Lab 2.1

Full System Integration

# Lab 2.0 – Camera acquisition interface (recap)

In lab 2.0 you built 2 pieces of an interface that captures images from a thermal camera and saves the output to a file on your host machine. The pieces you built were:

* Statistics computation unit, which is in charge of computing the minimum, maximum and average pixel values in an image.
* Level adjuster unit, which is responsible for interpolating the frame’s pixel intensities to obtain a much more visible image.

We provided you a full system and you just needed to modify 2 small VHDL files for the thermal camera interface.

# Lab 2.1 – Full system integration

Up until now, we have always employed a bottom-up approach in the labs, i.e. we always provided you with the system, and you were just implementing small pieces of various interfaces. Following this approach, the goal of this lab is to learn how the system is created, i.e. how all components are assembled together.

## Creating a system manually

In [CS-209](http://edu.epfl.ch/coursebook/fr/architecture-des-systems-on-chip-CS-209), you built the full system shown in Figure 1 manually. You did this by implementing each system component (processor, memory, LEDs, buttons, timer …) from scratch, and by connecting them all together through a bus. The problem with this design is that all components are custom-made and cannot be used in another system without significant modifications. Additionally, the design suffers from low performance due to its very simple microarchitecture.

Manually creating systems is great for learning how computer systems work and interact, since one starts from the basics and builds the system incrementally. However, such systems are unsuitable for production environments where high performance is expected and is the norm.

All systems use a standard set of components (processors, memories, timers …), so one could theoretically spend a considerable amount of time to create such components for re-use in various projects. However, the amount of time required to implement and debug such designs would greatly surpass the time available for the project you have been hired for! There must be a way for engineering time to be better spent.

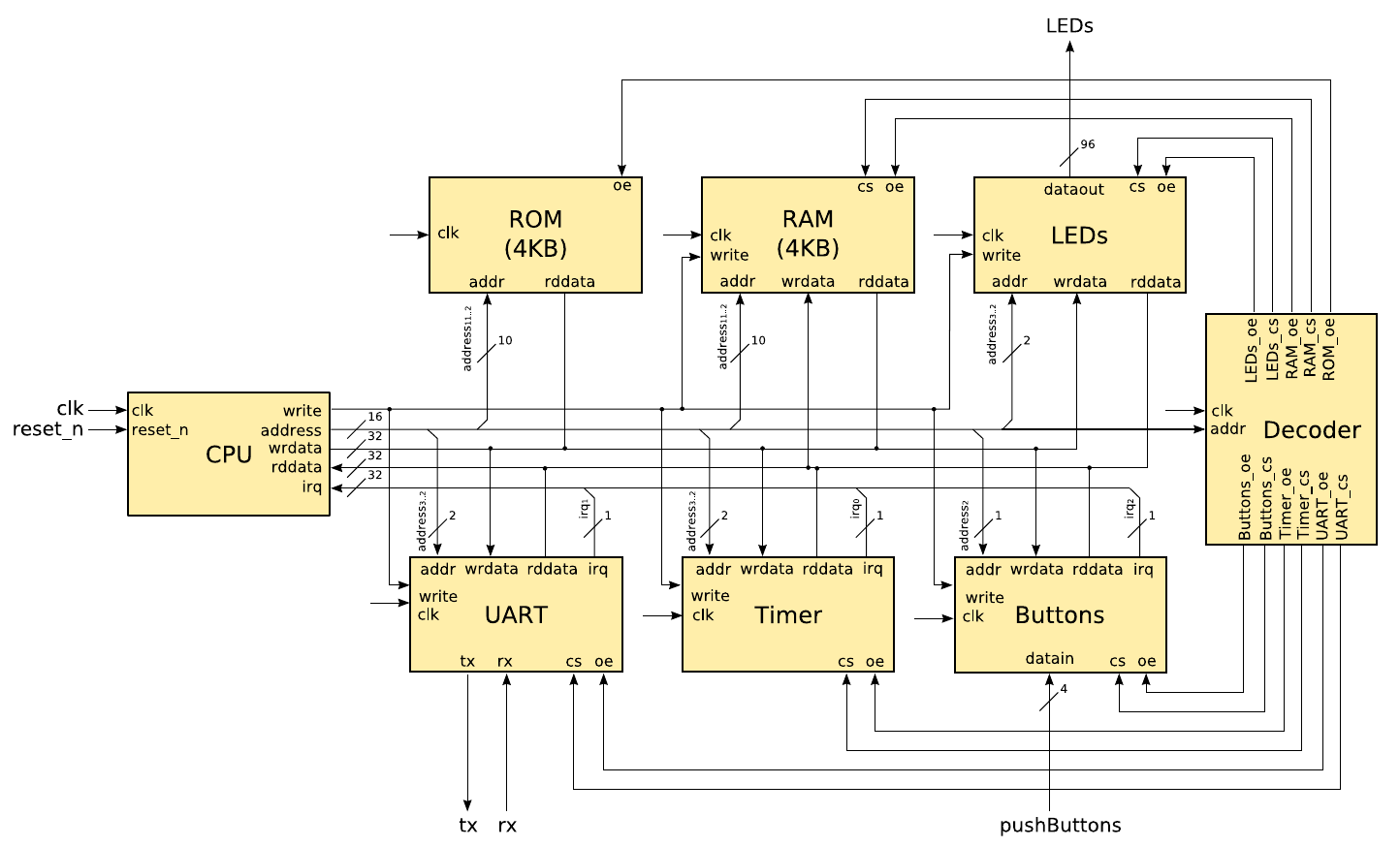


Figure 1. ArchSoC Lab System: Custom-made system components interconnected with a shared bus.

## Creating a system automatically

Solutions exist for the previously described problem, and come in the form of system integration tools. Most FPGA & ASIC design tools have their own system integration tool, but they all essentially expose the same functionality.

A system integration tool provides a library of standard components. Designers can choose any component from the library and use the tool to connect them together. All components don’t forcefully have the same interface, but most tools generally take care of this by inserting adapters to convert between the different bus sizes and handle minor timing adjustments required by each component.

Using such a tool, we can connect standard interfaces together to build up the backbone of the system, and concentrate on implementing custom hardware only for our specialized task.

We will look at one such tool provided by Altera for use in their FPGAs, Qsys.

### The Qsys system integration tool

Using a tool is like speaking a language, as you must first learn it before you can use it. The only way to learn a tool is to follow the same method all serious engineers have previously used. It consists of an ancestral technique that can be summarized by 1 acronym: RTFM.

If you are not familiar with this acronym, we highly suggest you look it up on the internet and add it to your skillset. The world is not a kind place, and you will see that RTFM is unfortunately your only friend when you find a job and are all alone in front of your computer.

As such, we are going to train you to become serious engineers. Let’s start!

#### Learning Qsys

Download the [Quartus Prime Standard Edition Handbook](http://moodle.epfl.ch/mod/resource/view.php?id=918492) and read the chapters relevant to Qsys. For this first introduction to Qsys, it is enough to only read “Chapter 5: Creating a System With Qsys” and “Chapter 6: Creating Qsys Components”.

You don’t need to read the full chapters, but reading through the following sections gives you the big picture of what Qsys can do.

* Volume 1 – Chapter 5: Creating a System With Qsys (pg 179)
  + Interface Support in Qsys (pg 180)
  + Adding IP Cores to the IP Catalog (pg 181)
  + Set up the IP Index File (.ipx) to Search for IP Components (pg 184)
  + Create a Qsys System (pg 185 – 208)
  + Integrate a Qsys System and the Quartus Prime Software With the .qsys File (pg 233)
  + View the Qsys HDL Example (pg 251)
* Volume 1 – Chapter 6: Creating Qsys Components (pg 362)
  + Qsys Components (pg 362)
  + Create IP Components in the Qsys Component Editor (pg 366 – 368)
  + Specify IP Component Type Information (pg 368)
  + Specify HDL Files for Synthesis in the Qsys Component Editor (pg 373)
  + Analyze Synthesis Files in the Qsys Component Editor (pg 374)
  + Add Signals and Interfaces in the Qsys Component Editor (pg 378)

#### Using Qsys

1. Open the lab template project in Quartus Prime. Observe that the only files included in the project are the top-level VHDL file and a clock constraint file (\*.sdc).
2. Copy your implementations for the various components designed in the previous labs to the appropriate areas in the “hw/hdl” directory.
3. Launch Qsys.
4. Open the “hw/quartus/soc\_system.qsys” file. This file contains the Qsys system design of Lab 2.0, but where all custom peripherals you developed are missing (PWM, MCP3204, Lepton). The partial design in show in Figure 2 and consists simply of a Nios II processor, an on-chip memory, and a JTAG UART (for printf support).

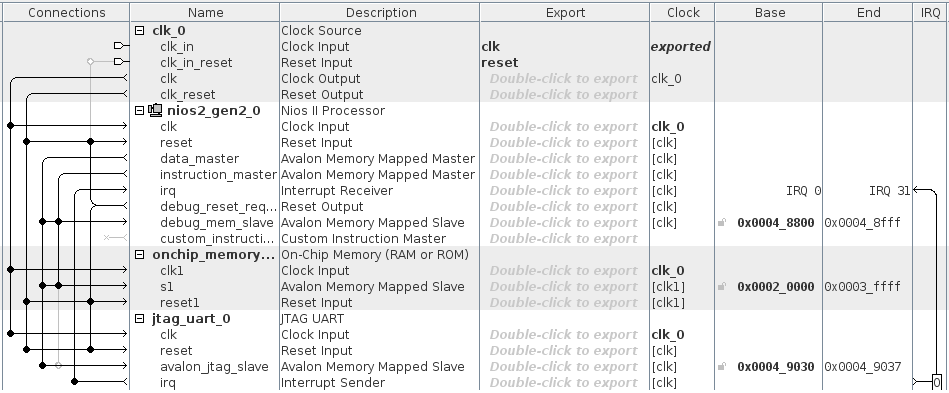


Figure . Partial Qsys System

1. Create the Qsys IP component for the PWM interface you used in Lab 1.1.
2. Create the Qsys IP component for the MCP3204 interface you used in Lab 1.2.
3. Create the Qsys IP component for the lepton thermal camera interface that you used in Lab 2.0. Unlike the PWM and MCP3204 interfaces, the Lepton requires you to set the number of read wait states to 9, as shown in Figure 3. This is due to the speed of the division which occurs over multiple clock cycles, so a CPU cannot read from the Lepton interface as fast as it can from the PWM and MCP3204 interfaces.



Figure 3. Qsys Lepton IP Core Timings

1. Use the IP cores available in the IP Catalog to create the full system shown in Figure 4 which you used in Lab 2.0 to capture images from the thermal camera.

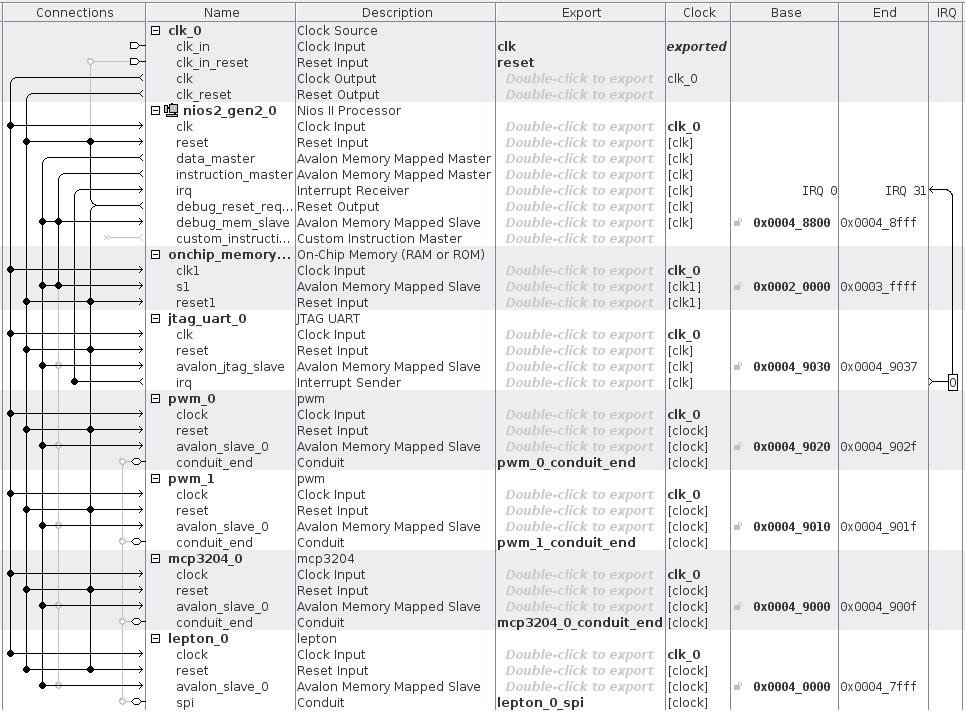


Figure 4. Lab 2.1 System to assemble

1. Do what is required to correctly instantiate your Qsys system in the top-level design file located at “hw/hdl/DE0\_Nano\_SoC\_PrSoC\_extn\_board\_top\_level.vhd”.
2. Create a Nios II SBT project for your new system and test if everything works like in Lab 2.0. By everything works, we mean that you can successfully capture a thermal image with the lepton and save it to a file on your host PC.

If you are stuck, remember RTFM. We stress that all the information you need to successfully implement points 3, 4, 5, and 6 listed above are fully described in the specified pages of the [Quartus Prime Standard Edition Handbook](http://moodle.epfl.ch/mod/resource/view.php?id=918492)!

Don’t hesitate to ask us any questions if something is unclear. It is essential you understand how all the components in the system are assembled and how they interact.