

TCE Exercise: Tremor Audio Decoder on TTA



TCE Exercise: Tremor Audio Decoder on TTA by [Otto Esko](#) is licensed under a [Creative Commons Attribution-ShareAlike 3.0 Unported License](#).

General information

You need to have TCE tools installed on your computer in order to complete this exercise. TCE tools can be downloaded from <http://tce.cs.tut.fi/download.html>.

In this you will compile and simulate Tremor Ogg Vorbis [1] on several TTA processors to see how changes in the architecture affects performance. Tremor Audio Decoder software sources used in this exercise should have been shipped with this document. The sources have been modified to suite the needs of this exercise and allow easy compilation, simulation and verification process.

Directory '**tremor_tta**' contains the Tremor source codes and all the essential files. All the commands in these instructions assume that your current working directory is **tremor_tta**.

Directory **tremor_tta/adf** contains the TTA processors which will be used in this task.

There are also three short ogg vorbis samples which are stored in **tremor_tta/input_samples**. These samples are published under Creative Commons license. Samples are authored by Michael David Crawford (sahara) and CambridgeBayWeather (a_canadian_boat_song and aiken_drum). See file **tremor_tta/input_samples/attribution.txt** for more information. The samples have been cut to a length of 20 seconds to speed up the exercise.

Directory **tremor_tta/ref_output** contains decoded raw bitstreams of the input samples. These raw bitstreams will be used to verify correct behavior of the decoder.

Test process

You need to modify the Makefile in tremor_tta directory to change the processor architecture you are using. This can be done by changing the architecture definition file (ADF) variable in the Makefile. By default it is **adf/minimalistic.adf**. Use a text editor (such as gedit or kate) for modifications.

Use TTA Processor Design-tool (ProDe) to view the architecture. ProDe can be started from command line with command:

```
prode &
```

You can also give the adf file as a command line parameter:

```
prode processor.adf &
```

where processor.adf is the name (and path) of the desired ADF.

To compile and simulate tremor against the selected architecture, simply run make:

```
make
```

Make will compile the source codes and execute ttasim instruction set simulator, which will output statistics of processor resource utilization as well as execution cycle count. Write down the cycle count for each architecture.

Successful simulation will create an output file called **output.raw** which contains the decoded raw pcm bitstream. To verify that the result is correct, you must compare it to the reference output file. This file is given to you and it has been generated by decoding the same input file on a desktop PC version of Tremor audio decoder. We assume that the correct behavior of Tremor has been verified on desktop PC so we can use the desktop PC version as reference.

By default, Makefile uses **a_canadian_boat_song_20s.c** as input data and the corresponding reference output file is **ref_output/ref_a_canadian_boat_song_20s.raw**. You can easily compare simulation output and reference output by using diff command:

```
diff --brief output.raw ref_output/ref_a_canadian_boat_song_20s.raw
```

If the command does not output anything, the two files were identical. If the files differ, the command will output: *Files output.raw and ref_output/ref_a_canadian_boat_song_20s.raw differ*

You can also play the raw output file using the **play_raw.sh** script in tremor_tta directory. The script invokes *sox* program to play the output. Here is the command how to use the script:

```
./play_raw.sh output.raw
```

If you don't want to listen the whole sample, press `ctrl+c` to quit the program. **Please notice** that the script **may not work** if you have different version of *sox* installed.

Task 1: Evaluate architectures

Perform the steps described in Test process for the processor architectures listed below and collect the execution cycle counts. The processor architectures to be tested in this task are:

- minimalistic.adf
- minimalistic_improved.adf
- three_bus_std_mul.adf
- six_bus_std_mul.adf
- ten_bus_std_mul.adf
- clustered_mul4.adf (simulation takes a long time with this architecture!)
 - Interconnection network of this architecture has been modified to increase maximum clock frequency
- rather_huge.adf

If you notice strange behavior after changing architecture, it might be a good idea to run:

```
make clean
```

and try again.

Questions for task 1:

1. List the execution cycle counts for all the tested architectures
2. Why does the execution clock cycle count reduce when `minimalistic_improved.adf` is used compare to `minimalistic.adf` even though both architectures have the same computational resources?
3. Why the difference in execution cycle count is small between `ten_bus_std_mul.adf` and `rather_huge.adf` even though `rather_huge.adf` has much more computational resources than the other architecture?
4. Length of the audio sample is 19.98 seconds. Calculate the minimum clock frequency (in MHz) for decoding the audio sample in real time for each simulated architecture. (Hint: minimum frequency is equal to the required number of clock cycles per decoded audio second)

Task 2: Custom operations

In this task you will test whether or not it would be helpful to use custom operations with this application. One obvious candidate for custom operation would be mdct because it contains heavy computation. But there are also other smaller operations which can potentially result in good speed up. For example mdct uses custom multiplication operations which are emulated on software by default. These custom multiplication operations multiply two 32-bit signed integers and produce internally a 64-bit result. The result of the whole operation is a 32-bit signed integer which is taken, depending on the operation, from different positions of this 64-bit internal result.

These operations are originally defined in file **misc.h** and their TTA counterparts (custom operation macros calls) are defined in **multiplications.h**. Take a look if you're interested.

Your task is now to enable custom multiplication operations for one architecture. Let's use the **three_bus_std_mul.adf** as our starting point. First copy the architecture:

```
cp adf/three_bus_std_mul.adf adf/three_bus_custom_mul.adf
```

and then open the copy with ProDe:

```
prode adf/three_bus_custom_mul.adf &
```

Next, remove the multiplication function unit and its input and output sockets. Then add the custom multiplication function unit from **tremor.hdb** (Edit -> Add from HDB -> select mul(2), mult32s(2), mult31s(2), mult30s(2), mult30s_shifted15s(2) -> click Add -> click OK). Then select Tools -> Fully Connect IC and save the architecture.

Now open the **Makefile** and change the ADF variable's value to the new architecture you created. This time you also have to add **-DMULT_OPS** to **USER_COMPILER_SWITCHES** in order to enable custom operations in the code. Then run make clean and proceed to compile, simulate and verify like done earlier.

Questions for task 2:

1. What is the execution cycle count now?
2. How much is the speed up compared to three_bus_std_mul.adf?
3. Calculate the minimum clock frequency (in MHz) for real time decoding of the audio sample.

Task 3: How input data affects execution time

In this task you will study the affect of input data on execution time using the architecture created in previous task (**three_bus_custom_mul.adf**) using custom multiplication operations. There are three input samples (44.1 kHz sample rate, stereo):

- input_samples/a_canadian_boat_song_20s.c
 - 44.1 kHz sample rate, 2 channels, average bitrate of 364 kbps
 - length: 19.98 seconds
- input_samples/aiken_drum_20s.c
 - 44.1 kHz sample rate, 2 channels, average bitrate of 365 kbps
 - length: 19.96 seconds
- input_samples/sahara_20s.c
 - 44.1 kHz sample rate, 2 channels, average bitrate of 185 kbps
 - length: 19.97 seconds

You can change the input sample by modifying input_data variable in the **Makefile** (see the comments in the Makefile for further instructions). Then execute:

```
make clean && make
```

and collect the results. Remember to verify the output.

Questions for task 3:

1. Calculate clock cycle count per decoded audio seconds for all input data samples.
2. Does input data affect decoding cycle count per audio second? Why?

Questions in general:

1. Up to this point we have measured performance in clock cycles but the execution run time is more interesting and descriptive measure of performance. Table 1 introduces synthesis results, i.e. logic element usage and maximum clock frequency, on a Stratix II FPGA for some of the architectures. Use the maximum clock frequencies to calculate execution run times for the architectures listed in table 1. List the calculated execution run times.
2. Using results from question 1 and information from table 1:
 - a. Draw a graph of logic element usage vs. run time for the architectures.
 - b. Draw a graph of logic element usage vs. execution clock cycle count
 - c. How does the usage of custom operations show in the graphs? Speculate, is it better to try to find and use custom operations than to increase “standard” computational resources on the processor?
3. Calculate execution clock cycles per decoded audio output second ratio for all the architectures in table 1 and draw a figure of it.
4. Using the results from previous question, which architecture (**excluding three_bus_custom_mul**) would be fastest
 - a. at 50 MHz clock frequency?
 - b. at 75 MHz clock frequency?
 - c. at 100 MHz clock frequency?
5. What benefits and opportunities customizable architecture brings to DSP applications?

Table 1: Synthesis results for Stratix II

Architecture	Logic Elements	fmax / MHz
minimalistic.adf	1214	141
minimalistic_improved.adf	1782	115
three_bus_std_mul.adf	2825	111
three_bus_custom_mul.adf	2931	106
six_bus_std_mul.adf	5637	90
clustered_mul4.adf	5218	149
ten_bus_std_mul.adf	11280	74

Documentation

These documents are referred in the work instructions.

[1] **Tremor Ogg Vorbis home page**, <http://xiph.org/vorbis/>