

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.

Creating a system manually

In [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.*

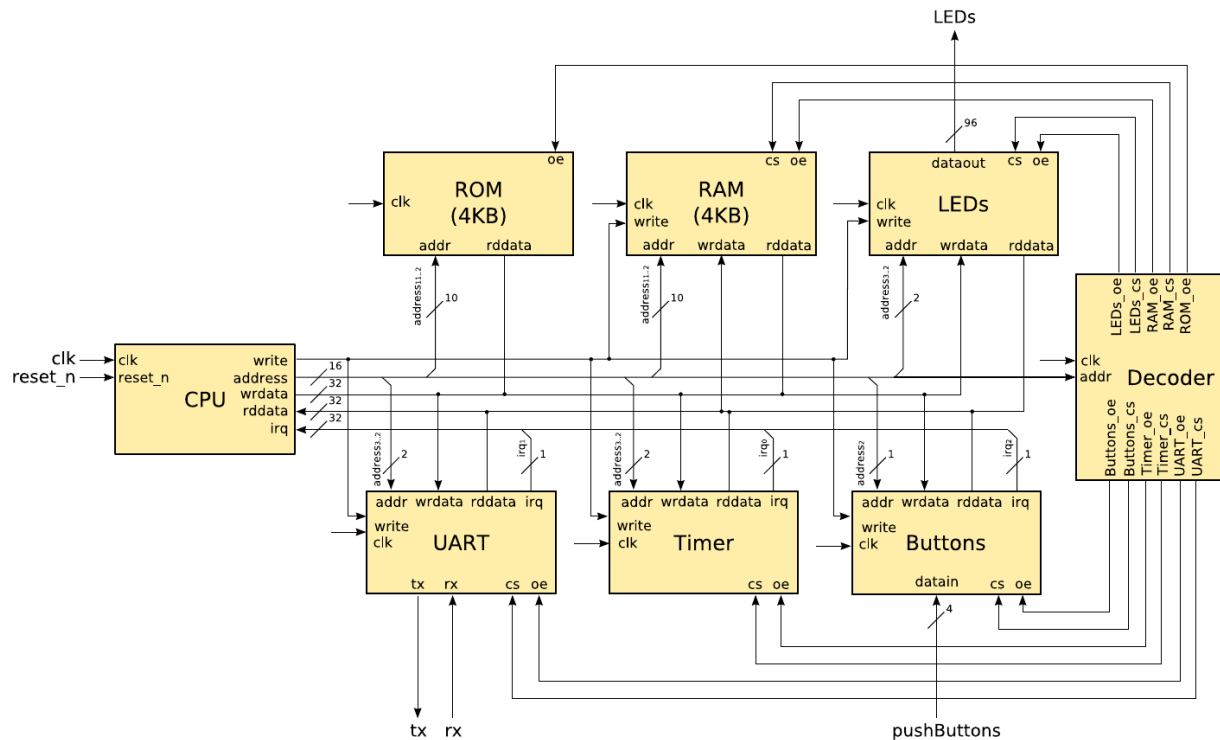


FIGURE 1. ARCHSOC SYSTEM

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 (QSYS) provided by Altera for use in their FPGAs.

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**.

Seriously, 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!

Your Job

Learning Qsys

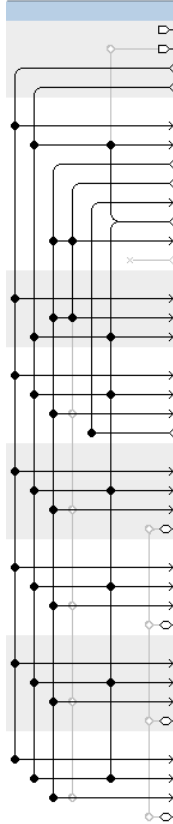
Download the **QUARTUS PRIME STANDARD EDITION HANDBOOK** and read the chapters relevant to QSYS. For this first introduction to QSYS, it is enough to only read "CHAPTER 5: CREATING A SYSTEM WITH QSYS". You don't need to read the whole chapter, but reading through the following sections gives you the 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)

Using Qsys

1. Open the lab template project in **QUARTUS PRIME**. Observe that the only file included in the project is the top-level design file.
2. Copy your implementations for the various components designed in the previous labs to the appropriate areas in the "lab2.1/hw/hd1" directory.
3. Launch Qsys.
4. Create the full system used in Lab 2.0 to capture images from the thermal camera. The system is shown in Figure 2 for your convenience.
5. Do what is required to correctly instantiate your QSYS system in the top-level design file located at "lab2.1/hw/hd1/DE0_Nano_SoC_top_level.vhd".
6. 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, and 5 listed above are fully described in the specified pages of the **QUARTUS PRIME STANDARD EDITION HANDBOOK**!



Connections	Name	Description	Export	Clock	Base	End	IRQ
	clk_0	Clock Source					
	clk_in	Clock Input	clk	exported			
	clk_in_reset	Reset Input	reset				
	clk	Clock Output	<i>Double-click to</i>	clk_0			
	clk_reset	Reset Output	<i>Double-click to</i>				
	nios2_gen2_0	Nios II Processor					
	clk	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clk]			
	data_master	Avalon Memory Mapped Master	<i>Double-click to</i>	[clk]			
	instruction_master	Avalon Memory Mapped Master	<i>Double-click to</i>	[clk]			
	irq	Interrupt Receiver	<i>Double-click to</i>	[clk]			
	debug_reset_req...	Reset Output	<i>Double-click to</i>	[clk]			
	debug_mem_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]			
	custom_instructi...	Custom Instruction Master	<i>Double-click to</i>	[clk]			
	onchip_memory...	On-Chip Memory (RAM or ROM)					
	clk1	Clock Input	<i>Double-click to</i>	clk_0			
	s1	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk1]			
	reset1	Reset Input	<i>Double-click to</i>	[clk1]			
	jtag_uart_0	JTAG UART					
	clk	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clk]			
	avalon_jtag_slave	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clk]			
	irq	Interrupt Sender	<i>Double-click to</i>	[clk]			
	pwm_0	pwm					
	clock	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clock]			
	conduit_end	Conduit	<i>Double-click to</i>	pwm_0_conduit_end			
	pwm_1	pwm					
	clock	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clock]			
	conduit_end	Conduit	<i>Double-click to</i>	pwm_1_conduit_end			
	mcp3204_0	mcp3204					
	clock	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clock]			
	conduit_end	Conduit	<i>Double-click to</i>	mcp3204_0_condui...			
	lepton_0	lepton					
	clock	Clock Input	<i>Double-click to</i>	clk_0			
	reset	Reset Input	<i>Double-click to</i>	[clock]			
	avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to</i>	[clock]			
	spi	Conduit	<i>Double-click to</i>	lepton_0_spi			

FIGURE 2. LAB 2.0 SYSTEM