HDMI / DVI TRANSMITTER USER GUIDE

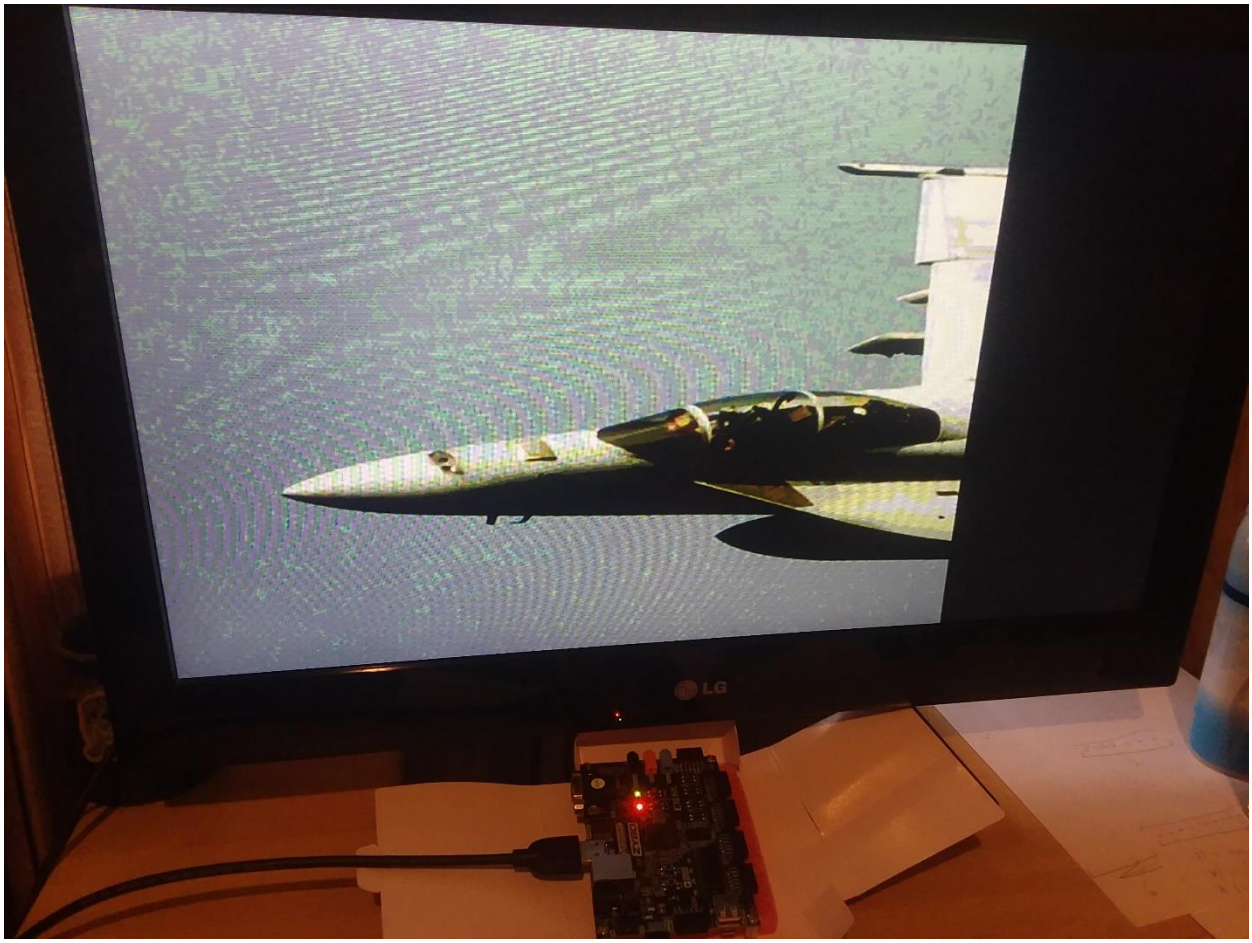


TABLE OF CONTENTS

ABSTRACT

The DVI Transmitter module converts user generated VGA signals into a single - link HDMI / DVI signal, which can be fed into the HDMI input of a monitor.

Required Input: *Valid* VGA data & control signals to control a 640 x 480 display, at a 25 MHz pixel address increment rate.

Output: Encoded DVI / HDMI signals which can be directly tied to the HDMI pins of the Zybo board via the constraint file.

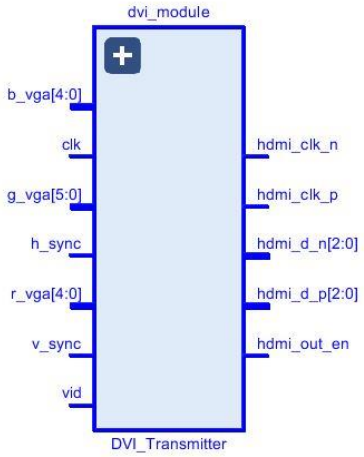
** IMPORTANT NOTICE**

This module is purposely designed to be driven by the VGA top-level design from Lab 4 (or the equivalent entities in Lab 5). If that code does not work correctly, or if it doesn't behave according to the specifications described by the lab manual, this HDMI / DVI module won't work.

Input Specification:

1. 25 MHz pixel address increment rate
2. V_sync, h_sync, and vid pulses with timing as specified by the top of page 5 in the Lab 4 manual, while using a 25 MHz clk_en
3. The timing of the VGA input must be so as to control a 640 x 480 monitor

MODULE I/O CONNECTIONS



Required Inputs

NAME	NOTES
clk	125 MHz Square Wave Clock
h_sync*	VGA h_sync output from vga_ctrl entity of Lab 4.
v_sync*	VGA v_sync output from vga_ctrl entity of Lab 4.
r_vga*	VGA red output vector from Lab 4.
g_vga*	VGA green output vector from Lab 4.
b_vga*	VGA blue output vector from Lab 4.
Vid*	Vid output from vga_ctrl entity of Lab 4

***ALL OF THESE VGA SIGNALS MUST BE TO CONTROL A 640 x 480 DISPLAY, AT A PIXEL INCREMENT (CLOCK_ENABLE) RATE OF 25 MHZ**

Outputs:

NAME	NOTES
hdmi_clk_n	Connect to hdmi_clk_n in the .xdc file
hdmi_clk_p	Connect to hdmi_clk_p in the .xdc file
hdmi_d_n	Connect to hdmi_d_n in the .xdc file
hdmi_d_p	Connect to hdmi_d_p in the .xdc file
hdmi_out_en	Connect to hdmi_out_en in the .xdc file

HOW TO INCLUDE IN YOUR PROJECT

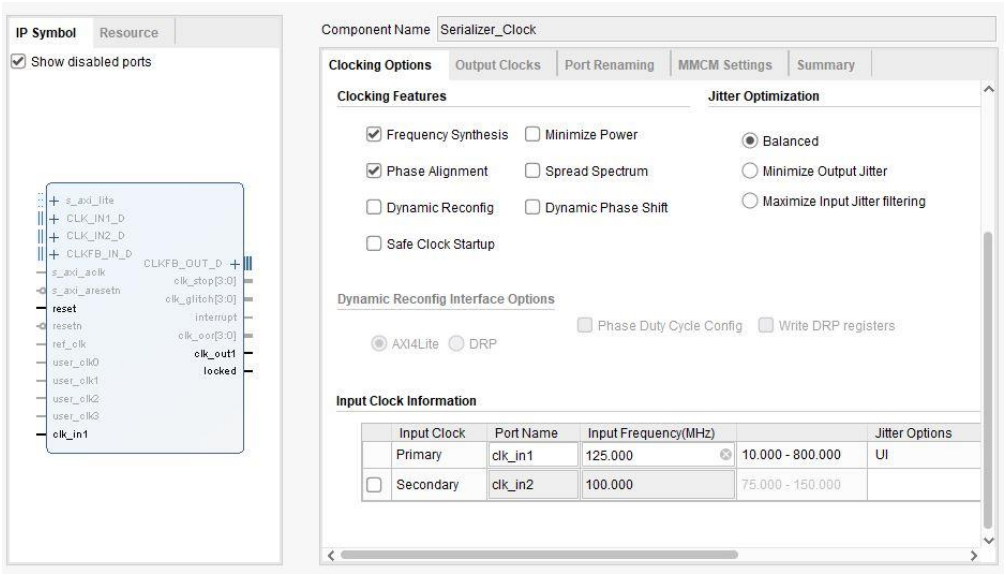
1. Download the VHDL source files for the module:
   https://github.com/AlexAmeri/HDMI_DVI_Transmitter

2. Import all of the above files into your project

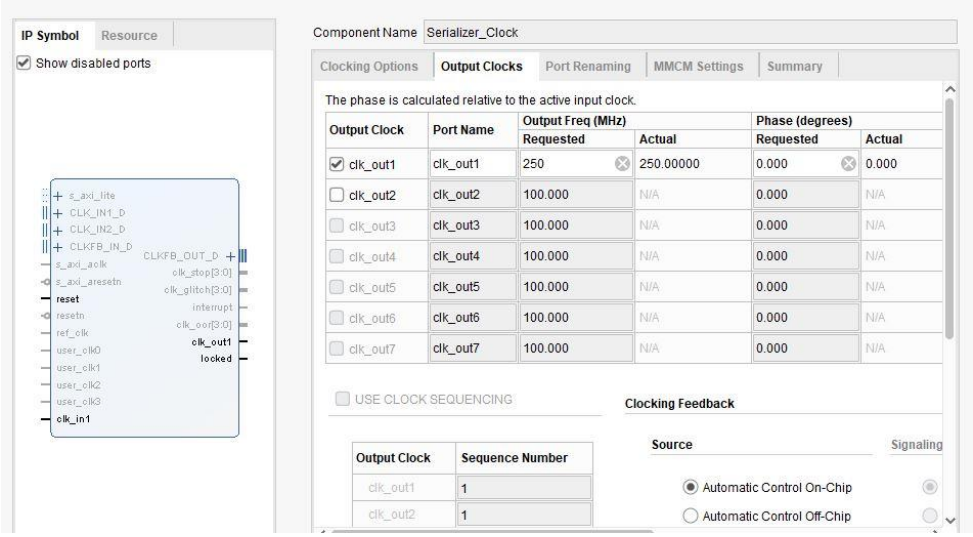
3. In the project manager, click on “IP Catalog”

4. Search for “Clocking Wizard” and click on the piece of IP that is returned from the search.

5. Change the name of the IP to “Serializer_Clock” . At the bottom of the “Clocking Options” tab, set the primary input frequency to 125 MHz.



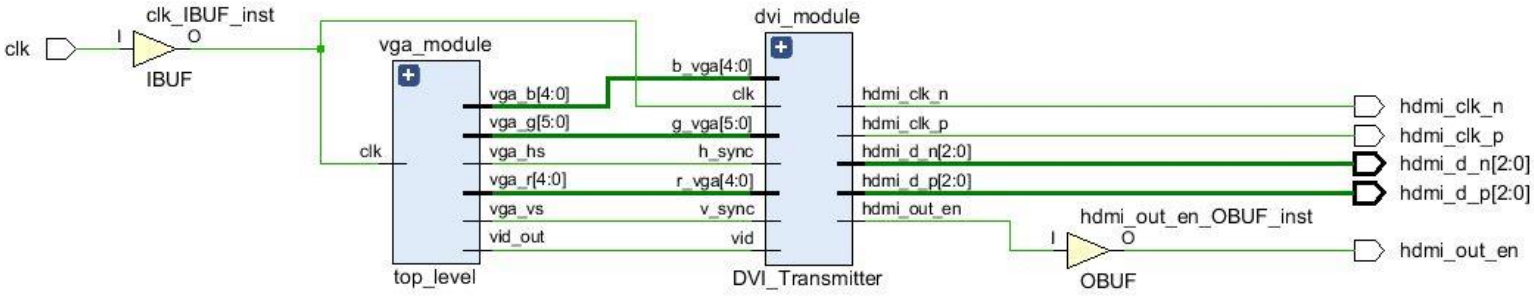
6. In the “Output Clocks” tab, set the requested frequency of clk_out1 to 250 MHz. Don’t enable any other output clocks.



7. Click “OK” to generate the IP.

8. Hook up the required signals as specified in the Module I/O Connections section.

SAMPLE PROJECT



The above picture is an elaboration schematic showing what the HDMI / DVI module looks like, when paired with the top_level module written in Lab 4.

The only modification needed to Lab 4 code, is to expose vid (which comes from vga_ctrl) as an output port for Lab 4's top level file. This is then connected to the vid input of the DVI module.

The bitfile and constraint file for this sample project are included in the github link.

THEORY OF OPERATION

Overview

Nowadays, almost all digital devices make use of the High Definition Multimedia Interface (HDMI) protocol to communicate video, audio, and control data to displays. HDMI is a backwards compatible extension of the Digital Video Interface (DVI), which communicates only video data.

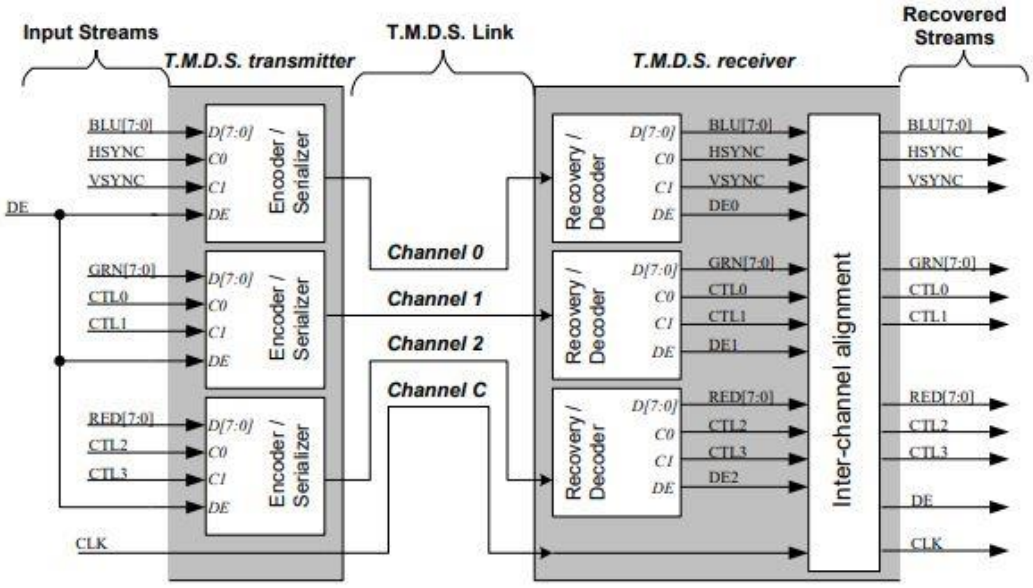
This module conforms to DVI specification 1.0, which allows for the transmission of 8 – bit R, G, and B color components.

At a high level, HDMI / DVI transmission works as follows:

1. Some piece of logic generates a VGA signal, as well as VGA controls signals (hsync, vsync, and vid) which have the correct timing for a given display.
2. If needed, the color outputs of that logic are padded to produce 8 bit R, G, and B components
3. Each color component is encoded using the Transition Minimized Differential Signaling (TMDS) algorithm, which takes as its input an 8 bit color vector, a 2 bit control vector, and a video enable flag; and produces a 10 bit output which represents an encoding of all of these inputs.
4. The 10 bit output vector is serialized and transmitted over a differential pair. There are 4 differential pairs: one for each color component, and a fourth that contains a pixel – rate clock signal.

Please see the next page for a diagram describing this process.

DVI Single Link Connection



Each channel contains a D[7 : 0] input, which is 8 bit color data, as well as 2 control bits, and a DE bit. The hsync and vsync signals, which are generated by your VGA controller, are hooked up to the two control bits of the Blue channel. Although this diagram doesn't show it, both control bits for the other two channels are tied to logic '0'. The DE input is connected to the VID output of your vga_ctrl . When DE is high, that tells the encoder that we are currently sending picture data. When DE is low, that tells the encoder that we are currently in a blanking period.

You may ask: If there are no dedicated hsync and vsync wires, then how does the receiver know the boundaries of the frame?

The answer is that hsync and vsync are encoded within the Blue channel. When (and only when) DE = 0, during the blanking period, then the 10 bit word produced by a channel's encoder will look like this:

[C1, C0] (Vsync and Hsync)	Channel Encoded Output
00	1101010100
01	0010101011
10	0101010100
11	1010101011

Because of the way the TMDS encoding algorithm works, the above four signals are never generated when DE = 1, for any of the 256 possible color inputs. So, when the receiver sees any of the above messages, it knows it is in a blanking period. In this way, by listening to the Blue channel, the receiver is able to determine hsync and vsync.

Transition Minimized Differential Signaling (TMDS)

TMDS is a data encoding scheme whose goal is to map a parallel input signal to a serial output signal, such that the serial output signal contains as few bit transitions as possible. The rationale behind this is to reduce Electromagnetic RF emission from the output wires. Recall from Faraday's and Ampere's law that RF emissions are produced by changing current in a wire, and thus to keep RF emission as small as possible, one should minimize the frequency at which the voltage (and consequently, current) changes on the transmission wires.

TMDS Encoding Algorithm

- The inputs to the TMDS encoder are an 8 bit color data vector, a 2 bit control vector, and a DE (vid) control bit.

- An internal state variable called "Cnt" keeps track of the difference between the TOTAL number of bitwise 1's and the TOTAL number of bitwise 0's which have been transmitted, since the device was powered on. Cnt(t) refers to this value calculated at iteration t, Cnt(t – 1) refers to this value calculated at iteration t – 1.

- An internal register called q(m) holds on to an intermediate value used in the calculation.

- $N_0(x)$ and $N_1(x)$ refer to the number of 0 bits and 1 bits respectively, present in "x".

Please see the next page for a flow diagram of the algorithm.

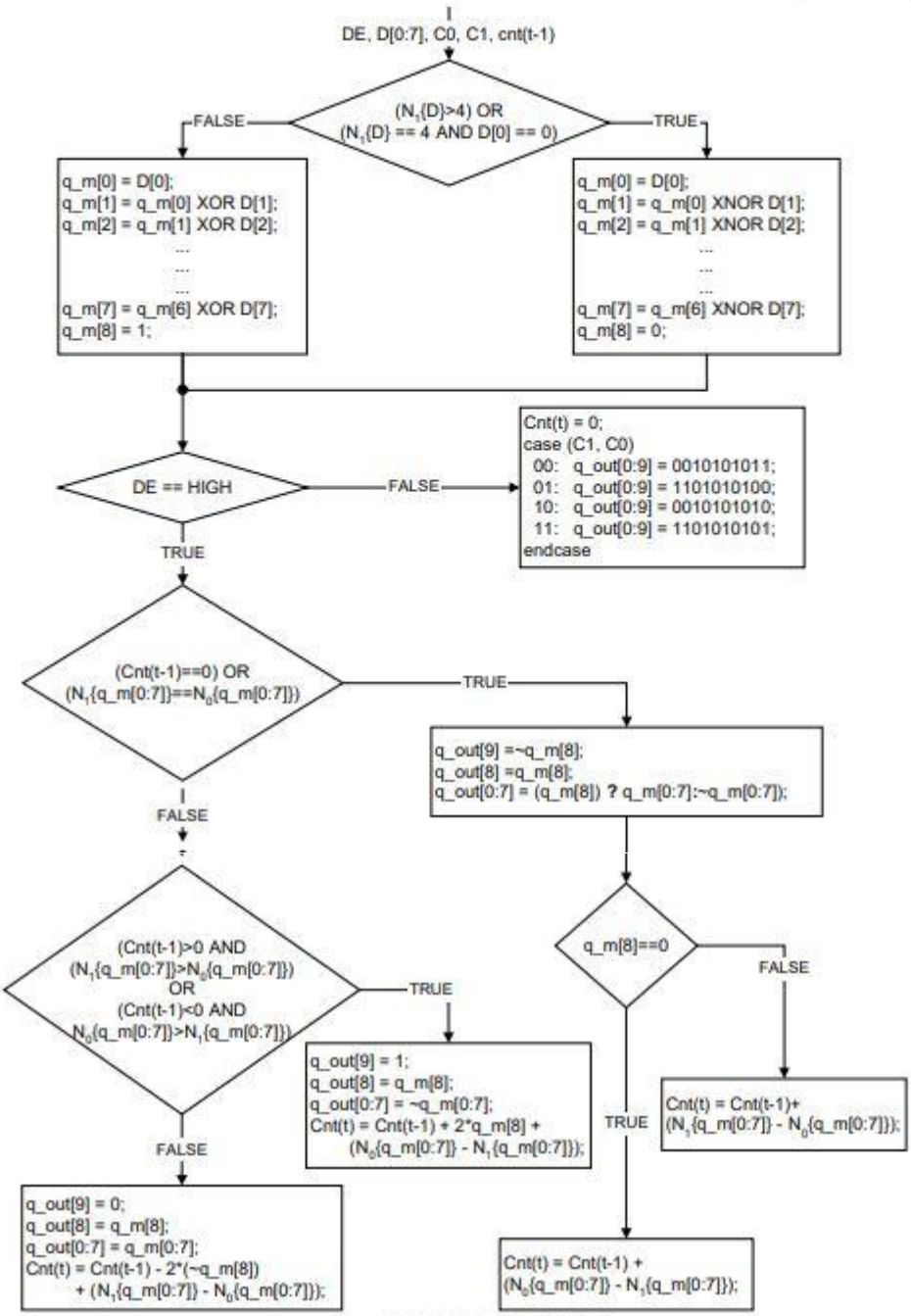


Figure 3-5. T.M.D.S. Encode Algorithm

TMDS Serialization and Clock Generation

Since each Channel in the DVI link is serial, the 10 bit output of the TMDS encoder must be serialized and sent 1 bit at a time. This requires a bit rate of at least 250 Mbps, which requires a 250 MHz clock.

250 MHz is faster than our root clock of 125 MHz, so in order to generate this clock, we must make use of a Xilinx primitive called the Mixed Mode Clock Manager (MMCM). The MMCM is a circuit inside of the FPGA which contains the utilities needed to take an input clock and multiply it to a higher frequency. These utilities consist of Voltage Controlled Oscillators, Phase Locked Loops, and clock divider circuitry.

To command an MMCM to fulfill our needs, we use Vivado's clocking wizard. We then feed the MMCM's input with a 125 MHz clock, and at 250 MHz clock will be produced as an output. To keep everything moving smoothly, the DVI transmitter creates its own 125 MHz and 25 MHz clock and clock_enable signals from this 250 MHz clock.

This creates a small problem – The circuitry inside of the DVI transmitter is now operating on a different clock domain than all the logic outside the circuit. To solve this problem, a series of FIFO registers are used at the input of the DVI transmitter. Input data and signals from your VGA logic are pipelined through using the 125 MHz clock generated by the DVI. This significantly reduces the likelihood of metastability issues occurring, and prevents the DVI from losing information.

TMDS Output Buffering

We are left with one final problem. Up to this point, our input and output signals have been making use of the Low Voltage Differential Switching (LVDS) physical specification. LVDS specifies what the input/output impedance of all pins must be, the characteristic impedance of the transmission line, and what the signal waveforms of the line must look like.

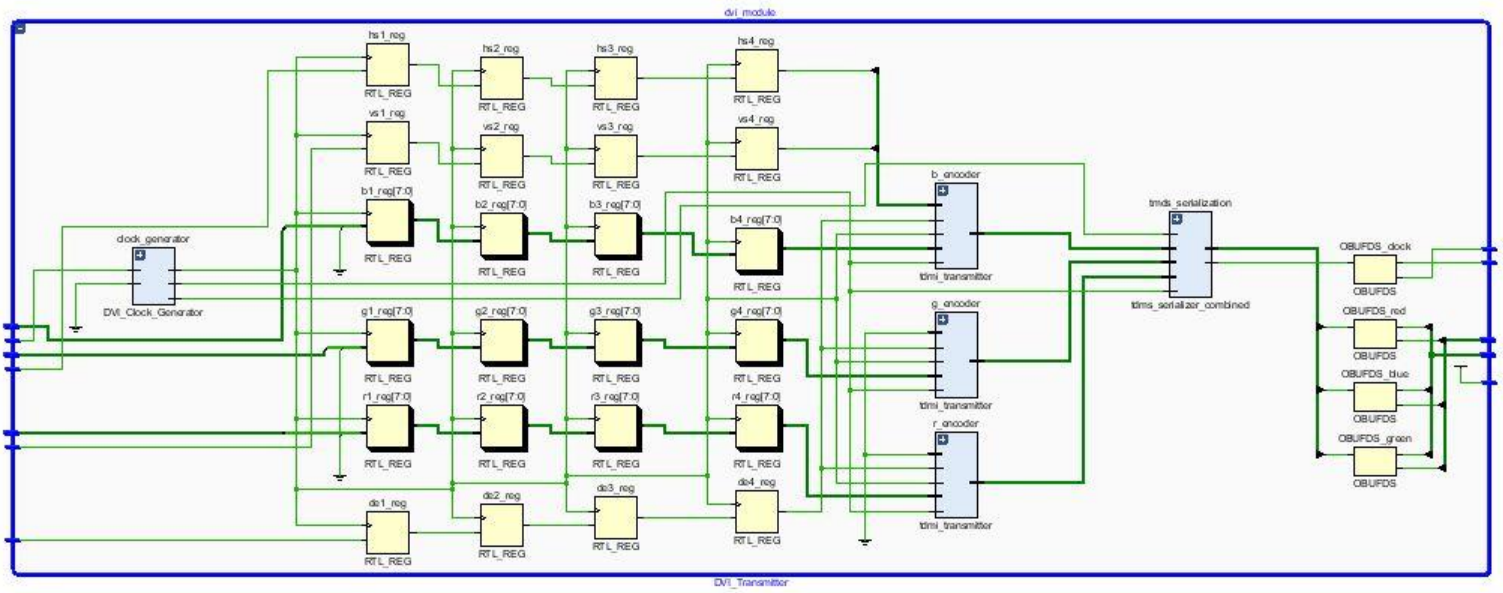
TMDS uses a different physical specification, called TMDS33, which is not compatible with LVDS. Luckily for us, the engineers at Xilinx thought to include TMDS33 output converters in the

I/O blocks of the Zynq chip for us to use. In order to use these primitives, which are called “OBUFDS” (Output Buffer Differential Signaling), we instantiate them by using the VHDL Structural paradigm. For example:

```
OBUFDS_blue  : OBUFDS port map
( O  => hdmi_d_p(0) , OB => hdmi_d_n(0), I  => TMDS_stream(0)  );
```

instantiates an OBUFDS called “OBUFDS_blue” which takes TMDS_Stream(0) as its input, and produces the two differential outputs “O” and “OB”, which are connected to signals hdmi_d_p(0) and hdmi_d_n(0), respectively.

HDMI / DVI Transmitter Elaboration Schematic



OFFICIAL DVI 1.0 SPECIFICATION

The official DVI v1.0 specification can be found here:

http://www.cs.unc.edu/Research/stc/FAQs/Video/dvi_spec-V1_0.pdf