wassolldas – An audio player using an Arty A7-35t FPGA Development Board

# Summary

This project “wassolldas” aims to implement WSOLA for Time-Scale Modification on an arbitrary sound file. In simpler terms, WSOLA shall transform a given sound file so that it is played back either slowed down or sped up without a change in pitch. Additionally, the transformation shall occur “online”, i. e. in real time. The Arty A7-35t FPGA development board provided by DHBW should run this algorithm over a sound file which is sourced from internal ROM storage. Having slowed down or sped up the sound file, it should play back the sound file using a Pmod module called “Pmod I2S2”. This module contains two audio jacks, a line in and a line out connection. Needless to say, “wassolldas” exclusively uses the line out connection.

Tragically, I could not implement WSOLA and resorted to naïve resampling. While it did count as Time-Scale Modification, it did not retain the pitch.

# Material

## Arty A7-35T FPGA Development Board

According to its manufacturer [Digilent](https://digilent.com/reference/programmable-logic/arty-a7/start), Arty A7-35T is a FPGA development board fitted with an Artix-7 processor which among other things provides 5200 logic slices and 1800 Kbits of Block RAM. It has many peripheral options already soldered onto the development board. For instance, this project utilizes 4 switches, 3 LEDs, 1 button, and Jmod Connector JA.

By interacting with the switches, someone can adjust the playback speed. SW0, SW1, SW2, SW3 correspond to 0.25x, 0.5x, 1x, and 2x respectively. Its possible to combine multiple factors. Thus, the playback speed is the sum of every factor selected by these 4 switches.

By pressing the reset button BTN0, a system reset is triggered. A system reset can also be triggered when internal clock signals are not stable. To document the status of playback, 3 status LEDs are exposed to the users. Firstly, if a system reset is in progress, LD0 is blue. If the reset button triggers a reset, LD1 is red. Lastly, if all internal clock signals are stable, LD2 is green.

## Pmod I2S2

[Digilent](https://digilent.com/reference/pmod/pmodi2s2/reference-manual) offers the Pmod module “Pmod I2S2” which allows Arty A7-35T to “transmit and receive stereo audio signals via the I2S protocol”. It supports up to “24 bit resolution per channel at input sample rates up to 108 kHz”.

# Research

## Audio playback via I2S

The Vivado project “wsolated” houses the source code which is to be uploaded onto the Arty A7. The audio playback section of “wsolated” (especially the I2S encoding part) is largely inspired by [Daniel Kampert’s tutorial on how to play audio with an FPGA](https://www.hackster.io/Kampino/playing-audio-with-an-fpga-d2bc85).

A picture containing diagram

Description automatically generated

Figure 1: Timing diagram of I2S

Figure 1 illustrates I2S operating principle. In “wsolated”’s case, Arty A7 serves as the so-called “I2S Master” whereas Pmod I2S2 functions as the “I2S Slave”. Thus, Arty A7 is responsible for setting SCK (in “wsolated” called SCLK”) and WS (in “wsolated” called WS). SCK and WS stand for “Serial Clock” and “Word Select” respectively. WS specifies which channel the transmitted data shall target. For instance, if WS is low, the transmitted data will target Channel 1 corresponding to the left ear otherwise Channel 2 i. e. right ear. Both SCK and WS shall behave like clock signals, i. e. periodically switch from low to high and vice versa. SCK has a frequency of 1.536 MHz and WS has a frequency of 48 kHz.

WS’s period is proportional to SCK’s period by a factor of 16. This translates to a word length of 16 bits which can be transmitted over “Serial Data” (SD or in “wsolated” called “SDIN”) for a particular WS. As Figure 1 shows, the transmission of the next word for a specific WS is delayed by one cycle after a signal edge for that specific WS was detected.

As stated before, Arty A7 sends data in the form of words to Pmod I2S2. Therefore, like SCK and WS, Arty A7 drives SD. The audio data needs to be encoded as 16-bit signed PCM words. PCM-encoded audio data (Pulse Code Modulation) captures raw air vibration for a specific time. For example, “0x8000” is the lowest possible amplitude while “0x7FFF” is the highest possible one. As opposed to stereo operation, “wsolated” pursues monophonic playback. So, if a word is sent when WS is low, the same word must be sent when WS is high.

Optionally, it’s possible to transmit a “Master Clock” (in “wsolated” called “MCLK”) alongside SCK, WS, and SDIN. According to [Wikipedia](https://en.wikipedia.org/wiki/I%C2%B2S), it is used to synchronize internal operations.

## Time-Scale Modification

As “Time-Scale Modification” (or TSM) remains sparsely documented on major platforms (i. e. [Wikipedia](https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling)), [Jonathan Drieger’s master thesis](https://www.audiolabs-erlangen.de/content/05-fau/professor/00-mueller/01-students/2011_DriedgerJonathan_TSM_MasterThesis.pdf) provides great insights into this topic. Their master thesis serves as the foundation of the following explanation. Time-Scale Modification is the process of speeding up or slowing down the tempo of an audio signal.

### Naïve resampling

The naïve approach to modifying the audio playback speed is to change the sample rate. In terms of playback, the sample rate defines how many PCM-encoded samples are played back per second. For instance, if the sample rate decreases, the playback speed will decrease and vice versa.

Unfortunately, the resampled audio signal suffers from the “Chipmunk effect”. This effect is characterized by a change in pitch. E. g. speeding up a voice recording with naïve resampling results in a higher perceived tone in voice.

Graphical user interface, text, application

Description automatically generated

Figure 2: Resampling pseudo code (DT: Discrete time, CT: Continuous time)

### OLA

OLA (standing for Overlay and Add) is an TSM algorithm which is superseded by multiple algorithms with one of them being WSOLA. Nevertheless, as WSOLA’s name implies, WSOLA builds upon OLA. In essence, small segments are taken from an input audio signal. These segments are then stitched together by “crossfading from one segment to the next to achieve the desired time-scale modification”.

Text

Description automatically generated

Figure 3: OLA pseudo code (DT: Discrete time)

The implementation of the pseudo code as written down in Figure 3 may need the following elaborations:

* represents the output window position of the nth window.
* represents input window position of the nth window.
* describes the number of samples contained in the audio signal y.
* , whereas is the window length
* .
* should be greater than or equal the period of the lowest frequency present in the playback.
* Typically, should be 0.5 and should be [the Hann function](https://en.wikipedia.org/wiki/Hann_function). This way, no amplitude adjustment is needed.

### WSOLA

OLA suffers from modulation and phase jump artifacts as seen in Figure 4. These result from mismatching phases of overlapping segments.

Text

Description automatically generated

Figure 4: Modulation and phase jump artifacts

To combat this, WSOLA and other successors to OLA avoid introducing new phase jumps. Specifically, for WSOLA, WSOLA now has a certain degree of freedom when choosing the next segment for a given segment. The next segment should be as similar as possible to the natural progression of the given segment. Figure 5 demonstrates this. To determine the similarity of two segments, one can calculate the cross correlation, i. e. pointwise multiply both segments and compute the sum of the resulting product. The segment with the highest sum is the most similar.

Chart

Description automatically generated

Figure 5: WSOLA natural progression

Graphical user interface, text, application

Description automatically generated



Figure 6: WSOLA pseudo code (The elaborations as defined in OLA also apply here)

# Protocol

After experimenting with I2S in a separate Vivado project and after researching TSM, I started working on “wsolated”. To create a new project for Arty A7-35T, I needed to set the processor target to XC7A35TICSG324-1L. Also, I imported [this XDC file for Arty A7-35T from GitHub](https://github.com/Digilent/digilent-xdc/blob/master/Arty-A7-35-Master.xdc) and created VHDL source files. Having successfully initialized this project, I implemented audio playback. This implementation was heavily inspired by [Kampert’s work](https://www.hackster.io/Kampino/playing-audio-with-an-fpga-d2bc85) (refer to the implementation section in order to dig deeper into the level of inspiration). Firstly, I defined “I2SClockGenerator”, “I2STransmitter”, and “ROMReader” entities. “I2SClockGenerator” and “ROMReader” required me to generate the IP “Clocking Wizard” and “Block Memory Generator” respectively. In the “Top” entity, I connected the three aforementioned entities with each other and / or with “Top”’sports.

Kampert’s code read the audio data from ROM, which is why I wrote a Python script to convert a wav file to a coe file, a file format accepted by Arty’s ROM. This Python script then allowed me to play back an arbitrary audio file.

Finally, I worked on implementing OLA. Firstly, in anticipation of WSOLA, I created “WSOLATransformer” and put it between ROM and the section responsible transmitting via I2S. Unfortunately, after weeks of debugging, where I made false assumptions about the ROM timing diagram and where I wrongfully interpreted OLA’s pseudo code, I had to give up on OLA. During that time, I faced enormous challenges in transitioning from traditional programming to VHDL. Me setting ambitious goals further exacerbated these difficulties.

To still have a Minimum Viable Product to show for, I naïvely resampled the audio signal. “WSOLATransformer” was replaced by the entity “Bridge”, otherwise the naïve code was a continuation of OLA. To verify I2S and later TSM, the design was repeatedly tested using the testbench “TopTB”. Having passed that test, I generated a bitstream file of “wsolated” and called it “wassolldas\_fallback.bit”. I, then, programmed Arty using that bitstream and validated “wsolated” on Arty. The cycle of developing, verifying, and validating was repeated for hundreds of iterations.

# Implementation

In contrast to Kampert’s work, “wsolated” needs to target a different FPGA development board, i. e. Arty A7 instead of Zybo Z7. To facilitate this switch, one merely needs to modify the Arty A7 35t xdc file. Furthermore, Arty A7’s reset shall be active high instead of Zybo Z7’s active low.

Plan of attack

Working with the Block Memory Generator proved to be a lot more difficult than anticipated.

Initially, all plans

Look at timing graph of block memory

ROMReader takes two cycles from increment to update.

AXI?

<https://stackoverflow.com/questions/66413266/vhdl-fsm-output-delayed-of-1-clock-cycle>

# Resources

## Basics

* [(66) ESP32 Sound - Working with I2S - YouTube](https://www.youtube.com/watch?v=m-MPBjScNRk&t=1527s)
* [GitHub - schreibfaul1/ESP32-audioI2S: Play mp3 files from SD via I2S](https://github.com/schreibfaul1/ESP32-audioI2S)
* https://youtu.be/ZNunxg7o8l0
* <https://www.fpga4fun.com/VHDLTips.html>
* [Taming Multiple Drivers (xilinx.com)](https://support.xilinx.com/s/article/1034745?language=en_US)

## Component specific

* [Pmod I2S2 Reference Manual - Digilent Reference](https://digilent.com/reference/pmod/pmodi2s2/reference-manual)
* [Getting Started with Digilent Pmod IPs - Digilent Reference](https://digilent.com/reference/learn/programmable-logic/tutorials/pmod-ips/start)
* [Pmod MicroSD - Digilent Reference](https://digilent.com/reference/pmod/pmodmicrosd/start?redirect=1)
* [GitHub - Digilent/vivado-library](https://github.com/Digilent/vivado-library)
* [Vivado Library - Digilent Reference](https://digilent.com/reference/vivado:library)
* [vivado-library/ip/Pmods at master · Digilent/vivado-library · GitHub](https://github.com/Digilent/vivado-library/tree/master/ip/Pmods)
* [vivado-library/ip/Pmods/PmodSD\_v1\_0 at master · Digilent/vivado-library · GitHub](https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodSD_v1_0)
* [AXI Basics 1 - Introduction to AXI (xilinx.com)](https://support.xilinx.com/s/article/1053914?language=en_US)
* <https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0012-vivado-high-level-synthesis-hub.html>
* <https://github.com/riccardonicolaidis/PmodSD_Arty_A7_Project>
* <https://docs.xilinx.com/v/u/en-US/pg058-blk-mem-gen>

## Previous implementations

* [Playing audio with an FPGA - Hackster.io](https://www.hackster.io/Kampino/playing-audio-with-an-fpga-d2bc85) goat
* [Play wav-files from an SD card with the ZYBO - Hackster.io](https://www.hackster.io/Kampino/play-wav-files-from-an-sd-card-with-the-zybo-ba4469)
* <https://justanotherelectronicsblog.com/?p=402>
* [Playing an Audio File - Programming FPGAs Getting Started with Verilog - FPGAkey](https://www.fpgakey.com/tutorial/section570)
* [Project FPGA-Audio – FPGA based MP3/WAV Player - FPGA SOLUTIONS (google.com)](https://sites.google.com/site/tgptechnologies/fpga-project/fpga-audio-fpga-based-mp3-wav-player)
* [Playing MP3s From An FPGA | Hackaday](https://hackaday.com/2012/03/13/playing-mp3s-from-an-fpga/)
* [GitHub - ultraembedded/FPGAmp: 720p FPGA Media Player (RISC-V + Motion JPEG + SD + HDMI on an Artix 7)](https://github.com/ultraembedded/FPGAmp)

## Time stretching

* [Audio time stretching and pitch scaling - Wikipedia](https://en.wikipedia.org/wiki/Audio_time_stretching_and_pitch_scaling)
* <https://www.jstage.jst.go.jp/article/elex/11/14/11_11.20140387/_article>
* <https://github.com/k2kobayashi/Shifter/blob/master/shifter/shifter.py>
* <https://www.audiolabs-erlangen.de/content/05-fau/professor/00-mueller/01-students/2011_DriedgerJonathan_TSM_MasterThesis.pdf>

## Python

<https://www.pythontutorial.net/python-basics/python-write-text-file/>

## TODO:

Familiarize myself with FPGA development

Output sound

~~Read SD Card data~~

Modulate speed