2IN35 – VLSI Programming – Lab Work

Assignment 5: 1024x Sample Rate Converter

Hrishikesh Salunkhe, h.salunkhe@tue.nl,
Alok Lele, a.lele@tue.nl

June 9, 2015

Contents

1 Introduction **3**

2 Background **3**
2.1 Example 3

3 Environment Setup **4**

4 Objective **4**

5 Additional Information **4**
5.1 Accounting for clock cycles 4
5.2 Synchronous Handshake Signaling: 5
5.3 Using Block RAMs 5

6 Reporting **5**

1 Introduction

In this assignment you will design and implement a scalable version of the sample rate converter of assignment 4. For this assignment you need the following files from the website:

- Dummy filter and test-bench in `multiscalar.zip`
- Multi rate EDK systems: `L5_audio_multi_rate_realtime.rar` and `L5_audio_multi_rate_offline.rar`
- Example audio files: `input.bin`

2 Background

In assignment 4 we implemented a sampling rate converter which is used to convert a signal from one sampling rate to another, while changing the information in the signal as little as possible. In assignment 4 we converted from a sampling rate of 44.1 kHz to 48 kHz.

Imagine now a scalable version of the same sampling rate converter where we consider our sample rate converter to simultaneously up-sample a number of input streams that are interleaved into a single multiplexed stream. We define scalability as the range of input streams that the sample rate converter can accept. The larger the range the better the scalability. This scenario can be observed in multi-radio modem where multiple independent radio streams are received by a receiver sent through a filter or rate converter before they receive their individual processing.

2.1 Example

Figure 1 shows three independent input streams that are multiplexed into a single stream in a fixed order. Here the scalability factor is $n = 3$ streams with each input stream has a sampling frequency of 44.1 KHz. The multiplexed stream with a sampling frequency of $n \times 44.1KHz$ is then up-sampled by the sampling rate converter to $n \times 48KHz$. At the output end the rate-converted stream is then de-multiplexed into n individual streams each with a sampling frequency of 48 KHz. It is essential that the order of the output produced by the sampling rate converter must match the order of its input.

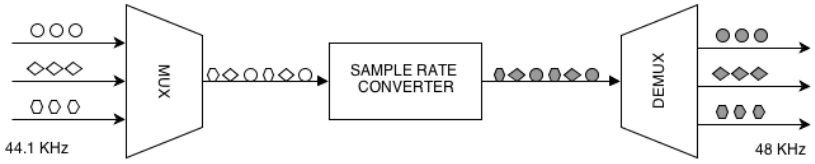

Figure 1: Scalable sampling rate converter for 3 input streams

The MUX and DEMUX stubs are already provided to you for this assignment in the test-bench (for simulation) and the multi-rate EDK (for the FPGA). Your task is to replace the dummy filter in between the stubs with your implementation of the scalable sampling rate converter.

3 Environment Setup

Within the package *multi-scaler.zip*, you will find a dummy filter *filter.v* a test-bench *test.v*. Within the test-bench you can specify the number of streams (i. e. the scalability factor n) that you intend to multiplex. For simplicity, out of the n multiplexed streams, only one stream has input data while all other streams contain only zeros. The stream index which corresponds to your actual data stream is specified by the test bench. The test-bench multiplexes all streams and sends a single multiplexed stream to the filter. At the output side the test-bench again receives the output stream from the filter and de-multiplexes it to the original number of streams. The stream corresponding to the original data stream index is then stored in an output file.

4 Objective

In this assignment you must design a sampling rate converter that is aligned to the number of multiplexed streams. For simplicity, all the input streams have the same sampling frequency of 44.1 KHz, and need to be converted to output streams with sampling frequency with sampling rate 48 KHz. To demonstrate the scalability of you design implement the sampling rate converter for a 2, 32, 128, 512 and 1024 streams. (HINT: Start from smaller number of streams, but design your system such that it is easily scalable to larger number of streams).

IMPORTANT: Sending actual data on just one out of the n streams is to facilitate easy debugging. The sample rate converter you have implemented MUST NOT use the index of the actual data stream, but must up-sample all input streams. This will be tested while demonstrating your implementation. Also the scalability factor n must be specified as a parameter for your design such that the design must work for a different number of streams simply by modifying this parameter.

5 Additional Information

Following is some additional insight to help you complete this assignment.

5.1 Accounting for clock cycles

In assignment 3, you learned how to minimize hardware resources utilized while maintaining the throughput of your filter design to (above) 44.1 KHz. This involved reusing hardware component at the expense of clock cycles. While some computation is performed every clock cycles on a set of shared resources, an output is produced only every few clock cycles. Contrarily, we must now try to optimize the rate at which output can be produced even if it costs more hardware. For instance, if an implementation of 44 KHz to 48 KHz up-sampler requires 10 cycles to produce one output, then for an FPGA running at 100 MHz this would imply that you theoretically cannot scale up to more than $(100 \times 10^3/10)/48 \approx 208$ streams. Thus your design must require as few clock cycles as possible to produce an output.

5.2 Synchronous Handshake Signaling:

The dummy filter `filter.v` provides a basic implementation of synchronous handshake signaling (refer lecture slides of week 8) to demonstrate that the handshake need not slow down a system design and can even facilitate an output to be produced at every clock cycle!

To facilitate faster design we have also modified the testbench to behave as an active element that will keep the next data ready to send to the filter. The testbench is therefore connected to your up-sampler via a passivator. Note that to allow synchronous handshake signaling we provide the synchronous implementation of the passivator in which we use an AND-gate instead of a Muller-C component.

5.3 Using Block RAMs

Since we have multiple streams multiplexed into a single stream, our design will require larger data storage on the input and output side. This may either inferred by the compiler as either Distributed RAMs or Block RAMs. Distributed RAM is created using the available LUTs on the FPGA can significantly slow down your system. Instead the Spartan 6 board comes with 116 dedicated Block RAMs. Each BRAM contains 1024 18-bit words, and can have 2 independent ports to perform reads and/or writes. BRAMs are automatically inferred from Verilog code if you use the proper coding constructs. Getting this right is a bit tricky: each port of the BRAM must have its own always-block, containing only the logic to read from and/or write to that port. If other statements are present, the synthesizer does not infer BRAMs. Furthermore reads should be done synchronously, meaning that the output of the block RAM becomes available 1 clock cycle after the address is supplied. In other words, it is pipelined internally.

To refer the coding examples go to *Edit* $\rightarrow$ *Language Templates*. An explorer window will open. Go to *Verilog* $\rightarrow$ *Synthesis Constructs* $\rightarrow$ *Coding Examples* $\rightarrow$ *RAM* $\rightarrow$ *BlockRAM* $\rightarrow$ *Dual Port*.

6 Reporting

After you have successfully simulated your design using the test-bench you must run the design on your FPGA board. Once you have successfully implemented the scalable sampling rate converter on the FPGA proceed to answer the following questions:

1. Assuming a functionally correct system running at a 100 MHz clock, can you derive a theoretical bound for the scalability of the $44.1KHz \rightarrow 48KHz$ sampling rate converter?
2. What is the maximum scalability achieved by your design? (HINT: This may be different from the five scalability factors you are asked to demonstrate.)
3. Justify whether all output streams meet their throughput requirement of 48 KHz.