- [6] "SPI vs I2C | Difference between SPI and I2C", *Rfwireless-world.com*, 2019. [Online]. Available: <http://www.rfwireless-world.com/Terminology/difference-between-SPI-and-I2C.html>. [Accessed: 06-Jun- 2019].
- [7] *Rfwireless-world.com, levels of SDA and SCL lines during START and STOP*. 2019.
- [8] *ADV7513 Low-Power HDMI 1.4A Compatible Transmitter PROGRAMMING GUIDE - Revision B*. 2012, p. 14.
- [9] P. Liikkanen, "Experimenting with AVR micros and VGA signal", *Operationalsmoke.blogspot.com*, 2019. [Online]. Available: <http://operationalsmoke.blogspot.com/2013/03/experimenting-with-avr-micros-and-vga.html>. [Accessed: 04- Jun- 2019].
- [10] P. Liikkanen, *Control signals*. 2019.

Wilbur Lewis-s3529819, Manar Yalid-3496165

Wilbur Lewis-s3529819, Manar Yalid-3496165

Appendix:



Wilbur Lewis-s3529819, Manar Yalid-3496165