

Tremolo FX Pedal

Final Project Report
Making Embedded Systems, Fall 2024

Elliott Miller

APPLICATION DESCRIPTION

The following paragraphs and diagrams describe a tremolo audio effect processor built using the Electro-Smith “Daisy” platform. The continued goal for this project is to mount the hardware into a guitar pedal format with low noise supporting analog circuitry. Currently, the system consists of the following hardware features:

- 24bit, 96kHz WM8731 CODEC with input and output buffer circuitry
- 3x encoder parameter controls
 - rate
 - depth
 - shape / pwm
- 1x push button control
 - pwm select
- 10pos LED bar graph meter with multi-state display

The software allows the user to select multiple wave-shapes for virtual VCA control. Each shape can be manipulated in the time and level domain via “shape,” “depth,” “rate,” and “pwm” controls:

SHAPE - user may choose between *sine*, *triangle*, *saw*, *ramp*, and *pwm* waveforms

DEPTH - adjusts the scaling of the control waveform

RATE - adjusts the frequency of the control waveform

PWM - adjusts the control waveform duty cycle in *pwm* shape mode (accessible by pressing the shape / pwm encoder button)

HARDWARE DESCRIPTION and SCHEMATIC

This project uses the Daisy Seed embedded audio platform as the central “hub” of communications and control. The board features a Cortex M7 processor, integrated CODEC, and debug headers with support for a Virtual COM Port via USB.

Peripherals:

The onboard CODEC receives commands via I2C and writes audio over I2S. The samples are handled by DMA and passed into a ring buffer for user access and manipulation.


10 LEDs display the CV waveform and parameter values. An Adafruit breakout for Texas Instruments TLC59711 communicates to the Seed via SPI to drive the LEDs.

3 Rotary Encoders are connected via GPIO.

STMicroelectronics Virtual COM Port is used for CLI and debugging using a single UART on the mcu.

A rudimentary shorting switch for the “pwm” button is made from a jumper wire, and a small capacitor to debounce the wire / breadboard contact.

So far this has been powered over USB with some supply filter caps added to the breadboard. In the future I’ll use a standard center negative 9v brick supply and use the on-board regulation.

Schematic:  [Daisy Tremolo.pdf](#)

Datasheets: [Daisy Seed](#), [TLC59711 PWM LED DRIVER](#)

SOFTWARE DESCRIPTION

Audio processing:

All audio processing is handled in a callback function that is triggered upon reception of new input samples. The function multiplies the input sample by an oscillator “control voltage,” and places the result in the output stream buffer.

Parameter control:

The main loop polls encoder objects for changes in state. A +1 indicates clockwise rotation, a -1 indicates counter-clockwise rotation, and 0 indicates no change. Parameter values for the oscillator control are updated according to the encoder state in each loop. The pwm button triggers an interrupt to set the encoder control state.

Bargraph meter:

The bargraph meter is updated each loop according to a “state” variable. (see State Table in *Architecture Diagrams*). The state variable indicates which virtual cv should be read to update the LEDs. The possible options are

DEFAULT - displays the oscillator control voltage

RATE - displays the oscillator frequency from 1-10Hz (1Hz / led, .5Hz steps)

DEPTH - displays the oscillator scale multiplier from 1 to 10

SHAPE - displays a two-LED indicator according to oscillator “shape” selection

PWM - triggered by button interrupt, displays the PWM percentage for the “square” oscillator waveform from 10-90% (10% steps).

Command Line:

The command line allows user input to set the oscillator cv parameters, as well as log control / state changes. Supported commands are

rate <int> sets the “rate” parameter to <int>

depth <int> sets the “depth” parameter to <int>

shape <int> sets the “shape” parameter. Each shape is assigned an <int> 0-4

pwm <int> sets the “pwm” parameter to <int>

Logging shows control updates for each encoder, including pwm state, and a “0” or “1” mode to indicate pwm or shape control.

CODE ORIGINS

TREMOLO PROJECT GITHUB REPO: <https://github.com/elliottohmiller/tremolo>

Much of the code uses the Daisy platform's *LibDaisy* and *DaisySP* libraries for hardware abstraction, DSP classes, and platform specific functions. *stmllib* includes some lower level control and peripheral setup functions. All three of the aforementioned libraries are included in the *Daisy Examples* functional pipeline at <https://github.com/electro-smith/DaisyExamples>. Much of *LibDaisy* and *stmllib* utilizes STMicroelectronics HAL, LL, and CMSIS APIs - some of which are called directly in the bespoke tremolo code.

Interrupt.cpp/hpp were ported from stm32duino here:

https://github.com/stm32duino/Arduino_Core_STM32/blob/main/libraries/SrcWrapper/src/stm32/interrupt.cpp#L209

Elicia's *reusable* console integration was experimentally added to my fork of *libDaisy* and used for CLI implementation. Edits were made to support added commands and VCOM i/o. (Find in `libDaisy/src/reusable`)

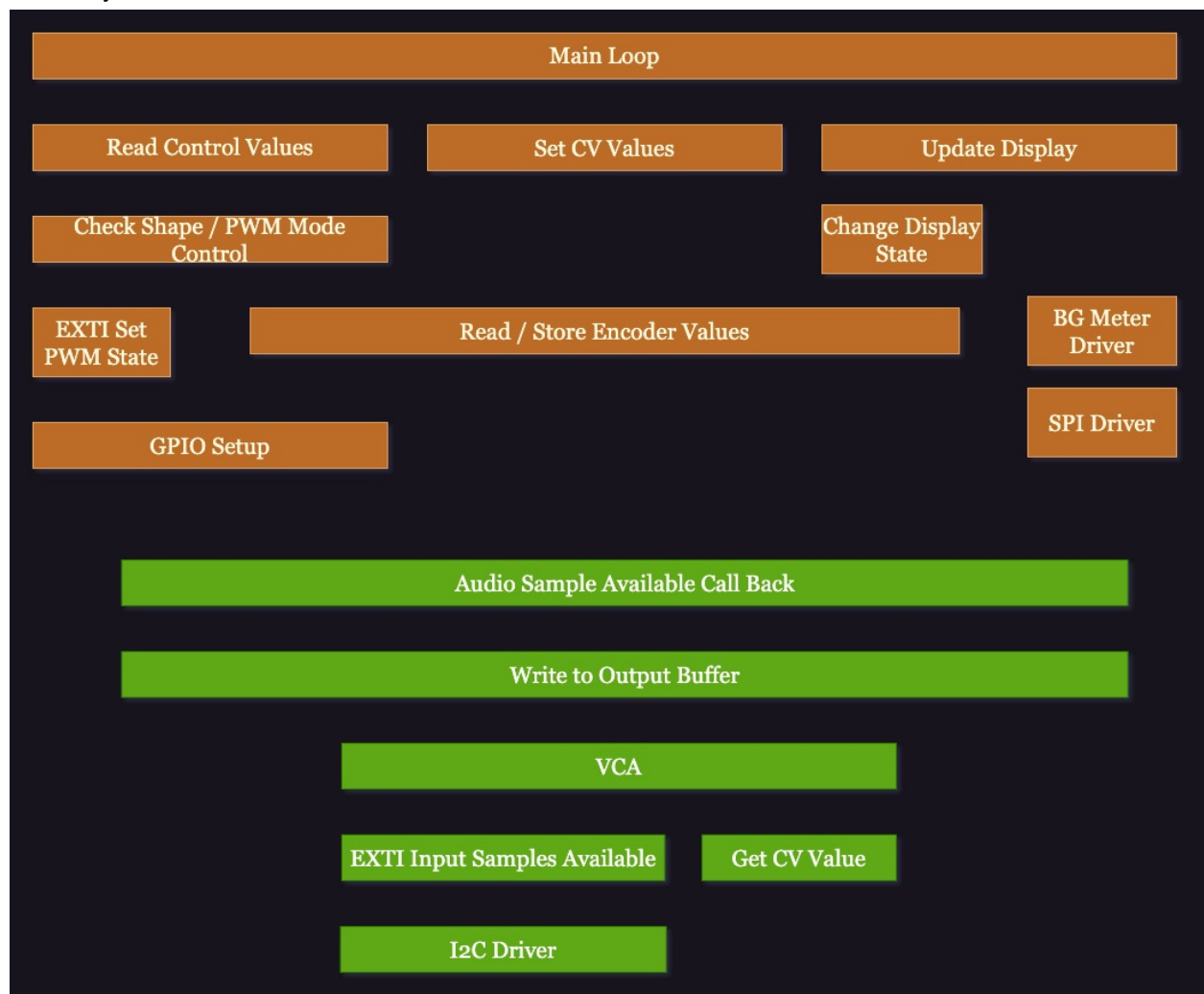
bgMeter.cpp/hpp is largely original with inspiration from Adafruit's TLC59711 library:

https://github.com/adafruit/Adafruit_TLC59711

main.cpp/hpp are entirely original.

ARCHITECTURE DIAGRAMS

Hierarchy of Control



LED State Table ([Link](#))

State	Action	LED Display	CC ++PWM / SHAPE Enc	CC -PWM / SHAPE Enc	CC ++Rate Enc	CC -Rate Enc	CC ++Depth Enc	CC -Depth Enc	CC PWM Button	CC Timeout / End Sequence
START	System Init (Clocks, Timers, Peripherals, Interrupts)	off	x	x	x	x	x	x	x	LOAD_LAST_STATE
LOAD_LAST_PARAMS	Read last state from memory and set parameters accordingly - to be	off	x	x	x	x	x	x	x	DEFAULT
DEFAULT	idle state	bargraph meter displays CV	SHAPE_CHANGE_PARAM	SHAPE_CHANGE_PARAM	RATE_CHANGE_PARAM	RATE_CHANGE_PARAM	DEPTH_CHANGE_PARAM	DEPTH_CHANGE_PARAM	PWM_CHANGE_PARAM	x
SHAPE_CHANGE_PARAM	set new oscillator CV shape	bargraph meter displays 2 LEDs corresponding to CV shape selection	x	x	RATE_CHANGE_PARAM	RATE_CHANGE_PARAM	DEPTH_CHANGE_PARAM	DEPTH_CHANGE_PARAM	PWM_CHANGE_PARAM	DEFAULT
PWM_CHANGE_PARAM	adjust PWM percentage	bargraph meter displays value 10:90	x	x	RATE_CHANGE_PARAM	RATE_CHANGE_PARAM	DEPTH_CHANGE_PARAM	DEPTH_CHANGE_PARAM	SHAPE_CHANGE_PARAM	DEFAULT
RATE_CHANGE_PARAM	adjust rate (number of cycles per 48000 samples)	bargraph meter displays value 1:10	SHAPE_CHANGE_PARAM	SHAPE_CHANGE_PARAM	x	x	DEPTH_CHANGE_PARAM	DEPTH_CHANGE_PARAM	PWM_CHANGE_PARAM	DEFAULT
DEPTH_CHANGE_PARAM	adjust CV scaling parameter	bargraph meter displays value 1:100	SHAPE_CHANGE_PARAM	SHAPE_CHANGE_PARAM	RATE_CHANGE_PARAM	RATE_CHANGE_PARAM	x	x	PWM_CHANGE_PARAM	DEFAULT

DEBUGGING

Debugging is achieved with the help of an STLINK V3MINI (Datasheet), and the CORTEX DEBUG extension for VSCode.

BUILD INSTRUCTIONS

The majority of the board bring-up and debugging used the Daisy platform's CI tools and build scripts. To replicate, follow the instructions in the README at the top-level repo(<https://github.com/elliottohmiller/tremolo>). There are small differences to the makefiles to include edits to libDaisy, but this shouldn't need user configuration. Programming can be done through the web based programmer (<https://electro-smith.github.io/Programmer/>) or with STLINK and VS Code using the build scripts in the tasks.JSON file.

In general, the toolchain consists of

GCC

Make

Cmake

Python3

CORTEX-Debug VS Code Extension

FUTURE

There's a lot that can be improved here, and with improvements to peripheral hardware I expect to make many changes.

I'd like to expand the CLI for bargraph meter debugging. Code can be broken up for modularity, and error reporting added.

There is some noise from the analog / digital circuitry proximity, some of which I believe can be fixed in firmware.

Currently the system resets upon power cycle, so I'll be adding access to the onboard flash memory to reload the pre-power-cycle state.

Plenty of hardware details still need to be implemented (case, PSU, analog i/o)

Future development will also add a kit-style BOM and build instructions for DIY communities.

POWER ANALYSIS

The whole system draws around 150mA of current on a 5V bench supply. Power to the LEDs and LED driver board is tapped from the Daisy Seed's on board 3V3 supply, with a roughly 15mA draw at worst case (depth: min, rate: max, shape: square, pwm: 90%) and 6mA at best case (rate param being set to min). Average draw is around 12mA over extended use.

GRADING and SELF-ASSESSMENT

Criteria	Self Grade (1-3)	Notes/Comments
Meets Minimum Goals	2.25	All project goals are met, with logging and additional commands added to CLI. Bargraph meter requires a somewhat complex state machine
Completeness of Deliverables	2.25	Reasonably thorough report. A more descriptive build and toolchain section would help the reader understand the build process. Find in the repo README for more info.
Clear intentions and working code	2.25	Some modularity for easier to follow code. The project performs each function as intended. Improvements could be made overall to modularity and error reporting
Reusing Code	2	Code is mostly separated, but licensing may be unclear or incorrect
Originality and scope of goals	2.4	It's a tremolo pedal...so examples abound. However, bargraph cv and parameter readout is pretty neat. Also pulse width options aren't particularly common. A lot of care was taken to make the U/I comfortable with parameter value readout and timeout for default cv display state.
Power Analysis	2	Measured and described; overall current draw and peripheral current draw
Version control was used (bonus)	2	Yes! Somehow this was the most difficult thing to learn, but eventually I was making regular commits and even updating submodules. As a result, I'm sure the commit history looks sporadic. Still a lot to learn here! https://github.com/elliottohmiller/tremolo/commits?author=elliottohmiller

