Prerequisites

- Compiling/Cross-compiling environment (Recommended: Cygwin or Codeblocks)
- Initial HW/SW projects (from ORTUS)

Introduction

This lab exposes one of the key characteristics of the heterogeneous FPGA-based SoC development. How should we approach algorithm realization? Which processing paradigm is more feasible for the algorithm implementation? How to approach the development of the system architecture? Most probably it is not possible to resolve these questions fully even in a lifetime, nevertheless, there are some key-guiding principles. In this lab, the challenge is to implement a couple of image manipulation algorithms in software and hardware. To further facilitate your intuition, these implementations shall be benchmarked on different processing architectures and systems - your host system, SoC processing system and FPGA logic.

The code base accompanied with this lab has the following structure:

```
|-- hw
|   --- prj    # Quartus project
|   --- sim    # Simulation testbenches and scripts
|   --- src    # VHDL sources (you will have to modify them)
|
|-- sw_hw      # Software project for interfacing with FPGA
|
|-- sw_img     # Software project for realization of image processing
    --- res    # Image for measurements
    --- src
        --- image.c    # Image processing source (you will have to modify them)
```

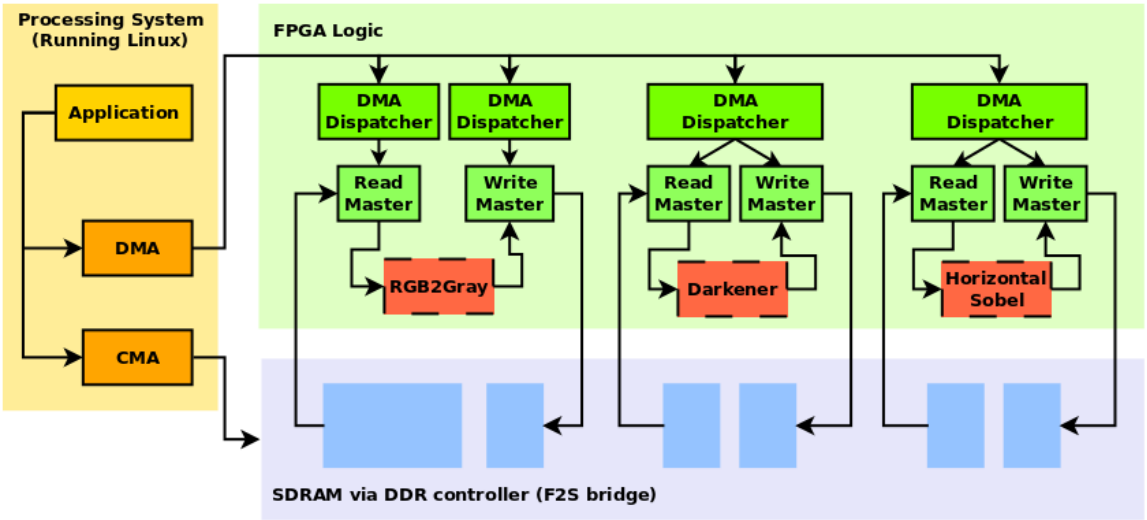

Figure 1. SoC-FPGA system's setup.

The SoC-FPGA system has a structure as shown in Figure 1. The components in this system can be described as follows:

- **Application** - a Linux application responsible for loading/writing images from/to file system and working with FPGA via DMA and CMA drivers.
- **DMA** (Direct Memory Access) - a custom Linux driver for the control of the DMA subsystems, it is used to set up read/write (physical) addresses of the DMA transactions, transaction size (in bytes) and configure/receive interrupts.

- **CMA** (Continuous Memory Allocator) - a custom Linux driver for the management of physically continuous memory, it enables an allocation of relatively large continuous memory buffers (great for images and DMAs) and user-space translation between virtual and physical addresses (for the buffers).
- **DMA Dispatcher** - an FPGA IP core provided by Intel (former Altera) which executes DMA transactions via read/write master IP cores.
- **Read Master** - an FPGA IP core provided by Intel (former Altera) which reads data from the memory bus (Avalon-MM interface) and translates it to (Avalon-ST interface).
- **Write Master** - an FPGA IP core provided by Intel (former Altera) which takes data from streaming interface (Avalon-ST interface) and writes it to the memory (Avalon-MM interface).
- "**Red blocks**" - representation of FPGA computing blocks to be deisgned in this lab.
- "**Blue blocks**" - representation of continuous SDRAM regions allocated by the CMA driver and used by DMA for the transactions.

The LAB.

Software.

This part of the lab is about software implementation of the image manipulation algorithms. You are provided a Makefile project located at ./sw_img directory. The makefile has the following intended commands:

```
make host (default) # compiles executable for your host computer
make device         # compiles executable for the SoC device's processor - Cortex-A9 (armv7a architecture)
make clean          # cleans project (required before switching host/device compilations)
```

Your task for this part of the lab is to finish code in the ./sw_img/src/image.c file for the following procedures:

- image_toGrayscale()
- image_darkener()
- image_sobelHor()
- image_sobelVer()
- image_medianFilter()
- image_averagingFilter()

Hints on solving the challenges are located in the comments, and the lecturer shall present key ideas during the lab (at first, we want you to try and solve these challenges on your own).

Hardware.

This part of the lab is about hardware implementation of the image manipulation algorithms. You are provided with simulation and template files in the ./hw/sim and ./hw/src directories. You have to complete the following design files:

- image_gray.vhd (converts RGB data to grayscale)
- image_darkener.vhd (makes grayscale image darker)
- image_sobelHor.vhd (filters image with simple horizontal edge detector)

These files are located in hw/src directory, use corresponding hw/sim/run_*.do files for simulation. Modelsim simulation can be invoked from terminal with

```
vsim -do run_*.do     # invokes run_*.do script
vsim -do run_*.do -c  # runs Modelsim without GUI (substantially faster)
```

When all simulations are passed, you can compile the Quartus project located in the hw/prj directory. This involves the following steps:

- **Generate source files for the SoC system** - Open platform designer and select soc_system.qsys file. When the system is loaded, press generate. Close the platform designer.
- **Run Analysis & Synthesis**
- **Execute SDRAM pin assignment tcl file** → tools → Tcl scripts... → hps_sdram_p0_pin_assignments.tcl
- **Generate programmable file**
- **Convert programmable file** (from .sof to .rbf)

The program ensuring communication with the FPGA is located at sw_hw directory, but it already should be located in the SD-CARD supplied with the lab.

For further details on programming your SoC turn to your lecturer.

Reporting.

Your report should consist of 3 parts put put into a single zip archive (StudentID_Name_Surname):

- **sw_img** code base,
- **hw/src** code base,
- report document where the following tasks are addressed.

Task 1.

Please fill the following table:

Algorithm	Image	Host PC, μ s	ARMv7, μ s	FPGA, μ s
Conversion to grayscale	lamp.png			
	noisy0.png			
	noisy1.png			
	noisy_lena.png			
Darkener	lamp.png			
	noisy0.png			
	noisy1.png			
	noisy_lena.png			
Horizontal Sobel Filter	lamp.png			
	noisy0.png			
	noisy1.png			
	noisy_lena.png			
Vertical Sobel Filter	lamp.png			**
	noisy0.png			**
	noisy1.png			**
	noisy_lena.png			**
Median Filter	lamp.png			**
	noisy0.png			**
	noisy1.png			**
	noisy_lena.png			**
Averaging Filter	lamp.png			**
	noisy0.png			**
	noisy1.png			**
	noisy_lena.png			**

* The realization of these tasks is optional, although it is encouraged, ** The realization of these tasks is not intended during the lab due to higher effort needed, nevertheless it is encouraged to at least think about the realization of these algorithms.

Task 2.

Please provide the following informaion:

- display input / output files when using the developed sw_img code base,
- display RTL schematics for the designed accelerators (code in **hw/src**).

Task 3.

Please provide answers to the following questions:

1. Why the measured latency for FPGA solutions is so consistent when comparing to the processor-based solutions?
2. What kind of challenges (algorithms) would you recommend the implementation in programmable logic and what kind of algorithms would you recommend for processors?