

# CSEE 4840

## Embedded System Design

### Lab 3: Peripherals and Device Drivers

Stephen A. Edwards  
Columbia University

Spring 2019

Implement on the FPGA a memory-mapped peripheral that can receive communication from the ARM processors on the Cyclone V. Communicate with your peripheral through a Linux userspace program that accesses a device driver you have written.

Your peripheral should display a ball on the VGA screen at coordinates given to it through software. Your device driver should implement an *ioctl* that takes coordinates from the user and sends it to your peripheral

#### 1 Introduction

In this lab, you will control your own hardware from your own software, communicating through a Linux device driver. We supply a base hardware design to extend and a working example of a VGA peripheral you will have to modify, and a working device driver for the existing peripheral that you will have to adapt to work with your own peripheral.

You will implement a video bouncing ball in this setting. Your peripheral will generate an VGA raster consisting of a ball at a particular location, your userspace C program (software) will make this ball bounce around the screen, and your device driver will mediate between your program and your peripheral.

#### 2 Compile the VGA Component Into a New FPGA Image

In this section, you will tell Qsys about a new peripheral component, connect it ultimately to the ARM processors, and synthesize a new FPGA configuration bitstream.

##### 2.1 Create the VGA Ball Component

Download *lab3-hw.tar.gz* from the class website and unpack it on your workstation. Change to the *lab3-hw* directory. Run *qsys-edit soc\_system.qsys*, which will bring up a GUI.

Create a new *VGA\_BALL* component and connect it to the base design. Select File→New Component. This should open the Component Editor window.

In the *Component Type* tab, set Name to *vga\_ball* and Display Name to *VGA Ball*.

In the *Files* tab, click *Add File...* under *Synthesis Files* and select the *vga\_ball.sv* file. Click on *Analyze Synthesis Files*. This should quickly complete successfully; close the pop-up window. Set *Top-Level Module* to *vga\_ball*. Some warnings and errors should appear in the *Messages* tab; we will fix them.

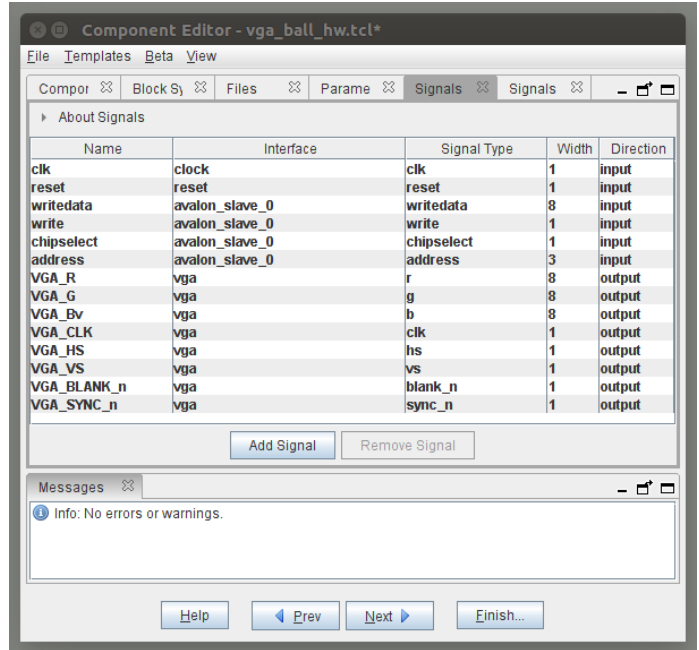
## 2.2 Assign the Interface Signals on the vGA Ball Component

When Qsys analyzes the synthesis files, it makes some good guesses about the meaning of each signal on the peripheral, but it is not perfect. Below, you will fix these mistakes.

Click on the *Signals & Interfaces* tab. Click on *avalon\_slave\_0* and set its *Associated Reset* to *reset*. Click on *<<add interface>>* in the left box and select *Conduit*. Set the name of the new conduit to *vga*.

Select *View*→*Signals* from the top menu. This should bring up a new *Signals* tab.

Change the *Interface* column of each of the *VGA\_* signals to *vga*. Change the *Signal Type* of each signal to a lower-case version of the name after the *VGA\_*, e.g., *VGA\_BLANK\_n* should become *blank\_n*.



Your signals show now appear like the list on the right. If it complains about *avalon\_slave\_0* not having an associated reset, click on *Signals & Interfaces*, click on *avalon\_slave\_0*, and set *Associated Reset* to *reset*.

Once you have eliminated all errors, click on *Finish*. It will warn you that it is saving *vga\_ball\_hw.tcl*; click on *Yes, Save*. The Component Editor window should close.

Open *vga\_ball\_hw.tcl* with a text editor and add the following three lines after the *module vga\_ball* section:

```
set_module_assignment embeddedsw.dts.vendor "csee4840"
set_module_assignment embeddedsw.dts.name "vga_ball"
set_module_assignment embeddedsw.dts.group "vga"
```

These make the device show up as compatible with *csee4840,vga\_ball-1.0* in the *.dtb* file, which we will discuss below.



connections within the instance of *soc\_system* near the end of the file:

```
.vga_r (VGA_R),  
.vga_g (VGA_G),  
.vga_b (VGA_B),  
.vga_clk (VGA_CLK),  
.vga_hs (VGA_HS),  
.vga_vs (VGA_VS),  
.vga_blank_n (VGA_BLANK_N),  
.vga_sync_n (VGA_SYNC_N)
```

Delete the two *assign* statements to the various VGA signals at the bottom of *soc\_system\_top.sv*.

## 2.5 Compile the Hardware Design with Quartus

Either run *make quartus* or open the *soc\_system* project in Quartus to compile the Qsys-generated system (whose source is in the *soc\_system* subdirectory) to generate the “SRAM object file” *output\_files/soc\_system.sof*.

After Quartus compilation, convert the .sof file to an .rbf file by running *make rbf*.

Copy the *output\_files/soc\_system.rbf* into the boot partition of your SD card. You can mount your SD card on your workstation and copy the file. Alternatively, mount the boot partition by running *mount /dev/mmcbk1op1 /mnt* on your DE1-SoC then use *scp* to copy the file from your workstation to your board, e.g.,

```
scp sedwards@micro11.ee.columbia.edu:lab3/soc_system.rbf /mnt
```

Make sure the file has actually been written out to the card: type *sync* at the command-line.

## 3 Tell the Linux Kernel About Your Peripheral

The Linux kernel employs a persistent data structure known as the Device Tree to describe the structure of a hardware platform. It contains information about processors, memory regions, bus bridges, and most importantly, the types and memory location of peripherals such as the VGA Ball. Qsys generates a similar *soc\_system.sopcinfo* file that, in concert with the *soc\_system\_board\_info.xml* file, can be used to generate an appropriate *soc\_system.dtb* file, a binary representation of the Device Tree that is normally loaded as part of the boot process.

Generate *soc\_system.dtb* by running *make dtb*. This needs *sopc2dts* to be in your path, which you can add by running *embedded\_command\_shell.sh* (on the lab computers, this is an alias; just type it at the command line).

Verify that the VGA Ball peripheral appears in the *soc\_system.dts* file, which should include

```
vga_ball_0: vga@0x100000000 {  
    compatible = "csee4840,vga_ball-1.0";  
    reg = <0x00000001 0x00000000 0x00000008>;  
    clocks = <&clk_0>;  
}; //end vga@0x100000000 (vga_ball_0)
```

The entry itself comes from the `vga_ball_o` module instance in Qsys (`soc_system.qsys`). The `compatible` string is controlled by the `set_module_assignment` statements you should have added to the `vga_ball_hw.tcl` file.

As for the `.rbf` file, copy the `soc_system.dtb` file to the boot partition on your SD card.

## 4 Communicate with Your Peripheral Through Software

Connect a VGA monitor and a USB keyboard to your board. Boot Linux on your board from the SD card with your new `soc_system.rbf` and `soc_system.dtb` files (your SD card from lab2 is otherwise fine). If your board is already powered on, restart it by typing `reboot` (don't just power-cycle it).

Connect the console port (mini-USB cable) to your workstation and run `screen /dev/ttyUSB0 115200` as you did in lab 2.

Boot Linux on your board. It should go through the normal boot process and you should see a white box against a colored background on the VGA monitor.

Verify that the kernel sees the VGA Ball device in the device tree:

```
# ls "/proc/device-tree/sopc@0/bridge@0xc0000000/"
#address-cells  clock-names  compatible  ranges      reg-names
#size-cells    clocks        name        reg         vga@0x10000000
# more "/proc/device-tree/sopc@0/bridge@0xc0000000/vga@0x10000000/compatible"
csee4840,vga_ball-1.0
```

### 4.1 Compile and Run the Sample Program

On your board, download and install `linux-headers-4.19.0.tar.gz`, which includes the `Makefile` for compiling kernel modules.

```
# wget http://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/linux-headers-4.19.0.tar.gz
# tar Pzxf linux-headers-4.19.0.tar.gz
# ls /usr/src/linux-headers-4.19.0
Documentation  arch      drivers  init     mm              scripts  usr
Kconfig        block   firmware ipc       modules.order  security  virt
Makefile       certs   fs        kernel   net             sound
Module.symvers crypto  include  lib       samples        tools
```

Install the kernel module management programs (e.g., `insmod`, `rmmod`).

```
# apt install -y kmod
```

Download `lab3-sw.tar.gz` from the class website to your board, unpack it, compile it, install the kernel module.

```
# wget http://www.cs.columbia.edu/~sedwards/classes/2019/4840-spring/lab3-sw.tar.gz
# tar zxf lab3-sw.tar.gz
# cd lab3-sw
```

Compile the device driver and user program, install the kernel module, and verify that it works. This should look like

```
# make
make -C /usr/src/linux-headers-4.19.0 SUBDIRS=/root/lab3 modules
make[1]: Entering directory '/usr/src/linux-headers-4.19.0'
CC [M] /root/lab3/vga_ball.o
Building modules, stage 2.
MODPOST 1 modules
CC /root/lab3/vga_ball.mod.o
LD [M] /root/lab3/vga_ball.ko
make[1]: Leaving directory '/usr/src/linux-headers-4.19.0'
cc hello.c -o hello
# insmod vga_ball.ko
# lsmod
Module                Size  Used by
vga_ball              16384  0
# ./hello
VGA ball Userspace program started
initial state: f9 e4 b7
ff 00 00
00 ff 00
00 00 ff
ff ff 00
...
ff 00 ff
VGA BALL Userspace program terminating
# rmmod vga_ball
rmmod: ERROR: ../libkmod/libkmod.c:514 lookup_builtin_file() could not open
builtin file '/lib/modules/4.19.0/modules.builtin.bin'
```

You may ignore the error from *rmmod*.

“make” compiles the kernel module (*vga\_ball.ko*) and the userspace program (*hello*).

“insmod” loads the generated kernel module. In the supplied device driver, doing this should change the display. “lsmod” lists installed modules.

The *hello* program is a userspace program that communicates with the *vga\_ball* device driver through the *ioctl* system call. It opens the device and reads and writes its state, which changes the color of the background.

“rmmod” removes the kernel module, which is necessary any time you modify and re-compile the module.

## 5 What to Do

Modify the hardware and software in the skeleton you have been provided to display a bouncing ball. Change both the interface and contents of the hardware peripheral so that it displays a stationary ball at a software-controllable set of coordinates. Have your peripheral respond to writes to one or more addresses that control the location of the ball.

Adapt the provided device driver to communicate with your peripheral. E.g., create an *ioctl* that sets the coordinates of the ball.

Write a userspace program that bounces the ball by repeatedly communicating the new coordinates to your peripheral through your device driver.

## 6 What to turn in

Find an overworked TA or instructor, and show him/her your bouncing ball. Once s/he is satisfied, collect just the files *you wrote or modified* for this lab in a directory called “lab3,” make a tarball with `tar zcf lab3.tar.gz lab3`, and submit that via Courseworks. This should include the SystemVerilog for your peripheral and source for your device driver and userspace program.

*Do not submit everything in your lab3-hw directory: it is too big.*

## 7 Qsys Hints

### 7.1 Editing the Source of Your Qsys Component

If you modify the SystemVerilog for your hardware component **without changing its interface**, regenerate your system with Qsys then re-run Quartus. Do this by running `make qsys-clean ; make qsys` or open Qsys from Quartus (Tools→Qsys) and click on *Generate HDL...*

If you **modify the interface** your hardware component (e.g., to change the number of visible registers, add a read function, or change the signals passed through the conduit), edit the component. Start Qsys (e.g., run `qsys-edit`), open your `.qsys` file, select your component under “Project,” and click “Edit.” This should bring up the Component Editor window.

Re-analyze the synthesis files as you did in Section 2.1, make sure the interface signals are assigned correctly, and click *Finish*.

Every time you update the component, re-insert the `set_module_assignment` directives mentioned in Section 2.2.

In Qsys, select File→Refresh System (or just press F5). It should complete with a reassuring warning indicating the version of your component has changed. Hovering over the instance of your component should also indicate its version has changed. Save your project after doing this to update the `.qsys` file.

Now, select Generate→Generate... to instruct Qsys to regenerate your system so Quartus can recompile it. Alternately, run `make qsys-clean ; make qsys`, which does the same thing from the command line.

### 7.2 Don't Edit Copies

Do not edit the files in the `synthesis` directory (e.g., in `lab3-hw/synthesis/submodules`). These are copied by Qsys and will be overwritten the next time Qsys runs.

### 7.3 Viewing Components as Blocks

Select a component and then View→Block Symbol. This shows how Qsys interprets the interface to a component.

