***Advanced System on Chip Course***

**LAB 9: HDMI output**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc86680385)

[1.1 Learning Outcomes 1](#_Toc86680386)

[2 Details 1](#_Toc86680387)

[2.1 Hardware 1](#_Toc86680388)

[2.2 Logic Blocks for HDMI interface 2](#_Toc86680389)

[2.2.1 Video DMA 2](#_Toc86680390)

[2.2.2 AXI-Stream Subset Converter 2](#_Toc86680391)

[2.2.3 AXI4 Stream to Video Out 2](#_Toc86680392)

[2.2.4 RGB2DVI 2](#_Toc86680393)

[2.2.5 Video Timing Controller 2](#_Toc86680394)

[2.2.6 Dynamic Clock Generator 3](#_Toc86680395)

[2.3 Connecting the Logic Blocks 3](#_Toc86680396)

[2.4 Software programming using Vitis 13](#_Toc86680397)

[2.4.1 Initial set up 13](#_Toc86680398)

[2.4.2 Code modifications 14](#_Toc86680399)

[2.4.3 Code explanation 15](#_Toc86680400)

# Introduction

## Learning Outcomes

In this module, we will design and implement an AXI4-Stream HDMI peripheral to display a simple colored image onto an HDMI monitor.

At the end of this module, you will be able to:

* Identify what IP blocks are required to implement an HDMI output in a System on Chip
* Configure and integrate IP blocks to design a functional HDMI output peripheral.
* Modify Write a C program to control the Cortex-A9 processor and access the memory that HDMI peripheral needs to output from
* Demonstrate a functional HDMI output system that displays a colored screen depending on whether the switch input on the board.

# Details

## Hardware

Video  
DMA

HDMI  
Interface

HDMI

Interconnect

DDR Controller

GPIO  
Peripheral

LEDs

Switches

Cortex-A9

Interconnect

Programmable Logic

Processing System

Figure 1. Hardware system diagram.

## Logic Blocks for HDMI interface

### Video DMA

System Data

Video DMA

AXIS\_MM2S

AXI Subordinate

AXI Manager

System Control

To HDMI

Figure 2. VDMA block

The VDMA (Xilinx Video Direct Memory Access) module is an IP core from LogiCORE and integrated in Xilinx Vivado. It is responsible of transferring data to/from the framebuffer, which is stored in the DDR memory. There are 2 buses between VDMA core and interconnects in the SoC. One is an AXI-Lite bus used for Cortex-A9 processor to access control registers in VDMA core. Another is an AXI bus that connected to AXI (e.g. DDR memory) subordinates for video data transmission.

There are 2 kinds of AXI-Stream channels in the VDMA core. The S2MM channels are used to input video data from AXI4-Stream peripherals. This will be discussed more in the next lab. VDMA core can converts the pixel data to the AXI format and stores it to in the DDR memory. The MM2S channels are used to fetch the pixel data from AXI subordinates (DDR memory) through the AXI bus and send it to AXI4-Stream output peripherals. In this lab, only one MM2S channel is used and mapped to the memory for framebuffer.

### AXI-Stream Subset Converter

This module maps the 32-bit data from VDMA into a 24-bit RGB format data which is fed into the AXI4 Stream to Video Out module.

### AXI4 Stream to Video Out

This module is another IP core from LogiCORE and it helps bridge the AXI4 Stream interfaces to video output using the help of a Video Timing Controller. The output of this block includes parallel video data, horizontal and vertical sync signals, and data valid.

### RGB2DVI

This module takes as input a 24-bit clocked parallel video data with synchronizing signals, along with a pixel clock generated by the dynamic clock generator and outputs a TMDS signal which directly interfaces to the HDMI output on the Zybo Z7-10 FPGA board.

The 24-bit video input is an RGB pixel bus for 3 colours, horizontal and vertical sync signals and a video data enable signal.

### Video Timing Controller

The Video Timing Controller module generates timing signals based on the inputs it is given. These inputs include number of active pixels per line and number of active lines through the AXI4-Lite interface.

### Dynamic Clock Generator

We will be using a separate Dynamic Clock Generator module to generate clock signals for the Video Timing Controller, AXI-4 Stream to Video Out and RGB2DVI modules.

## Connecting the Logic Blocks

Before proceeding with connecting the logic blocks, we first need to download a repository that contains a library of the logic blocks:

1. Download the [GitHub - Digilent/vivado-library](https://github.com/Digilent/vivado-library) and save it in a local folder that has no spaces in its folder path.
2. Open up the previous lab Vivado project file (which already has the GPIO for switches and LEDs).
3. Click on **Settings** in the **Project Manager** section of the Flow Navigator pane in Vivado.
4. In **Project Settings**, expand **IP**, and click **Repository**.
5. Under IP Repositories, add the folder path that contains the Digilent Vivado library that you downloaded in step 1.
6. Click **Apply** and **OK**.

Next, we will add the IP blocks to the design by following these steps:

1. In the Vivado Project, click **Open Block Design** under the IP Integrator section of the Flow Navigator pane.
2. Click the + sign to Add IP. Type VDMA in the option and press Enter, as shown below.

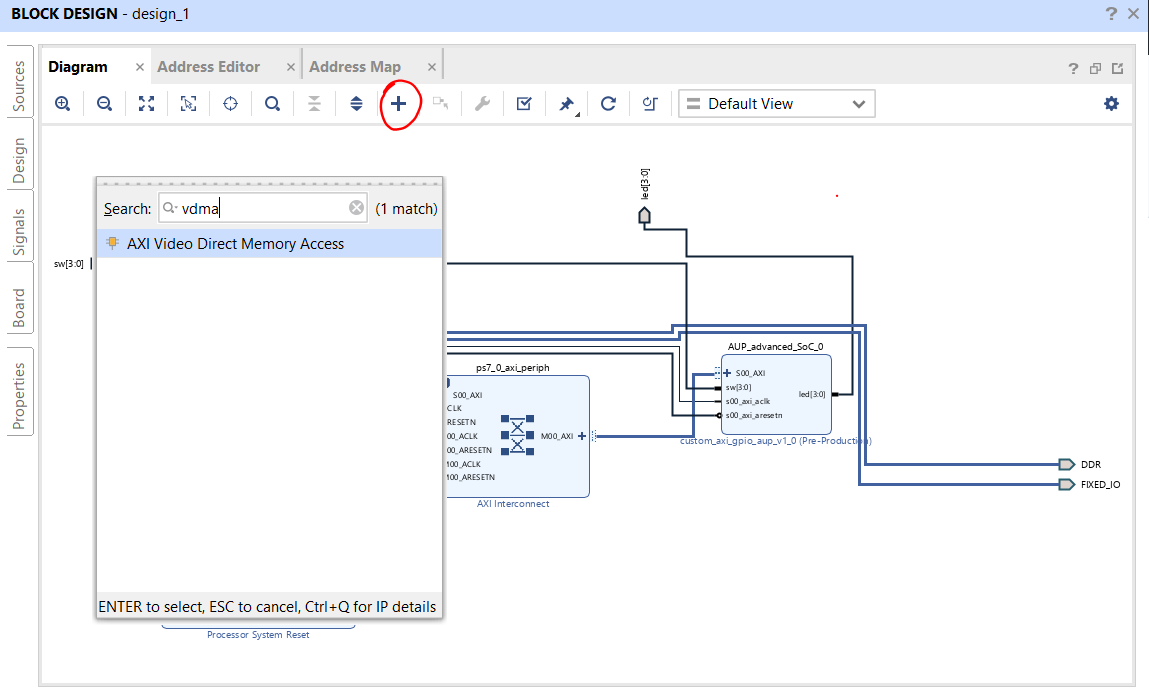


Figure 3: Add VDMA IP in block design

1. Double-click on the VDMA block (or right-click on it and select **Customize Block**). Ensure that the VDMA properties are as follows:
   1. Address width: 64
   2. Frame buffers: 2 (Note: we are using 2 frame buffers, 1 to display black, and the other to display color on the screen).
   3. Enable Read Channel **only** (we will enable write channel in the next lab for HDMI input)
   4. Memory map data width: 64
   5. Read burst size: 32
   6. Stream data width: 32
   7. Line buffer depth: 2048
   8. Advanced options – Fsync options: None, GenLock Mode: Master. **Uncheck** Allow Unaligned Transfers.
2. Next, add the AXI-Stream Subset converter and configure it so that it remaps 4 bytes of TDATA to 3 bytes of TDATA in the format of **tdata[23:16],tdata[7:0],tdata[15:8]**, as shown in the snapshot below:

Graphical user interface, application, table

Description automatically generated

Figure 4: AXI-Stream Subset Converter configuration

1. Add the AXI-Stream to Video Out module and ensure that the Fifo Depth as 4096, **Clock Mode** is **Independent**, and the **Timing Mode** is **Master**:

Graphical user interface, application

Description automatically generated

Figure 5: AXI4-Stream to Video Out configuration

1. Add the RGB2DVI module and:
   1. Uncheck ‘Reset Active High’
   2. Uncheck ‘Generate SerialClk internally’ as we will be setting up a separate clock generator for this later on.

Graphical user interface, application

Description automatically generated

Figure 6: RGB2DVI module configuration

1. Add the Video Timing Controller. Make sure you **disable/uncheck the Enable Detection** option and the video mode is set to 720p (in the Default/Constant tab):

Graphical user interface, application

Description automatically generated

Figure 7: Video Timing Controller module settings

Graphical user interface, application

Description automatically generated

Figure 8: Set video mode to 720p in the Video Timing Controller module

1. Add the Dynamic Clock Generator and ensure that the **Add BUFMR is unchecked**.

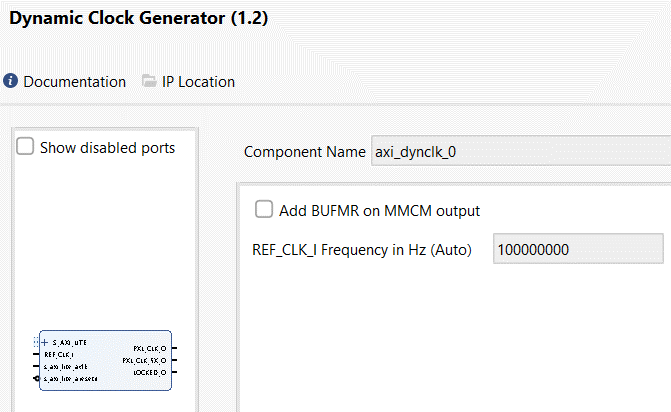


Figure 9: Dynamic Clock Generator module settings

1. Add a constant IP and set the value to 0, this will be used to ensure that the aresetn input of the AXIS Subset Converter is always 0, and not floating:

Graphical user interface

Description automatically generated

Figure 10: Constant block configuration

1. Right-click on the TMDS output signal in the Rgb2dvi module (when selected, only the TMDS signal will turn orange instead of the whole module), and click **Make External**. A TMDS\_O pin will now appear on the diagram. Now right-click on the external pin and click **External Interface Properties**. Then rename this pin as hdmi\_output as shown.

Graphical user interface, application

Description automatically generated

Figure 11: Rename output pin

1. Double-click on the Zynq7 Processing System block (that was added in the previous lab), and select PS-PL Configuration in the Pane. Then expand HP Slave AXI interface and tick the S AXI HP1 interface as shown. Also set FCLK\_CLK to 100.

Graphical user interface, text, application, email

Description automatically generated

Figure 12: HP Configuration for Zynq

Graphical user interface, application, table

Description automatically generated

Figure 13: FCLK configuration

1. Now manually draw the wire connections to connect the modules so that they are connected as shown:

|  |  |
| --- | --- |
| **Signal** | **Connect to** |
| AXI VDMA: m\_axis\_mm2s\_aclk | AXI Subset Converter: aclk |
| AXI VDMA: M\_AXIS\_MM2S | AXI Subset Converter: S\_AXIS |
| Constant: dout | AXI Subset Converter: arestn |
| AXI Subset Converter: M\_AXIS | AXI4 Stream to Video Out: video\_in |
| AXI4 Stream to Video Out: vid\_io\_out | Rgb2dvi: RGB |
| Dynamic Clock Generator: REF\_CLK\_I | Dynamic Clock Generator: s00\_axi\_aclk |
| Dynamic Clock Generator: PXL\_CLK\_O | Video Timing Controller: clk  AXI4-Stream to Video Out: vid\_io\_out\_clk  Rgb2dvi: PixelClk |
| Dynamic Clock Generator: PXL\_CLK\_5X\_O | Rgb2dvi: SerialClk |
| Dynamic Clock Generator: LOCKED\_O | Rgb2dvi: aRst\_n |

Your block design should now look similar to this:

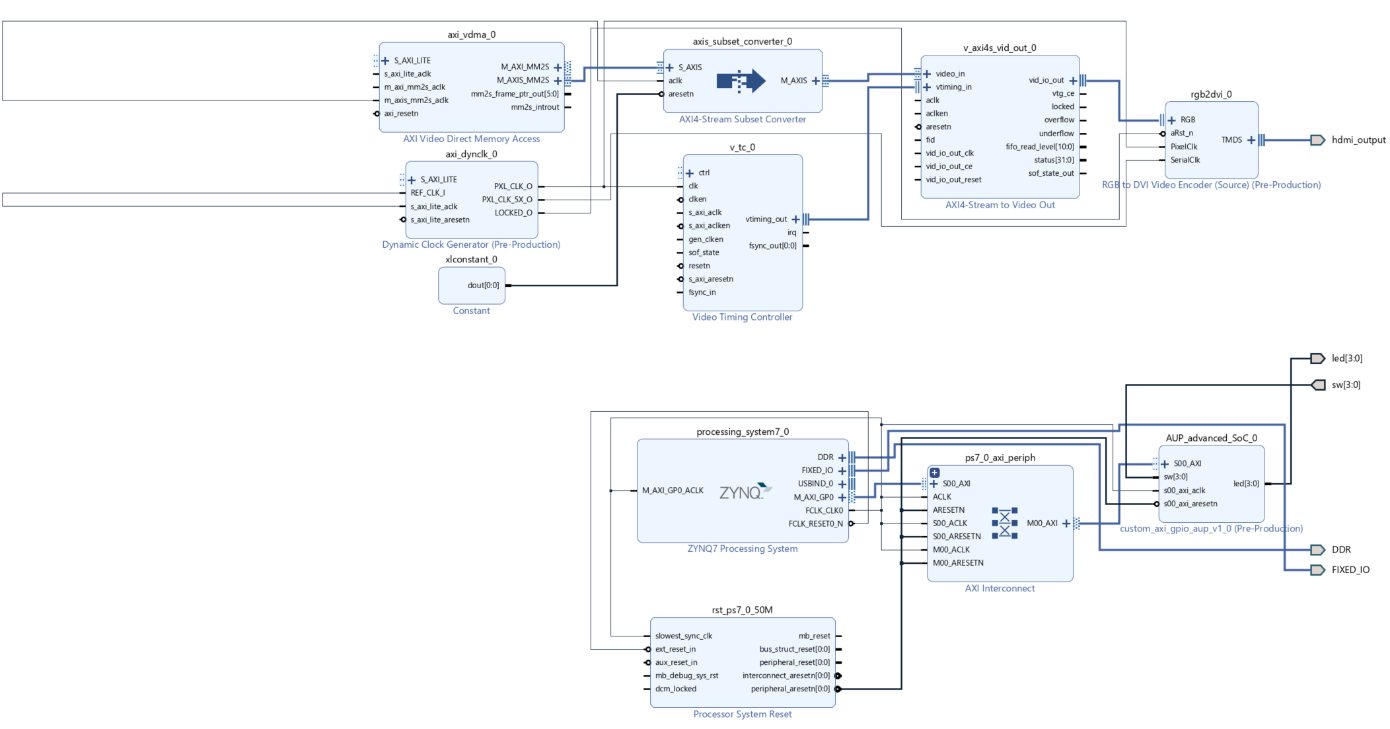


Figure 14: Block design

1. Next, click **Run Connection Automation** (this option is available in a green bar just above the block design). Ensure that the connections to be automated are as shown:

Graphical user interface, application

Description automatically generated

Figure 15: Connections to automate

1. After running the Connection Automation, your block diagram should look like this:

Diagram, schematic

Description automatically generated

Figure 16: Block diagram fully connected

1. Update the XDC file so that it also contains the following:

# Zybo Z7 HDMI TX pins

set\_property -dict { PACKAGE\_PIN H17 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_clk\_n }]; #IO\_L13N\_T2\_MRCC\_35 Sch=hdmi\_tx\_clk\_n

set\_property -dict { PACKAGE\_PIN H16 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_clk\_p }]; #IO\_L13P\_T2\_MRCC\_35 Sch=hdmi\_tx\_clk\_p

set\_property -dict { PACKAGE\_PIN D20 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_n[0] }]; #IO\_L4N\_T0\_35 Sch=hdmi\_tx\_n[0]

set\_property -dict { PACKAGE\_PIN D19 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_p[0] }]; #IO\_L4P\_T0\_35 Sch=hdmi\_tx\_p[0]

set\_property -dict { PACKAGE\_PIN B20 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_n[1] }]; #IO\_L1N\_T0\_AD0N\_35 Sch=hdmi\_tx\_n[1]

set\_property -dict { PACKAGE\_PIN C20 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_p[1] }]; #IO\_L1P\_T0\_AD0P\_35 Sch=hdmi\_tx\_p[1]

set\_property -dict { PACKAGE\_PIN A20 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_n[2] }]; #IO\_L2N\_T0\_AD8N\_35 Sch=hdmi\_tx\_n[2]

set\_property -dict { PACKAGE\_PIN B19 IOSTANDARD TMDS\_33 } [get\_ports { hdmi\_output\_data\_p[2] }]; #IO\_L2P\_T0\_AD8P\_35 Sch=hdmi\_tx\_p[2]

1. Generate the bitstream (including synthesize and implement) and open the Hardware Manager. Then, connect the FPGA board, click autodetect. Right-click on the FPGA device and click Program Device.
2. Export hardware platform (include bitstream) and save the .xsa file in a location which hasn’t got any spaces in the path.

Note: It is also recommended that you read the following material as additional reference:

* <https://wiki.york.ac.uk/display/RTS/Zybo+Z7+HDMI+Output>
* <https://digilent.com/reference/learn/programmable-logic/tutorials/zybo-z7-hdmi-demo/start>
* [Basics of Video Processing on the FPGA of a Zybo using VHDL (I) - Mis Circuitos](https://miscircuitos.com/video-processing-fpga-zybo-using-vhdl/)
* Critical warnings: Hardware Errata - https://digilent.com/reference/programmable-logic/zybo-z7/reference-manual?redirect=1#hardware\_errata

## Software programming using Vitis

### Initial set up

We will be using the APIs provided by Digilent to program the logic blocks to enable HDMI output.

1. Download and install PuTTY, which is a free SSH and Telnet client for Windows.
2. Go to <https://github.com/Digilent/Zybo-Z7-10-HDMI/tree/master/sdk/appsrc> and download the following folders:
   1. Display\_ctrl
   2. Dynclk

Note: You may have to clone or download a zipped version of the whole repo in [GitHub - Digilent/Zybo-Z7-10-HDMI](https://github.com/Digilent/Zybo-Z7-10-HDMI)

1. Open Vitis and launch in the same workplace that contains the .xsa file that you have exported earlier.
2. Create a new application project by selecting: **File** > **New** > **Application Project** **> Create a new platform from hardware (XSA)** > **Browse** > (select previously exported .xsa file) > **Open** > **Next** > (give an application project name) > **Next** > **Next** > (select Hello World) > **Finish**.
3. Build the platform project.
4. In the Explorer view of Vitis, expand the Application Project and right-click the src folder. Select **Import Sources** > **Browse**. Choose the path that contains the folders you downloaded from the Digilent/Zybo-Z7-10-HDMI GitHub repo and then click **Finish**, as shown:

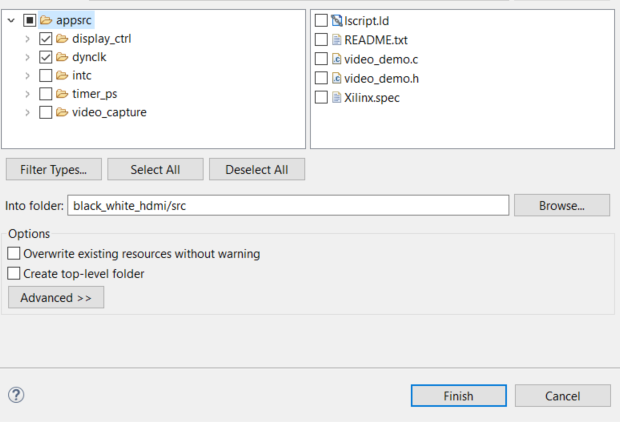


Figure 17: Imported files

1. Build the Application Project.
2. Connect the FPGA board to your PC and make sure that the board is powered up.
3. Launch PuTTY and in the Session settings, select Serial. Fill in the correct COM port and ensure that the speed is set to 115200. (You can find out which COM port you are using by going to Device Manager in Windows, expand Ports > USB Serial Port. Make sure you FPGA board is powered up). Click **Open** in PuTTY. A blank terminal will appear.
4. Run the simple Hello World application to verify that all the setup is good. Right-click on the Application project and select Run As > Launch Hardware. You should see that Hello World has successfully run, and the PuTTY terminal should contain the following output:

Graphical user interface, text

Description automatically generated

Figure 18: Hello World output

### Code modifications

Now that you have gotten Hello World application to work and rule out any environment and build issues, next you will need to modify helloworld.c so that you are able to display black or a color via the HDMI output, depending on whether you turn on SW1 on the FPGA board. Use the functions provided by files you have imported from the Digilent/Zybo-Z7-10-HDMI GitHub repo.

1. Modify display\_ctrl.h and display\_ctrl.c files so that:
   1. DisplayCtrl struct contains a u32 \*framePtr
   2. DisplayInitialize function accepts a u32 value for \*framePtr.
   3. DISPLAY\_NUM\_FRAMES is set to 2
   4. dispPtr->vdmaConfig.HoriSizeInput = (dispPtr->vMode.width) \* 4;
2. Copy the code provided with this lab into helloworld.c.
3. Rebuild application project and load program onto FPGA board.

You should notice that when you flick the first switch, it will display light blue color. If the switch is turned off, it will display black. You can now modify the code to change the light blue colour to white.

### Code explanation

The first thing we do is create an instance of “DisplayCtrl” which is the display driver struct.

DisplayCtrl dispCtrl;

The definition of this struct is given within the file “display/display\_ctrl.h”. This is an easy to use API for controlling a monitor connected via HDMI. After creating an instance, we pass a pointer pointing to this instance as an argument to the function “DisplayInitialize”. The other arguments include the VDMA, the Video Timing Controller ID, base address to the Dynamic clock module, an array of pointers to the frame buffer and the line stride of the framebuffers in bytes.

DisplayInitialize(&dispCtrl, &vdma, XPAR\_VTC\_0\_DEVICE\_ID, XPAR\_DYNCLK\_0\_S\_AXI\_LITE\_BASEADDR, pFrames, FRAME\_STRIDE);

Once the VDMA has been configured, the “DisplayInitialize” function enables all the appropriate signals and drivers for the video timing controller and the VDMA to start working.

The following line of code assigns 2 frames and initiate the pixel RGB values for those frames.

frame1[y\*stride + x] = 0x91B2C7;

frame0[y\*stride + x] = 0;

Finally, we create an infinite while loop to continuously read the switch value. It displays a light blue/black screen depending on the value of switch 0.