EECS578 – Prof. Bertacco
Fall 2015

EECS 578 – SVA mini-project
Assigned: 10/08/15 – Due: 10/27/15

1. Overview

This project focuses on designing a test plan and a set of test programs for a digital reverberation module, using a hardware verification language. Digital reverberators are digital sound processors used to produce surround effects. The effect is produced by delaying the sound input channels by different amounts of time with the objective of creating the impression that the sources of sound come from different locations.
If you would like some more information on reverberation and surround effects, check:
https://en.wikipedia.org/wiki/Reverberation

Before dwelling in the verification of the reverberator, you will first write a test program for a very simple arbiter design, which will serve the purpose of making you at ease with the use of HVL testbenches in the verification flow.

This document describes the assignment with reference to SystemVerilog Assertions (SVA) and the VCS logic simulator. We provide manuals and a tutorial on SVA in the class webpage.

2. How to run VCS

In order to run VCS, connect to a Linux machine in any of the CAEN student labs or SSH to the host oncampus-course.engin.umich.edu. Note that this host is accessible only on campus.

3. Bus arbiter warm-up

To get started with this part of the project, download the file `arbiter.tar.gz` from the class website and copy it to a local directory in your *engin* account. Unzip the file; it will create the `arbiter` directory.
If you go to the `arbiter` directory, you will find a sample bus arbiter design (unfortunately it includes one bug). The arbiter design takes requests from two clients and grants access to only one client at a time. It should be fair (*i.e.*, if both clients keep sending requests, they both should receive grants sooner or later) and it should only give a grant to a client after it has received a request from it. The grant signal should be high for 1 clock cycle. The arbiter design is described in Verilog.
You can start writing your testbench in SystemVerilog by filling the FIXME section in `test_arbiter.sv`, and you can also modify other sections of the code. We provide a simple `Makefile` for you. You can compile your source code and run the simulation by running the following command.

```
make run
```

If you just want to compile your source code, then use the following command.

```
make
```

Or you can start DVE (a graphical waveform viewer) by running the following command.

```
make dve
```

Your assignment for this part is to write a test that verifies the main specifications of the arbiter: granting the bus only upon request, to the correct requestor, all while respecting mutual exclusion.

You need to submit:

1) The SystemVerilog testbench that you developed, called `test_arbiter.sv`. Please add comments to the testbench, explaining for each portion which feature of the arbiter you are trying to stimulate and verify.
2) The corrected design of `arbiter.v`.

4. Verification of a digital reverberation module

The main part of your work consists of verifying a digital audio reverberation module. A digital reverberation module is a multi-channel digital audio transfer unit that can add a programmable amount of delay to the input signals. Digital reverberation modules are mainly used for digital surround effects. You can find the specification of the module in the last part of this document.
We provide the design (containing some bugs) in Verilog, a testbench template in SystemVerilog, a `Makefile` and a directory structure, which you can find in the file `DAR.tar.gz`. Please download and unzip the tar file, and read the `README` file provided in the `DAR` directory. Your job is to come up with a thorough verification plan and a set of testbenches that cover each of the items in your plan.

4.1. The verification plan (1 page)

After studying the specification and the design, you should write a simple verification plan. Write a list of verification goals that you think should be checked for this design. Note: you should be able to develop most of the plan by simply studying the specification, without needing to study the design itself. Your verification plan should:

1) Include at least 5 verification goals.
2) Include any assumptions you have made in trying to understand all the features of this design. For instance: "we made no assumptions on the nature of the PCM audio sample."
3) Be no longer than 1 page.

4.2. The test suite design

Create a full testbench written in SystemVerilog. The test must target the goals of the verification plan and use the major verification features/constructs with the objective to generate a compact, modular set of tests. Again, make (and document) all the assumptions you need in completing this task: feel free to update your verification plan based on this part of the work.

Try to organize your testbench in sections, for each section verify one aspect of the design, and then use print statements (`$display` in SystemVerilog) indicating if that part of the test passed or failed. You may use tasks and functions to keep your testbench organized.

4.3. The verification report (1 page)

Write a final report on your verification job, including:

1) A section on the bugs you found, which test exposed them, or how you came across them otherwise. If possible, suggest the action required to correct or work around them.
2) Instructions so that we can run the tests you prepared.
3) The report should be 1 page or less in length.

You need to submit:

1) The verification plan.
2) All the SystemVerilog source code you wrote. You may submit the relevant portions of your simulation output, when relevant to the documentation in the final report. Create a `README` file that documents the purpose of all source files and describes how your verification testbench must be executed. If you wrote new scripts to run your tests, please submit those, too, so that we can reproduce your results.
3) The verification report.

You are NOT required to submit the source design files, unless you made any changes to them that are required in order to run your simulations.

5. How to submit

To submit the project, follow the instructions on the submission webpage:
http://eecs.umich.edu/courses/eecs578/submit

Please create a single archive file that includes all the deliverables (report, source code, *etc.*), and upload it to the submission webpage. We only accept the following file extensions: tar, gz, tgz, bz2 and zip. The file name will be automatically changed to your uniqname by the submission webpage.

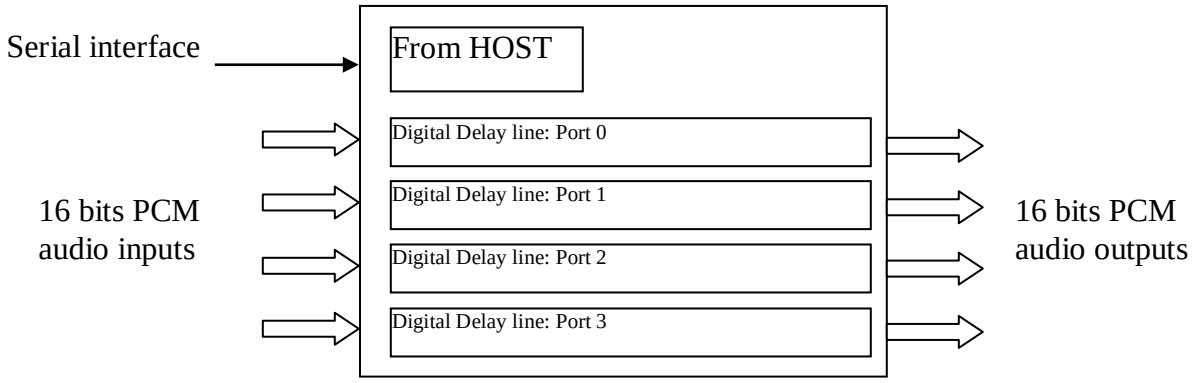
Digital Audio Reverberation module
Functional specification document

1. Introduction

Although this design represents a dedicated DSP block for an audio application (digital reverberation, digital mixer, etc.) it has been built for HW verification educational purposes. The purpose of this document is to describe the functionality of this digital audio component.

2. Functional Overview

The digital reverberation module consists of a serial interface that communicates with a Host processor under host processor control. It is characterized by having 4 digital delay line channels with each channel having its own 16 bit PCM input and 16 bits PCM output. Each PCM sample received at a particular port is routed to the correspondent output port. Every output represents the input affected by a programmable delay: This delay is programmed via the Host serial interface. The max delay is 8 clock cycles.



3. Logic Interfaces Specifications

There are 2 types of interfaces: a serial interface (from Host) used to program the delay for each port and a 16 bit PCM interface (input and output) that allow the digital audio samples to flow in the circuit. A more detailed description follows in the System Description Section.

4. Performance

The system is supposed to support digital audio at a frequency of 44.1Khz.
The speed of the serial PCM interface is also 44.1Khz

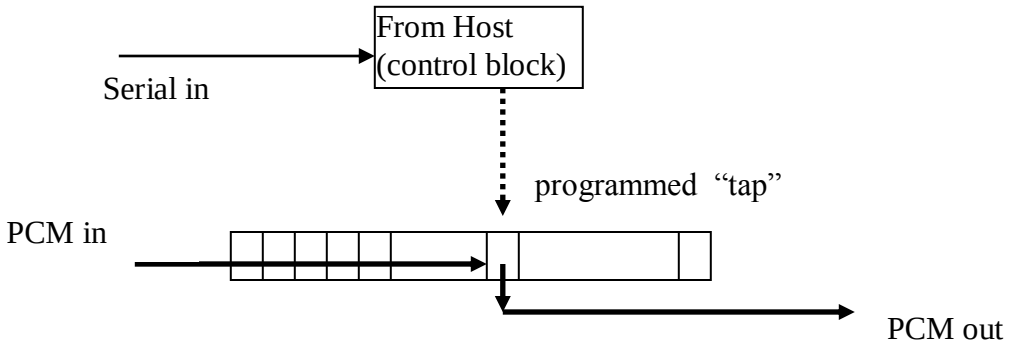
5. System Description

This section describes the operations performed by the Digital Reverberation Module.

5.1. Structural Diagram

The structural diagram of the Digital Reverberation Module is reported below.

In order to make things easier we report only one delay line: the important issue is that all the delay lines work identically since the "From Host" interface will control each delay line in the same way.



5.2. The "from Host Control" block

This block is accessed serially: Only the "write" mode is supported. Every time this block is accessed, it will be given two pieces of information (three after the read mode is supported – in a future generation of this design):

1. which delay line to program (0 to 3)
2. the amount of delay (0 to 7)

It is not possible to read the registers back, so you will have to think of some type of test (direct test) to verify that this block correctly programs the delay for each delay line.

5.3. The "delay line" block

This block is intended to delay the PCM output a programmable number of clock cycles. The "tap" can be chosen programming the "from Host" block as discussed in the previous paragraph. The tap chosen will determine the output delay. If the tap chosen is zero, it programs an output delay of 1 clock cycle. A tap of 7 (the maximum delay) will delay the output by 8 cycles. This can be of special importance for your part of the work. If your testbench is waiting for data with an expected delay of 8 cycles while the tap is reprogrammed to fewer cycles, let's say 3, then the data from tap 4 through 8 will not arrive at the output, and there will be a discontinuity in actual data transmitted. Reprogramming of a Delay line during data delivery is normal, expected behavior, so your testbench must take into account the possibility of this programming change and adjust for the discontinuity in data.

6. Functional Description

In the following, we will describe some basic assumptions made while building this design and provide more technical details on the protocol.

6.1. Reset

Every block that belongs to the Digital Reverberation Module is provided with a synchronous reset. The "control" block and the "delay line" share the same synchronous reset, "`rst_`".

6.2. Interfaces

6.2.1. PCM interface

This is a 16 bits PCM interface: the audio samples are represented in a linear digital format: every port is provided with one input and one output. There are no control signals for this type of interface since the data changes at the system clock frequency of 44.1 KHz. Each PCM channel is independent of every other channel. There is no interaction between channels.

The PCM interface can be reprogrammed at anytime. No more than a maximum of seven (7) words of data should be lost because of reprogramming the delay tap. This maximum occurs when the old delay for the PCM is seven (7), and the new delay is zero (0). In this case, seven words of data in the delay line would be lost when the tap is moved from the output of the last delay register to the output of the first delay register. In the inverse case, when the delay is changed from a smaller delay to a larger delay, the data sequence that had already passed out of the tap will be repeated up to the difference between the new and the old tap value. For instance, when the delay is changed from zero (0) to seven (7), then no more than 7 words of data sequence should be repeated.

6.2.2. Serial interface from Host processor

At this time, the frequency of operation of the serial interface is the same as that of the system (44.1KHz).

Module inputs from the Host

There are two pins controlled by the host that allow the Host processor to program the Digital Reverberation Module:

`prgrm_go_`
`prgrm_in`

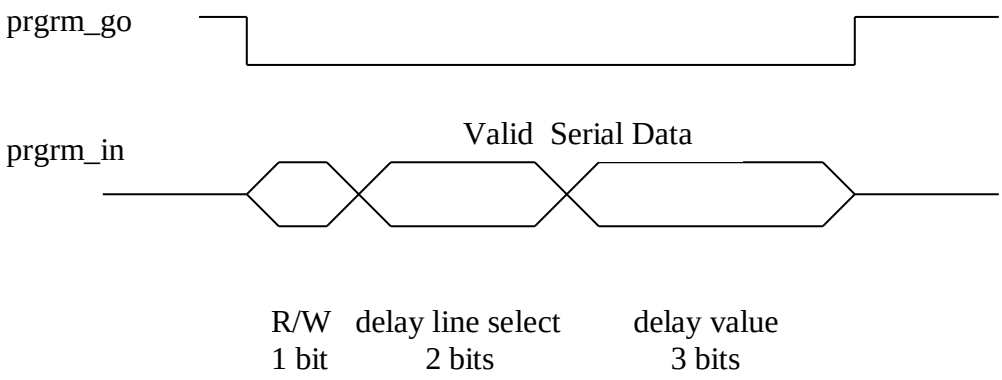
`prgrm_go_` : This signal is asserted active low (1'b0) and frames the valid serial program sequence. Once `prgrm_go` is asserted, it must not be de-asserted until the valid program sequence is complete. If `prgrm_go` is de-asserted before the entire program sequence is complete, the `err_` pin will be asserted signaling an invalid termination of the programming cycle. See the `err_` description for more details.

`prgrm_in` : This signal is used to program a legal programming sequence of six bits. The valid `prgrm_in` sequence contains the following information in the following order:
bit 1: The first bit in the sequence selects the programming mode.

- 1'b0 : A low value selects the write mode
- 1'b1 : A high value selects the read mode. This mode is not supported in this version of the Digital Reverberation module. If this mode is selected, the `err_` pin will be activated. See `err_` for more details.

bits 2-3: These two bits select one of four delay lines. The lsb should be shifted in first.
bits 4-6: These three bits select one of eight possible delay values. A value of 0 selects for a delay of one cycle and a value of 7 a delay of eight cycles. The lsb should be shifted in first.

Timings of the serial interface
The timings of the serial interface are reported in the following diagram.



rst_ : This synchronous signal drives all of the outputs to zero and clears the contents of all the delay line registers when asserted low, 1'b0.

Module outputs to the Host
err_ : This synchronous signal is an output and is asserted low (1'b0) for one cycle whenever the host block reaches an illegal state. The host will then return to the idle state to wait for the start of a program sequence.