Lab 4: Audio Pipeline on the FPGA

6.375 Laboratory 4
Assigned: February 27, 2010
Due: March 5, 2010

1 Introduction

In this lab we will run the audio pipeline constructed in the previous labs on an FPGA. We introduce Sce-Mi as means for communicating between the host processor and the FPGA and show how Sce-Mi integrates with the Bluespec Workstation, giving us a way to simulate the processor-FPGA link. We will then synthesize the audio pipeline for the FPGA, look at area and timing results, and, finally, we will run the audio pipeline on an actual FPGA.

1.1 Getting Started

Lab 4 uses the audio pipeline from lab 3 with the pitch modulation. We need additional infrastructure to use Sce-Mi for communication between the host computer and FPGA. This infrastructure is provided in the lab4 harness.

1. Extract the code from the lab4 harness and add it to your subversion repository. Just as we did for the previous labs, you will need to add the 6.375 course locker and source the `setup.csh` script. A new `setup.sh` script is also available for `sh` users to use if they prefer. Navigate to the directory which contains `lab1`, `lab2`, and `lab3` folders and run

   ```
   % tar -xzvf /mit/6.375/lab-harnesses/lab4-harness.tar.gz
   % svn add lab4
   ```

   This will create a directory called `lab4` with some new code in the `scemi/` directory and Sce-Mi projects setup in `sim/` for simulation and `fpga/` for running on the FPGA.

2. Copy your FIR filter, FFT, and pitch transform from lab 3 and add them to subversion.

   ```
   % cd lab4
   lab4 % cp ../lab3/fir/FIRFilter.bsv fir/FIRFilter.bsv
   lab4 % cp ../lab3/fft/FFT.bsv fft/FFT.bsv
   lab4 % cp ../lab3/pitch/Transform.bsv pitch/Transform.bsv
   lab4 % svn add {fir,fft,pitch}/*.bsv
   ```

3. Check the code into subversion

   ```
   lab4 % svn ci . -m "Lab4 Initial Checkin'
   ```

2 Sce-Mi

The Standard Co-Emulation Modeling Interface (Sce-Mi) is an Accellera standard which was designed to aid in verification of hardware designs. The standard specifies a transaction-based modeling interface used to pass messages between an un-timed software test bench and a design under test (DUT) described in register transfer language (RTL). The DUT can be emulated on an FPGA to achieve better performance than software RTL simulators are capable of.

Bluespec provides a complete implementation of the Sce-Mi standard, which makes Sce-Mi attractive as a means for us to interact with the designs we run on FPGAs.

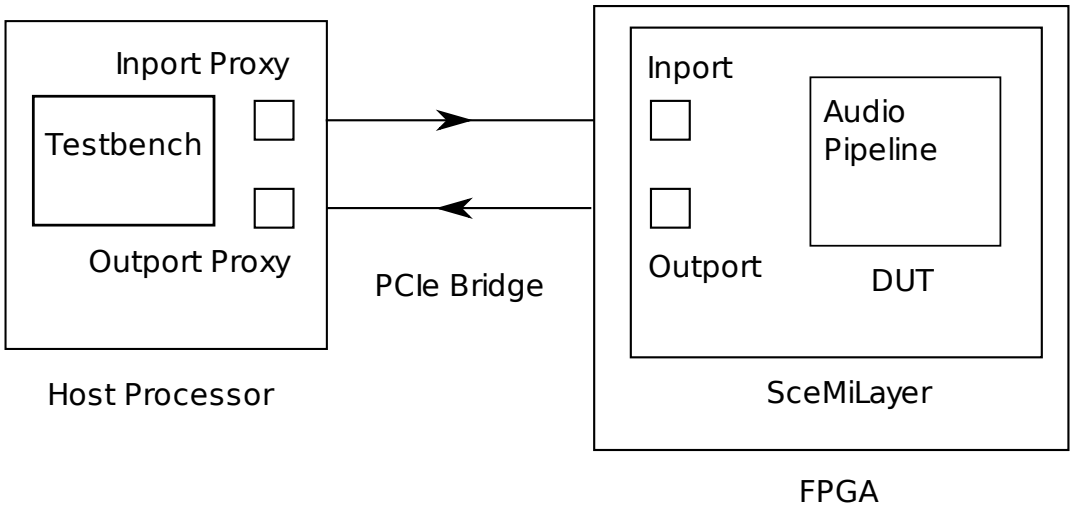

Figure 1: Audio Pipeline with Sce-Mi

Figure 1 shows how the audio pipeline looks using Sce-Mi to pass the samples from the host processor to the FPGA and back. The `mkAudioPipeline` module becomes our design under test (DUT). The SceMiLayer is a wrapper around our DUT which hooks it up to the Sce-Mi ports `inport` and `outport`. Samples will arrive on `inport` and be passed directly to the audio pipeline. Samples coming out of the audio pipeline will go to `outport`.

We use PCI Express as our bridge from processor to FPGA. On the host processor we have port proxies, corresponding to the Sce-Mi ports on the FPGA, which the software test bench can interact with. The test bench is implemented in c++ and runs on the host processor.

The net effect is an unchanged audio pipeline which still accepts a sample at a time and outputs samples, and a new test bench written in c++ running on the host processor which can send and receive samples through the Sce-Mi port proxies to the audio pipeline on the FPGA.

The `scemi/` directory contains three files.

`scemi/SceMiLayer.bsv` implements the wrapper around `mkAudioPipeline`. It instantiates `inport` and `output` and hooks them up to `mkAudioPipeline`.

`scemi/Bridge.bsv` selects from among a number of bridges supplied by Bluespec. Each bridge is specific to the FPGA we are running on. We will use the TCP bridge for simulating the hardware with Sce-Mi and the XUPV5 bridge when running on the FPGA.

`scemi/Tb.cpp` implements the software test bench. It has the same functionality as the test bench we have been using so far. It reads `in.pcm`, sends the samples through the audio pipeline and records the transformed samples to `out.pcm`.

3 Simulating with Sce-Mi

We introduced Sce-Mi so we can run our design on an FPGA instead of just simulating it in software, but it is still useful to simulate the design using the same Sce-Mi infrastructure. This allows us to see the output of `$display` calls in our design and lets us avoid the long synthesis times needed to synthesize the design for the FPGA when we are still debugging the functionality of the design.

To use Sce-Mi we have pulled our system into two distinct pieces, the test bench and the audio pipeline. The only interaction between the test bench and audio pipeline is through the bridge. In simulation we will still have those two distinct pieces, only now the audio pipeline runs in a separate Bluesim process on our computer from the test bench, and the bridge between those two processes is TCP instead of PCI Express.

Building a project with Sce-Mi is a little more complicated than the simple projects we have been using up to this point. The test bench is compiled separately from the audio pipeline because the test bench is now implemented in software. We also produce extra information for the test bench to communicate with the audio pipeline, including things such as what bridge to use and what the Sce-Mi port names are.

Bluespec has introduced a build process which knows how to build everything for Sce-Mi and is specifically designed to make builds repeatable. We have set up a Workstation project which uses this build process.

Navigate to the `sim/` directory. The file `project.bld` describes what we want Bluespec's build process to build for us.

1. Copy over our test input to the local directory.

   ```
   sim% cp ../data/foo.pcm in.pcm
   ```

2. Open up the `sim.bspec` workstation project in `bluespec`.

   ```
   sim% bluespec sim.bspec
   ```

3. Compile, link and simulate the design. This will build two programs, a simulator for the audio pipeline called `bsim_dut`, and the software test bench `tb`. When the simulate command is run in the Workstation, the Workstation first starts up the `bsim_dut` program, giving it a little time to get ready, then runs `tb`. The two programs communicate over TCP.

4. Verify the output from simulation is correct.

   ```
   sim% cmp out.pcm ../data/foo_pitch8.pcm
   ```

3.1 Fixing FIR Filter Reset

It will be convenient to rerun the test bench multiple times without having to reprogram the FPGA. To get the expected results every time, this requires us to properly reset the state in our design whenever end of file is reached. For our design this means fixing the FIR filter to reset the sample registers to all zeros when it reaches end of file.

The test bench has been augmented to run twice if you specify the `--repeat` flag. You can use this to test whether your design resets appropriately when it gets the end of file.

Problem 1: Fix your FIR filter to reset when the end of file is reached.

1. Modify your fir filter to reset the sample registers to 0 whenever end of file is reached.

2. Verify your design works when run repeatedly by running the test bench with the `--repeat` flag. This can be accomplished by manually running the `tb --repeat` in a terminal after starting `bsim_dut`, or you can set the Workstation to use the `--repeat` flag when you run simulation by going to `Project->Options`, the `SCE-MI` tab at the bottom in the `Simulate command` field.

   Compare the output `out.pcm` against the expected output `../data/foo_pitch8.fft` and verify they match.

4 Synthesizing for the FPGA

In the `fpga` directory we have setup a Workstation project for synthesizing and running the audio pipeline on an FPGA. Open the project in `bluespec`, compile and link it, but do not "simulate".

Compiling and linking the audio pipeline calls the Bluespec compiler to generate Verilog code. It then uses the Xilinx tools to map and place-and-route the design for our FPGAs. This takes somewhere around an hour to complete.

The Xilinx tools output reports to the `xilinx/` directory from which we can learn the area and critical path of our design. The file `xilinx/mkBridge.srp` contains a summary with information about how many slices our design takes up and the critical paths in our design. The clock for our design is `scemi_clk_port_clkgen/current_clk`. This information should be enough for you to answer the first two discussion questions.

We can get additional information about which parts of our design take which resources in the file `mkBridge_map.mrp`, which shows resource utilization by hierarchy.

5 Running on the FPGA

Now that you have synthesized your design for the FPGA, all that remains is to run it!

1. Log into your assigned machine, source the class setup script, and navigate to the lab 4 `fpga/` directory.
2. Run the `program_fpga` command to program the board.

```
lab4/fpga% program_fpga
```

3. Run your test bench.

```
lab4/fpga% cp ../data/foo.pcm in.pcm
lab4/fpga% ./tb
```

4. Verify the output is correct.

```
lab4/fpga% cmp out.pcm ../data/foo_pitch8.pcm
```

Because we fixed the FIR filter to reset at the end of file, you should be able to rerun the test bench repeatedly without reprogramming the board and still get the correct results.

6 Discussion Questions

1. What is the number of slice registers and number of slice LUTs used in your design as reported in `mkBridge.srp`? What percentage of the total slice resource available on the board do these account for?
2. What is the clock period for the `scemi_clk_port_clkgen/current_clk` reported in `mkBridge.srp`? Can you tell from the critical path reported where in your design the critical path is?
3. Were you able to successfully run your design on the FPGA? What problems did you encounter if any?

7 What to Turn In

When you have completed the lab you should check in a final version via subversion. This should include your modifications to `fir/FIRFilter.bsv` and a file `answers` in the top level lab directory with your answers to the discussion questions. For example

```
lab4% svn add answers.pdf
lab4% svn ci -m "Lab 4 final submission"
```