HDMI transmitter implementation on FPGA

Emanuele Trossarello
Department of Physics
Università degli studi di Torino
emanuele.trossarello@edu.unito.it

***Abstract*—This paper focuses on the implementation on an Arty-A7 FPGA of an HDMI module to drive signals to an external screen device. Implementation is done using Verilog HDL, simulation and synthesis using Xilinx Vivado 2019.2. An additional target of this work is to understand TMDS signaling and its use in video data transmission.**

***Index Terms*—HDMI, TMDS, FPGA, ArtyA7, LVDS, Video Timings**

I. Introduction

The High-Definition Multimedia Interface (HDMI) is currently the standard for transmitting high-quality audio and video signals. When working with Field-Programmable Gate Arrays (FPGAs), implementing an HDMI interface is a relatively straightforward process although it presents its own challenges. In this implementation the main objective is to simulate and synthesize the sending of a color sequence from the FPGA to an external screen. To accomplish this it is necessary to implement a module which connects to an apposit HDMI PMOD used for cable adaptation, a dedicated TMDS encoding module and a PLL used to send data at the necessary frequency. TMDS encoding is the most complex aspect of HDMI transmission, as this encoding algorithm works to optimize voltage balance and transmission time of data sequences. This optimization is necessary because for every frame on screen it is required to send for every pixel its color information, which in standard rgb format amounts to 24 bits, on a resolution like 1920x1080@60Hz this would require a frequency of almost 1.2 GHz; to reach such speeds on cable what TMDS encoding does is crucial, adding 2 bits for each pixel color is a small drawback compared to optimized data integrity and reduced electromagnetic interference. In this implementation a resolution of 640x480@60Hz was targeted, as hardware implementation on the Arty-A7 FPGA poses significant limitations of the switching speeds of its components, thus this resolution which requires a pixel frequency of 25,2MHz was chosen.

II. Modules

In this section all the modules which were implemented will be presented and described.

A. HDMI driver

The HDMI driver, as the top level module of this implementation, is the core of the HDMI signal transmission. Its main function is to determine and track which pixel is being transmitted, this is done using two counters SX, SY updated at each clock cycle. The data necessary to render a screen frame are divided in four parts, the active screen, a backporch and frontporch, and a sync pulse both horizontal and vertical. To identify these regions of the screen's three flags are used ScreenArea, hSync and vSync. Each flag is set active when the pixel counters SX and SY, respectively for horizontal and vertical pixel position, fall inside of the correspondent region. The timings of each region for this implementation's target resolution are shown in table I alongside a visual representation of the screen sections in figure 1.

TABLE I
Timing Specifications (640x480 @ 60Hz)

Parameter	Horizontal (Pixels)	Vertical (Lines)
Visible	640	480
Front Porch	16	10
Sync Pulse	96	2
Back Porch	48	33
Total	800	525

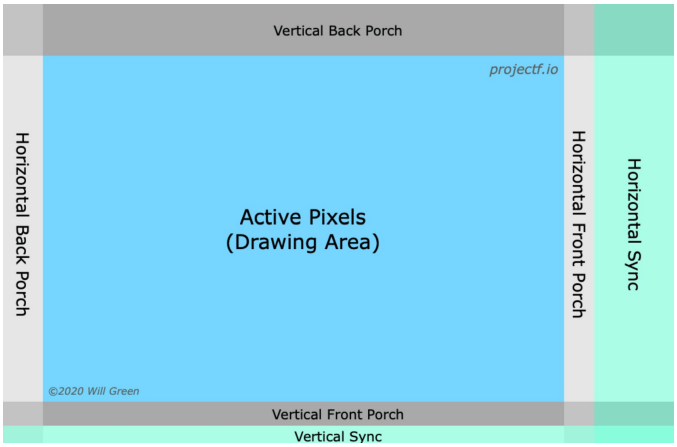

Fig. 1. Screen timing visual representation. Source: [2]

During the active screen portion actual video data are sent, while during the rest of the frame other predetermined control data sequences are transmitted. This is done to ensure precise frame syncronization which may be affected from improper clock generation on the signal's sender side. This module also uses three TMDS encoders for the conversion of the green, red and blue components of each pixel's information. It also uses three shift registers to serialize each encoder's output

for transmission. The outputs necessary for driving an HDMI cable are 4 LVDS pairs, three for the encoded colors and one for the TMDS clock. This module receives only the clock from FPGA and generates the outputs necessary for HDMI transmission, which are then mapped to an high-speed PMOD connector.

Fig. 2. HDMI driver module symbol

Below, in figure 3, is presented an image of how the connections to an HDMI cable are expected and where the data outputs from this module should be connected.

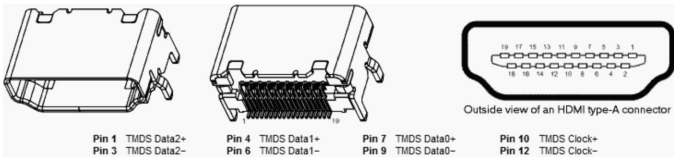

Fig. 3. HDMI cable connections Source: [3]

B. Phase Locked Loop - PLL

Xilinx's PLL ip core was used for generating the clocks necessary for HDMI signal, the Arty-A7 board provides an internal clock from a crystal oscillator at the frequency of 100MHz, the PLL from this clock generates two different clocks, a pixel clock at 25MHz and a TMDS clock 10 times faster than the previous one at 250MHz, this is necessary for sending 10 color bits for each pixel, from this calculating that there are three channel the actual data rate reached is three times the TMDS clock speed. The discrepancy between the required 25,2MHz clock and the 25MHz clock provided is a 0.8% deviation from standard may cause synchronization issues with some displays, but most screen still accept and recognize this frequency.

C. TMDS encoder

The TMDS encoder module is responsible for converting a pixel's color information to its TMDS encoded equivalent, this module expects in input an 8 bit input representing the value of a color component for a pixel, ranging from 0 to 255. This value is encoded through a specific algorithm and is presented as a 10 bit output ready to be sequentially transmitted through the HDMI cable, the TMDS encoder receives different signal to determine what its output should be, particularly important is the VDE or video data enable flag, which signals to the encoder whether it should output encoded video data or one of 4 control signals. The control signal is selected by CD a 2 bit input of the module, which in HDMI is sent only on the blue channel and assumes the values of the hSync and vSync flags from the HDMI driver module.

Fig. 4. TMDS encoder module symbol

Control data encoding

Input control bit		Output codeword
C0	C1	0 ... 9
0	0	0010101011
0	1	0010101010
1	0	1101010100
1	1	1101010101

Fig. 5. Control data for TMDS Source: [4]

III. TMDS ENCODING ALGORITHM

The TMDS encoding algorithm can be broken down into the following steps:

1) **Input Data:** 8-bit data is received.
2) **DC Balancing:** The number of '1's and '0's in the input data is counted.
3) **Conditional Inversion:** Based on the count and the running disparity, the data may be inverted.
4) **Transition Minimization Encoding:** The data is encoded into a 10-bit stream using a specific algorithm which aims to reduce the number of transitions in data sequences.
5) **Output Data:** The 10-bit encoded data is transmitted, alternated with predetermined codes used for synchronization during the non active screen portion of the signal.

A. Transition Minimization

The TMDS encoding process uses a specific algorithm to achieve transition minimization. The encoding process involves a bitwise manipulation on the input to minimize the number of transitions; these manipulations consist of converting the input signal into a new encoded signal using the XOR or the XNOR operators based on the ones versus zeros count. This temporary signal is then analyzed to predict the DC balance at the next iteration and inverted if necessary.At the end of the conversion a signal is generated where the first

bit represents inversion, the second the encoding operation chosen, while others are actual color information.

B. DC Balancing

DC balancing is crucial for minimizing the low-frequency components of the transmitted signal as HDMI transmission operates at very high frequencies on its most recent resolutions. A DC-balanced signal ensures that the average voltage level remains constant, preventing signal distortion and reducing electromagnetic interference. In TMDS, DC balancing is obtained through the following considerations:

1) Count the number of ones (N_1) in the 8-bit data word after the first encoding stage (transition minimization).
2) Calculate the disparity, which is the net sum of all ones minus all zeros, that would result from transmitting this encoded word as: $N_1 - (8 - N_1) = 2N_1 - 8$
3) Based on the current running disparity and the calculated new disparity:
   - If transmitting the non-inverted encoded word would bring the running disparity closer to zero, transmit it as is and set the inversion bit (bit 9) to '0'
   - If transmitting the inverted encoded word would bring the running disparity closer to zero, invert the encoded word and set the inversion bit to '1'
   - If both options would result in the same absolute disparity, invert the word if the current disparity is negative, otherwise transmit it as is
4) Update the running disparity by adding the disparity contribution of the transmitted 10-bit symbol

In figure 6 a summary of the whole encoder is presented.

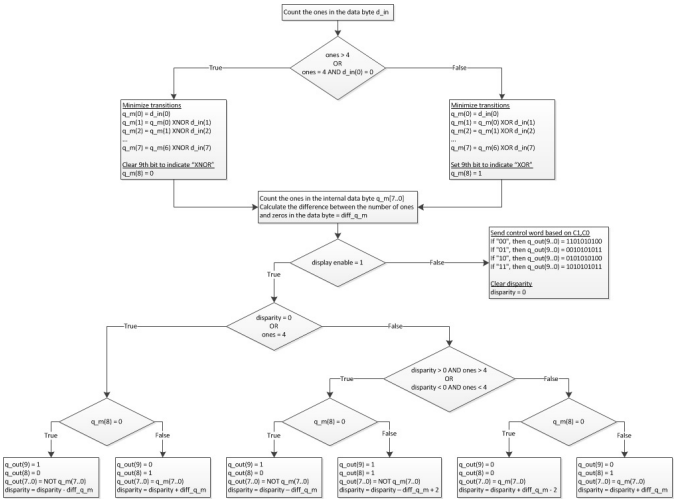

Fig. 6. TMDS algorithm flowchart. Source: [1]

IV. Simulation

To simulate this design an additional ClockGen module was used, its function to simulate the clock provided by the actual FPGA. At first a simulation with a VD input set to 0 was sent to the TMDS encoder module, and various combinations of VDE and CD were transmitted. In the red boxes of figure 7 are visible respectively, the encoded VD and three different control data as expected from the values shown in figure 5.

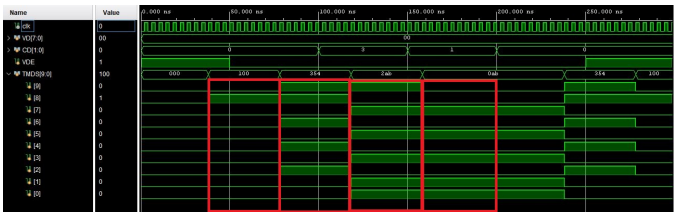

Fig. 7. TMDS encoder simulation waveforms

After the same black pixels were sent in serial transmission, in figure 8 the same pattern for the pixel can be seen, with variations induced by the DC balancing portion of the encoders.

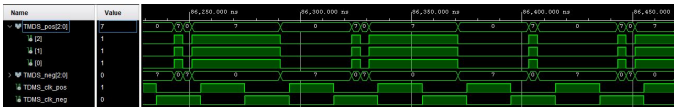

Fig. 8. Black pixels simulation

It is important to notice that during TMDS serial data transmission bits are sent in reverse order respect to how they are produced by the encoder.

In the same simulation it is also interesting to notice the stark difference between active screen data and the control signals sent while traversing the porches and the sync portions of the screen. In figure 9 and 10 it's possible to observe the transition from black pixels to the control signal of the horizontal front porch and the subsequent transition to vSync control signal. The higher number of transitions observed in the control signals is necessary for clock alignment purposes between the sender and the receiver of the HDMI signal.

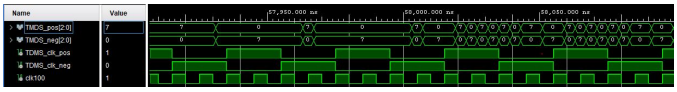

Fig. 9. Data and front porch control signal

Fig. 10. Transition from front porch to vSync control signal

V. Implementation

A. Synthesis

As the simulation behaves as expected, next is the synthesis of the design which is converted from its RTL schematic to an actual gate level implementation.
In figure 11 and 12 we can see both parts of this process.

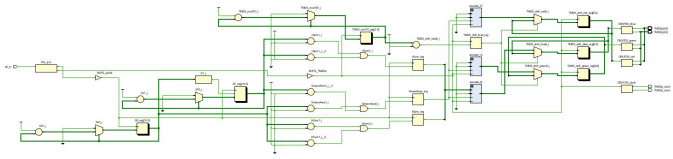

Fig. 11. RTL schematic

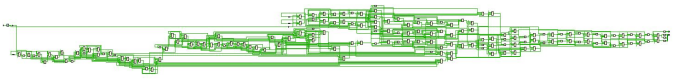

Fig. 12. Gate level schematic

B. Placement and Routing

After synthesis is completed the placement stage takes the synthesized netlist and assigns physical locations to each of the logic elements on the target FPGA fabric. The routing stage connects the placed logic elements by determining the physical paths for the signals on the FPGA's routing resources, at the end of routing Vivado provides a series of useful reports, interesting in our implementation are:

- Timing report - Figure 13
- Utilization report - Figure 14
- Device netlist placement - Figure 15

It can be seen from figure 13 that the static timing analysis results without timing violations, usually when working with elements driven at different clock speeds, like in this case with the pixel clock and the TMDS clock timing violations occur easily. In this implementation our clocks are correlated, thus no violations occur.
For resolutions higher than 1920x1080@60Hz on FPGA it was impossible to generate the required clocks with Xilinx's IP PLL, while for 1280x720@60Hz it could be done but such clock had a higher switching speed than the global buffers on the FPGA, without these buffer the clock distribution resulted in several timing violations.

Regarding resource utilization, a simple HDMI driver is not burdensome on the hardware as can be seen from the reports in figure 14 and actual placement on FPGA in figure 15. The only noticeable statistic is that this module uses up 40% of the resources dedicated to creating PLLs, which in certain applications could be a detriment.

Fig. 13. Timing report

Resource	Utilization	Available	Utilization %
LUT	60	20800	0.29
FF	73	41600	0.18
IO	9	210	4.29
PLL	2	5	40.00

Fig. 14. Utilization report

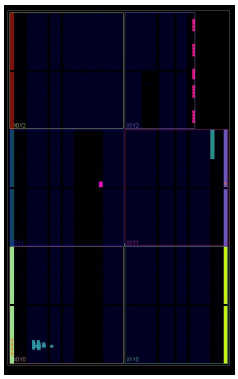

Fig. 15. Device utilization

VI. Conclusions

This implementation of the HDMI module was successfully completed with no timing errors or synthesis problems. Simulation results are aligned with expected outputs and confirm the correct working of the inner TMDS modules. Physical implementation could not be carried out due to a particular LVDS pin coupling on the Arty-A7 board, which makes its PMOD connectors not compatible with the HDMI PMOD available. Working around this problem would require a specific design of the PMOD or using connecting wire to rearrange the connections which is not feasible at these wires could introduce skew in the different parts of the signal.

REFERENCES

[1] DigiKey, "TMDS Encoder(VHDL)," DigiKey TechForum, May 12, 2020. [Online]. Available: https://forum.digikey.com/t/tmds-encoder-vhdl/12653 [Accessed: Mar. 28, 2025].

[2] W. Hammond, "Video Timings: VGA, SVGA, 720p, 1080p," Project F, Jan. 18, 2022. [Online]. Available: https://projectf.io/posts/video-timings-vga-720p-1080p/ [Accessed: Feb. 24, 2025].

[3] fpga4fun.com, "HDMI". https://www.fpga4fun.com/HDMI.html (accessed Mar. 30, 2025).

[4] Wikipedia, TMDS. https://en.wikipedia.org/wiki/Transition-minimized_differential_signaling

[5] "HDMI VGA Output Implementation on FPGA," YouTube, uploaded by Nandland, Apr. 10, 2018. [Online]. Available: https://www.youtube.com/watch?v=uQK360D3b60 [Accessed: Mar. 1, 2025].