

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](#) camera
- [LT24](#) LCD display
- [CDK3404](#) Triple Video DAC



FIGURE 1. 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.
 - a. 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.
 - b. 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");
```

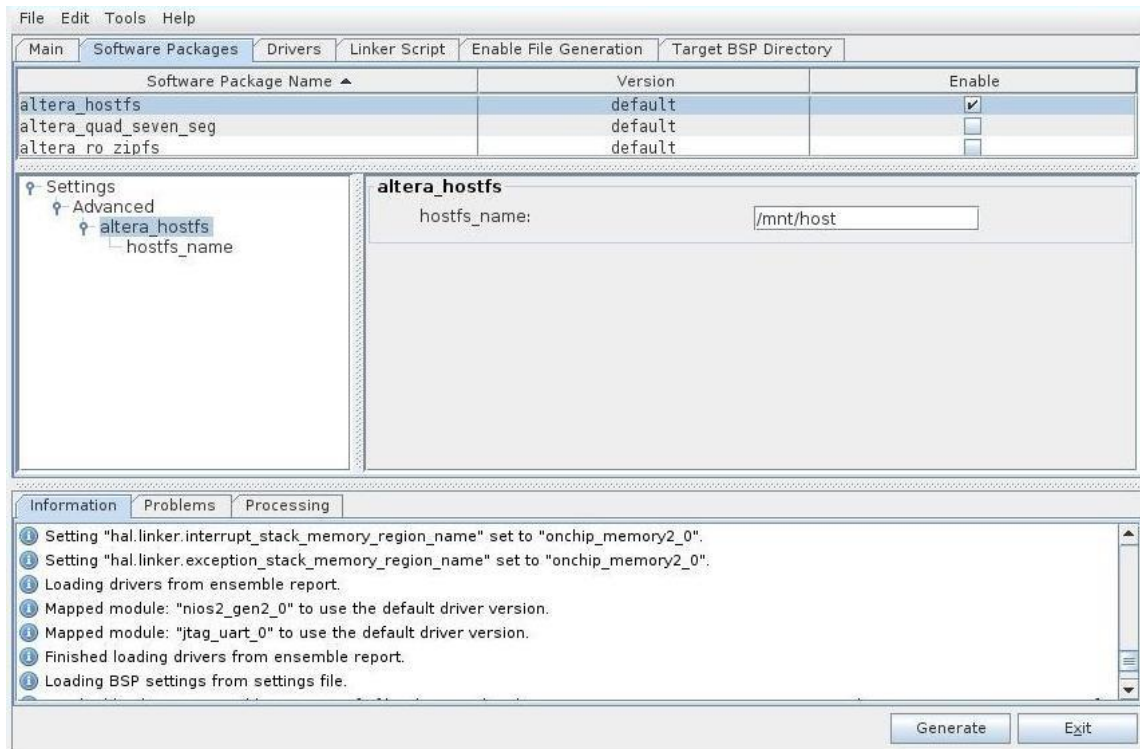


FIGURE 2. HOSTFS SOFTWARE PACKAGE

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.

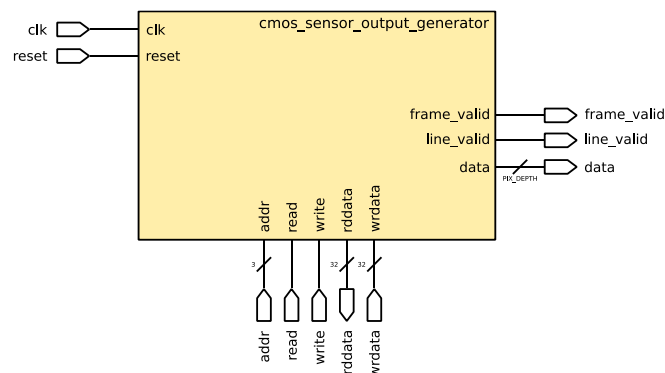


FIGURE 3. CMOS_SENSOR_OUTPUT_GENERATOR INTERFACE

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](#) 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.

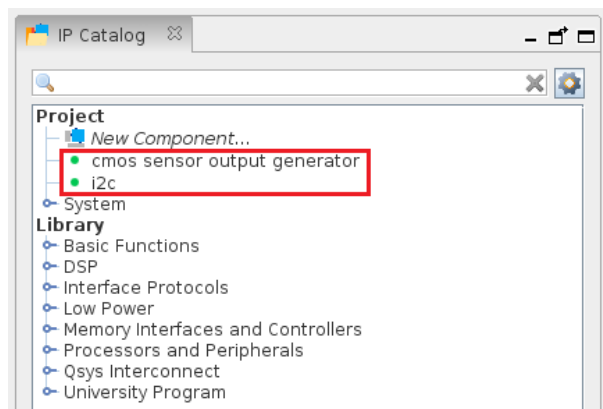


FIGURE 4. IP CATALOG

Qsys System

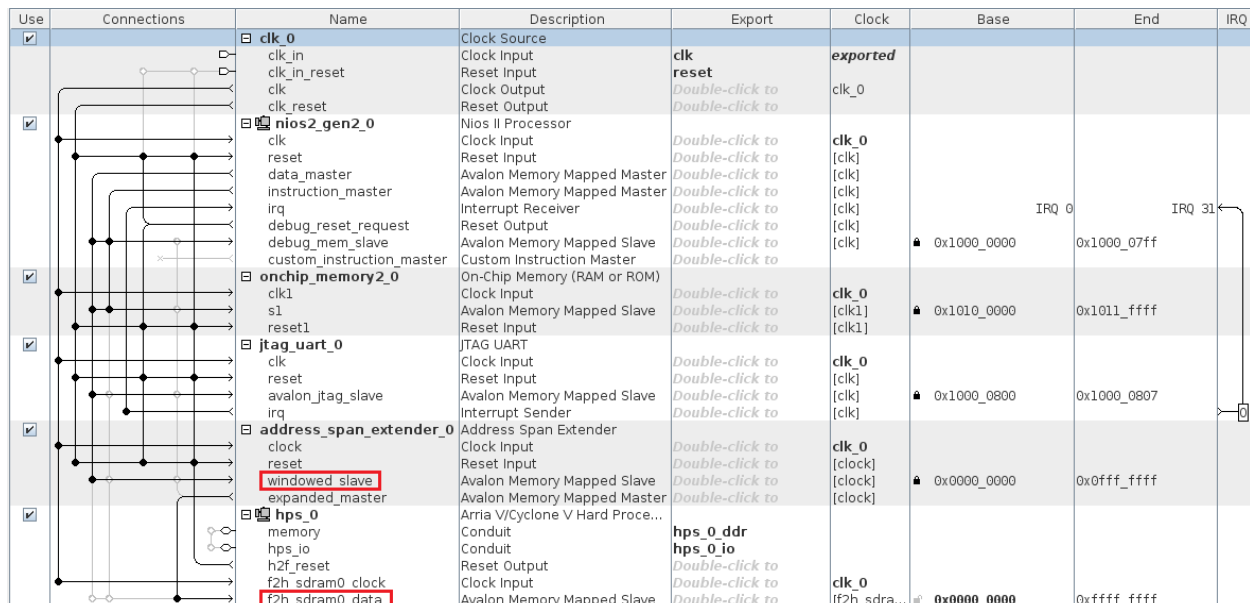
Figure 5 shows the base Qsys system we provide you. You must add the custom IP core any other component deemed necessary (PLL, ...) for interfacing with your camera, LCD/VGA controller appropriately here.

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.



Use	Connections	Name	Description	Export	Clock	Base	End	IRQ
<input checked="" type="checkbox"/>		clk_0	Clock Source		exported			
		clk_in	Clock Input	clk				
		clk_in_reset	Reset Input	reset				
		clk	Clock Output	Double-click to	clk_0			
		clk_reset	Reset Output	Double-click to				
<input checked="" type="checkbox"/>		nios2_gen2_0	Nios II Processor					
		clk	Clock Input	Double-click to	clk_0			
		reset	Reset Input	Double-click to	[clk]			
		data_master	Avalon Memory Mapped Master	Double-click to	[clk]			
		instruction_master	Avalon Memory Mapped Master	Double-click to	[clk]			
		irq	Interrupt Receiver	Double-click to	[clk]			IRQ 0
		debug_reset_request	Reset Output	Double-click to	[clk]			
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x1000_0000	0x1000_07ff	
		custom_instruction_master	Custom Instruction Master	Double-click to	[clk]			
<input checked="" type="checkbox"/>		onchip_memory2_0	On-Chip Memory (RAM or ROM)					
		clk1	Clock Input	Double-click to	clk_0			
		s1	Avalon Memory Mapped Slave	Double-click to	[clk1]	0x1010_0000	0x1011_ffff	
		reset1	Reset Input	Double-click to	[clk1]			
<input checked="" type="checkbox"/>		jtag_uart_0	JTAG UART					
		clk	Clock Input	Double-click to	clk_0			
		reset	Reset Input	Double-click to	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to	[clk]	0x1000_0800	0x1000_0807	
		irq	Interrupt Sender	Double-click to	[clk]			
<input checked="" type="checkbox"/>		address_span_extender_0	Address Span Extender					
		clock	Clock Input	Double-click to	clk_0			
		reset	Reset Input	Double-click to	[clock]			
		windowed_slave	Avalon Memory Mapped Slave	Double-click to	[clock]	0x0000_0000	0x0fff_ffff	
		expanded_master	Avalon Memory Mapped Master	Double-click to	[clock]			
<input checked="" type="checkbox"/>		hps_0	Arria V/Cyclone V Hard Proce...					
		memory	Conduit	hps_0_ddr				
		hps_io	Conduit	hps_0_io				
		h2f_reset	Reset Output	Double-click to				
		f2h_sdram0_clock	Clock Input	Double-click to	clk_0			
		f2h_sdram0_data	Avalon Memory Mapped Slave	Double-click to	[f2h_sdra...]	0x0000_0000	0xffff_ffff	

FIGURE 5. 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 ☺).