


GSoC Proposal for BeagleBoard.org

BeagleBoard.org
Jul 02, 2025

Table of contents

Chapter 1

Introduction

1.1 Summary links

- **Contributor:** Alec Denny
- **Mentors:** Jack Armitage, Chris Kiefer
- **Code:** Google Summer of Code / greybus / cc1352-firmware · GitLab
- **GSoC:** Google Summer of Code

1.2 Status

This project is currently just a proposal.

1.3 Proposal

1.4 About

- **Forum:** u/alecdenny (Alec Denny)
- **OpenBeagle:** openbeagle.org/alecdenny
- **Github:** github.com/alecdenny
- **School:** Columbia College Chicago
- **Country:** United States
- **Primary language:** English
- **Typical work hours:** 8AM-5PM US Pacific
- **Previous GSoC participation:** N/A

Chapter 2

Project

Project name: Granular style transfer instrument with differentiable logic gate networks

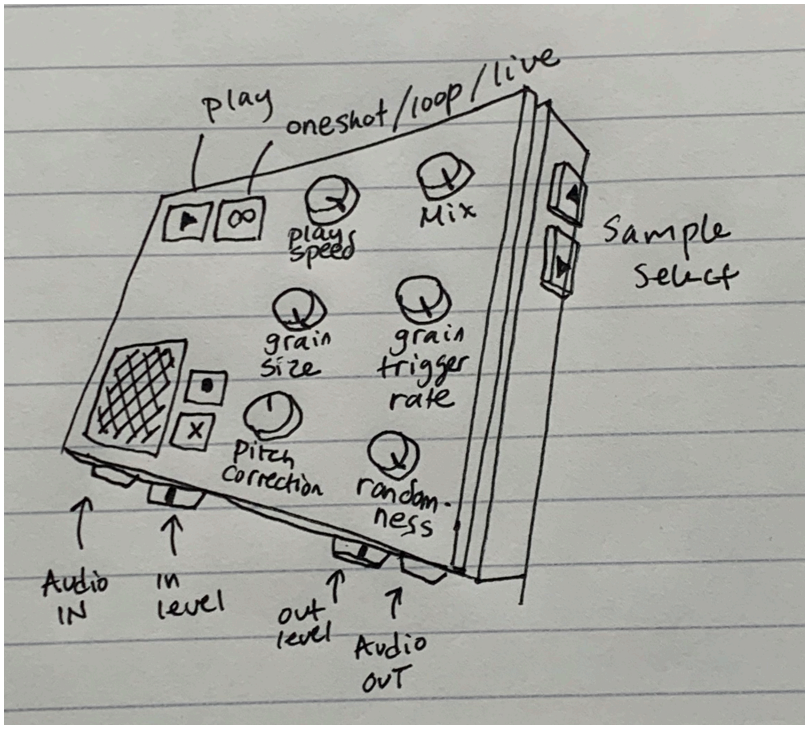
2.1 Description

Create an AI powered granular style transfer algorithm runnable on the bela board with the goal of creating a usable instrument. The algorithm will allow for users to provide their own style target sample whose timbral characteristics can be explored through interaction via the microphone or audio input. The synthesis engine will be based on granular / concatenative synthesis methods, the AI component will be a network that maps input spectral features to a predicted point in the sample from which to playback audio.

Neural Network: The AI component of the synthesis engine will have input in the form of MFCCs, coefficients which describe the frequency spectrum in terms of amplitude. MFCCs are like the Fourier transform's amplitude spectrum but with amplitude and frequency scaled to match the characteristics of human perception. The model will use this information to predict from what point in the target sample to play back audio. Based on input, the network should output the most perceptually similar grain from the sample. To train the network, MFCCs from each grain of the sample will be input, with the prediction being the time offset at which the grain occurs in the sample. The neural network itself will be based on differentiable logic gate networks whose neurons have boolean inputs and outputs, and each perform one of 16 logic gate operations [1]. These networks can contain fewer neurons and be much smaller because each neuron's operation can be complex, and weights do not need to be stored, only which logic gate operation to perform [1].

Output representation will be similar to thermometer format, which is closest to how classification is achieved in models previously trained by the authors and our mentors. The classification examples use binary adders, which sum a group of output neurons in order to predict the most likely class. In this case, many output neurons will be summed to determine from which section of the sample to play back grains from. As an example, if there are 1000 binary output neurons, when 500 are 1s and 500 are 0s, the output of the model would be considered 50% time offset in the target sample. Since the resolution is limited, grains will be played back from around the range specified by the network's prediction. I have chosen to limit the resolution this way just based on the capabilities of the models I've read about, if there is a way to represent time with more detail than thermometer format I would be open to using that as well.

In order to feed the model enough data, vocal data will be analyzed and connected to its most similar timbral points in the sample using algorithms found in concatenative synthesis. This synthetic data can be used to condition the network for vocal inputs. Ideally the inclusion of this synthetic data will allow for vocal interaction from the user to navigate the granular synthesis engine. By synthetic data, I am not referring to synthesized vocal sounds, but rather vocal sounds that have been prepared in the format of MFCCs, and tied to their most similar grain in the target sample. Concatenative synthesis hosts a family of techniques by which individual grains of audio can be analyzed, compared and connected to each other based on similarity [2]. Training the model to replicate this process for inputs with features far outside those of the target sample could allow for an instrument that is better conditioned for any user input, but here I think focusing on vocal inputs would be most relevant.



Resynthesis: Resynthesis will rely on the neural network to determine which sample to play back. The user should be able to record a sample from which to apply a style transfer, or even use live input to the device. When preparing the training data and collecting input on the instrument, the pitch of each grain should be determined. While triggering grains, they can be resampled to match the pitch of the input (ideally relatively close). I would start by just resampling to match pitch between instrument input and model output, but there's room for exploration of frequency domain techniques both for matching pitch and timbre. A variety of controls could be added if time permits, giving the user access to familiar granular synthesis controls.

Potential Controls: - Play Speed: this would control the rate at which the instrument plays through user recorded loops. Since the MFCCs from performance are translated to grains from the sample, slowing or raising the speed at which we move through the control audio could allow for a variety of creative uses.

- Mix: Allow the user to mix in the sound of the recorded audio that is controlling the synth.
- Grain Size: change the size of grains played back by the resynthesis engine.
- Grain Trigger Rate: change the rate at which control audio is polled for input to the model (and used to trigger grain output from the synthesis component).
- Pitch Correction: control the degree to which grains are stretched to match the pitch of control audio.
- Randomness: control the amount of randomness applied to the time offset at which grains are triggered.

Goals: 1. Granular style transfer network using differential logic gate networks

2. Handheld instrument for navigating the network using audio input

Related Work: Here are a few similar instruments and pieces of software. https://gitlab.com/then-try-this/samplebrain https://dillonbastan.com/store/maxforlive/index.php?product=coalescence https://learn.flucoma.org https://github.com/ircam-ismm/catart-mubu https://www.xlnaudio.com/products/xo

2.2 Software

Python C++ Max/MSP

2.3 Hardware

Bela board Knobs / Buttons microphone Audio I/O

Chapter 3

Timeline

Provide a development timeline with 10 milestones, one for each week of development without an evaluation, and any pre-work. (A realistic, measurable timeline is critical to our selection process.)

Note: This timeline is based on the official GSoC timeline

3.1 Timeline summary

Date	Activity
February 26	Connect with possible mentors and request review on first draft
March 4	Complete prerequisites, verify value to community and request review on second draft
March 11	Finalized timeline and request review on final draft
March 21	Submit application
May 1	Start bonding
May 27	Start coding and introductory video
June 3	Release introductory video and complete milestone #1
June 10	Complete milestone #2
June 17	Complete milestone #3
June 24	Complete milestone #4
July 1	Complete milestone #5
July 8	Submit midterm evaluations
July 15	Complete milestone #6
July 22	Complete milestone #7
July 29	Complete milestone #8
August 5	Complete milestone #9
August 12	Complete milestone #10
August 19	Submit final project video, submit final work to GSoC site and complete final mentor evaluation

3.2 Timeline detailed

3.3 Community Bonding Period (May 1st - May 26th)

GSoC contributors get to know mentors, read documentation, get up to speed to begin working on their projects

3.4 Coding begins (May 27th)

3.5 Milestone #1, Introductory YouTube video (June 3rd)

Explore options for concatenative synthesis (i.e. how to prepare synthetic data). Get acquainted with training DiffLogic networks.

3.6 Milestone #2 (June 10th)

Prototype DSP, granular synthesis options (maybe in Max). Start the granular synthesis engine.

3.7 Milestone #3 (June 17th)

Generate training data from target samples, generate synthetic data from vocal training data.

3.8 Milestone #4 (June 24th)

Train networks and get them on the board.

3.9 Milestone #5 (July 1st)

Create an interface for training models based on user supplied style targets.

3.10 Submit midterm evaluations (July 8th)

Important: July 12 - 18:00 UTC: Midterm evaluation deadline (standard coding period)

3.11 Milestone #6 (July 15th)

On board granular synthesis engine.

3.12 Milestone #7 (July 22nd)

Develop recording / playback interface.

3.13 Milestone #8 (July 29th)

Add additional parameters and functionality to the synthesis engine.

3.14 Milestone #9 (Aug 5th)

Figure out extra controls, inputs / outputs for the board.

3.15 Milestone #10 (Aug 12th)

Assemble the instrument and add finishing touches.

3.16 Final YouTube video (Aug 19th)

Submit final project video, submit final work to GSoC site and complete final mentor evaluation

3.17 Final Submission (Aug 24nd)

Important: August 19 - 26 - 18:00 UTC: Final week: GSoC contributors submit their final work product and their final mentor evaluation (standard coding period)

August 26 - September 2 - 18:00 UTC: Mentors submit final GSoC contributor evaluations (standard coding period)

3.18 Initial results (September 3)

Important: September 3 - November 4: GSoC contributors with extended timelines continue coding

November 4 - 18:00 UTC: Final date for all GSoC contributors to submit their final work product and final evaluation

November 11 - 18:00 UTC: Final date for mentors to submit evaluations for GSoC contributor projects with extended deadline

---

Chapter 4

Experience and approch

I have previously used pytorch to train small neural networks for granular synthesis. My college capstone used simple audio features from analyzing individual grains to try to rearrange source samples into new compositions. alecdenny.wordpress.com

I have also manipulated small convolutional image synthesis models for a personal project. dropbox.com/scl/fi/mmnkrnr

I am currently building the interface for a VST plugin in C++ through which I've gained experience with real time audio code and musical interface design. This project is a collaboration with Nathan Blair. dropbox.com/scl/fi/w75xzhwy

I have experience with spectral processing and synthesis. I am currently working on a python framework for spectral synthesis that mimics the structure of 3d graphics shaders called WavKitchen. github.com/alecdenny/wavkitchen

I make audio / visual art and have coded a variety of visualizers and interfaces for performance which I got the chance to use all over the US last year.

4.1 Contingency

I studied music technology in a program that focused more on composition and performance, but have taught myself most of what I know about machine learning, C++ and digital signal processing. If I get stuck, I know I have experience bringing myself up to speed on these topics, and know where to find resources for them.The main thing I see myself getting stuck on is programming the board itself, I have limited experience with hardware though I have used arduino before in school.

4.2 Benefit

Generative AI is a major topic in music software / hardware right now. One of the main constraints for developers right now is the feasibility of running or training models for real time audio applications. Differentiable logic gate networks offer a novel approach to shrinking models for these use cases. One remaining concern I had after reviewing the literature was the ability to generate high resolution outputs from the model, or represent continuous numbers with accuracy and trainability. This network architecture is an approach that pushes the potential uses for AI on hardware into the realm of generative synthesis in a way that elides some of these difficulties.

For creatives using AI instruments, the challenges are in interfacing with AI, as well as originality. AI can shrink very complex synthesis algorithms down to a handful of controls, but can struggle to come up with easily name-able parameters, or provide the user with too many parameters to possibly feel comfortable using them. For this project, the control interface is primarily the user's voice, but potentially any audio or sound in their immediate environment. Because the synthesis engine is not entirely AI, familiar granular synthesis controls are also available, adding a level of familiarity to the instrument. Originality with AI instruments poses a challenge for artists when massive datasets

are required for one to work, or outputs could contain copyrighted material. With this model, artists could train models themselves, with no potential for accidental infringement.

The potential benefits for this project would be a working library for generative AI synthesis on the beagle board, as well as an exploration of the potential use cases for small neural networks in electronic instruments. The benefits of this project are more geared towards the NIME and music AI communities than specifically the beagle board community. However I think creating this instrument and allowing users to train models and upload them to the board could yield a low commitment entry point for those involved in these communities to be introduced to beagle boards.

This project would yield a reusable neural network architecture, the synthesis component could be greatly extended. Using frequency domain pitch shifting or cross synthesis techniques could build a variety of instruments on top of the framework of this project. Creating an interface to prepare data and train models for use on the board could also prove useful for other projects.

4.3 References

1. F Petersen, C. Borgelt, Hilde, H. Kuehne, O. Deussen. Deep Differentiable Logic Gate Networks. *Neurips.* [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/0d3496dd0cec77a999c98d35003203ca-Paper-Conference.pdf

2. S Diemo, “Concatenative Sound Synthesis: The Early Years,” *Journal of New Music Research*, vol. 35, March, 2006. [Online]. Available: https://hal.science/hal-01161361/document

4.4 Misc

to do : have not made merge request yet