Lab 4.0

Camera, LCD & VGA implementation

# Introduction

In lab 3.0 you proposed a detailed design for an FPGA-based system which can interface with one of the following peripherals on the DE0-Nano-SoC:

* [TRDB-D5M](http://www.terasic.com.tw/cgi-bin/page/archive.pl?No=281) camera
* [LT24](http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=68&No=892) LCD display
* [CDK3404](https://www.exar.com/content/document.ashx?id=21420) Triple Video DAC

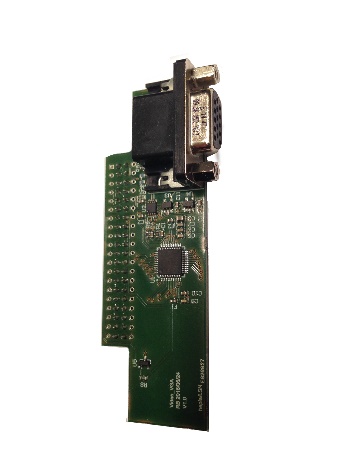
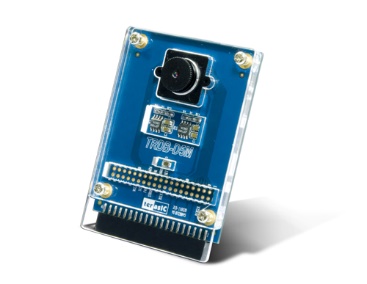


Figure . TRDB-D5M, LT24 & VGA Interfaces

# Goal

The goal of lab 4.0 is for you to implement the detailed design you proposed in lab 3.0:

1. Each group of 2 students will independently implement their camera, LCD, or VGA design. You will need to test your implementation by performing suitable simulations with ModelSim to ensure that the design functions as you expected. Before you implement your design, remember that your two groups must agree on the format a frame will have once in memory for your designs to work correctly once put together.
2. Once your simulations are conclusive and correspond to what you expected, we will then provide you with the actual camera, LCD or VGA extension boards so you can test your design on the real hardware.
   1. For the groups implementing the camera design, the ultimate test will be to save a frame in memory then transfer it to your host PC so you can visually inspect it with an image viewer.
   2. For the groups implementing the LCD or VGA design, the ultimate test will be to store a frame in memory from your host PC, then to output it through your LCD/VGA controller so you can visually inspect the result on the target displays.
3. Once each team’s hardware design functions correctly in isolation, you will then proceed to merge the designs together to obtain a complete frame acquisition and visualization system ☺

# Theory

## I/O with the host PC

When it comes to testing your designs, you will need to have a way to transfer an image from a file on your host PC to the memory on the development board (LCD/VGA design), or from the development board’s memory to a file on your host PC (camera design).

Essentially, we need the Nios II processor to be able to orchestrate the reads/writes from/to a file on your host PC through standard C code. To do this, we need to enable a specific software package in the BSP Editor.

After creating your Nios II SBT project, follow the steps below to enable this software package:

1. Right-click on the BSP project > Nios II > BSP Editor …
2. In the Software Packages tab, enable the altera\_hostfs package.
3. Save the configuration.
4. Press the Generate button.
5. Close the dialog. From this point onwards, you can continue to use the Nios II development tools just as you were doing in the previous labs.

Figure 2 shows the altera\_hostfs configuration dialog where you specify the name of the virtual directory used on the development board to access the host filesystem (“/mnt/host”), and the example below shows how you can use the fprintf() function to write to a file on the host filesystem. A very similar code snippet would use the fscanf() function to read from the host filesystem.

#include <stdio.h>  
  
char\* filename = "/mnt/host/image.ppm"  
  
FILE \*foutput = fopen(filename, "w");  
if (!foutput) {  
 printf("Error: could not open \"%s\" for writing\n", filename);  
 return false;  
}  
  
/\* Use fprintf function to write to file through file pointer \*/  
fprintf(foutput, "Writing text to file\n");



Figure . hostfs Software Package

## Simulating camera output signals

For teams designing the camera acquisition interface, there exists an additional difficulty when simulating your design: feeding your camera controller a representative set of signals as what would be outputted by the camera.

To ease the testing process for the camera designs, we provide you an IP core which is directly instantiable in Qsys and generates such signals: the cmos\_sensor\_output\_generator. It provides a 32-bit Avalon-MM slave control interface, and exports a conduit interface containing the frame\_valid, line\_valid, and data signals which a camera would have outputted.



Figure . cmos\_sensor\_output\_generator interface

Its documentation and a demo of how to use its C library can be found in the lab template.

## Communicating with the camera

The TRDB-D5M camera exposes an I2C control interface, so you need an I2C controller to be able to read/write its internal registers. We provide you an IP core which is directly instantiable in Qsys for this purpose: the i2c controller. It provides an 8-bit Avalon-MM slave control interface, and exports a conduit interface containing the scl and sda pins.

A demo of how to use its C library can be found in the lab template.

# Practice

## Template

Unlike the previous labs, there is not enough on-chip memory available on the development board to store a full frame. Instead, you will be using the external DDR3 memory available on the board. Setting up the DDR3 memory interface is more involved than necessary for this lab, so we provide you a [template](http://moodle.epfl.ch/mod/resource/view.php?id=937864) where the setup is already done.

The template contains 2 projects depending on which extension boards you are using:

* “ES\_mini\_project\_TRDB\_D5M\_LT24.qpf” (TRDB-D5M & LT24)
* “ES\_mini\_project\_TRDB\_D5M\_VGA.qpf” (TRDB-D5M & VGA)

## IP Catalog

As stated previously, we provide you with two IP cores which will help you for the TRDB-D5M design. Figure 4 shows where you can find the IP cores in the Qsys IP catalog.

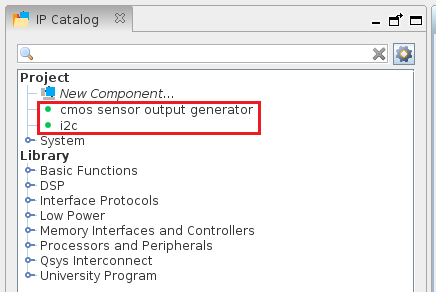


Figure . IP Catalog

## Qsys System

Figure 5 shows the base Qsys system we provide you. You must add the custom IP core any other component deemed necessary (PLL, …) for interfacing with your camera, LCD/VGA controller appropriately here.

Since there is not enough on-chip memory available to store a full frame, you will only use the on-chip memory for the Nios II processor’s instruction and data memory, but will need to connect your camera or LCD/VGA designs’ master ports to the DDR3 memory.

The DDR3 memory’s slave port provides access to a large 1 GB memory. Despite having all this memory available, some of it is reserved for use for other purposes and cannot be touched by your masters (come to CS-476 Real-Time Embedded Systems to learn more!).

To prevent you from accessing any region of the DDR3 memory that is off limits, we have used a bridge (address\_span\_extender\_0.windowed\_slave) to interface with port 0 of the DDR3 memory controller (hps\_0.f2h\_sdram0\_data). Accesses through the bridge are redirected to a specific offset within the DDR3 memory where we have reserved 256 MB of continuous memory which is safe to access freely by your masters.

Because of this, please do not connect your masters directly to port 0 of the DDR3 memory controller, but instead go through the bridge (address\_span\_extender\_0.windowed\_slave) for all accesses.

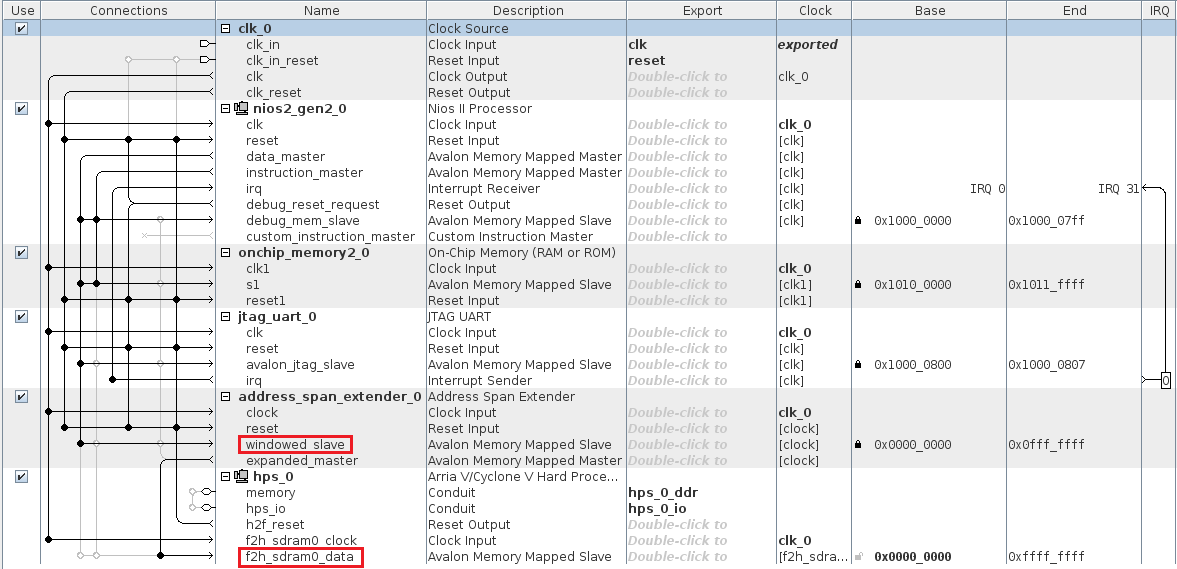


Figure . Base Qsys system with DDR3 Interface Bridge

## Sdcard

In order to use the DDR3 memory, you need to set up and configure the ARM processor’s memory controller on the development board. This is done by writing a preloader and bootloader to the DE0-Nano-SoC’s sdcard, but is quite an involved process and is outside the scope of this course (we see this in CS-309 & CS-476).

The TAs will get you up and running on this front, so we ask that you please bring us your development board’s sdcard so we can prepare it for you before you test your system with actual hardware. Please do not try to test the project without giving us your sdcard first. It will save you many hours of trouble and much frustration debugging unresponsive memory accesses (Trust us, we’ve been there ☺).