Bruno Luiz da Silva 150724708

Drum Machine RT-DSP Project

Introduction

This project consists on developing a drum machine using BeagleBone Black (BBB) with Xenomai kernel. We used BELA, a BBB environment focused on real time applications.

The drum machine do not work with buttons or sliders, but with an accelerometer. Depending on the board orientation, the machine will play a different drum pattern. The only orientation that don't play a different pattern is when the board is turned upside down, which makes it play the previous pattern in reverse. The board have the "tap to fill" feature implemented on it, as a bonus feature.

Development and Implementation

The implementation followed the sent guidelines, including the bonus one. As an early note, in this report and at the code, a slot is a reference to a readPointer + gDrumBufferForReadPointer + output set and, as the draft states, the machine can only play 16 multiple drums at the same time, so it just enables a maximum of 16 slots.

There are two main functions in BELA: render() and setup(). In the setup(), the program:

1. Setup the buttons and the LED pins
2. Initialize some variables with the default values (mostly -1)
3. Instantiate the High Pass Filter (used for the bonus step)

The render() is quite longer, mainly because the functionalities of the program depend on it. The function follows this workflow:

1. Check if the button is pressed and then toggle the playing state
2. Check the value of the potentiometer and update the timeout interval for the next event
3. Check if there was any tap on the board (if no tap was detected before)

    a. It always waits 100 samples until the first check. Mainly because the parameters of the filter needs to be normalized. Otherwise the drum machine will always get a “tap” at the beginning of the execution

4. If no tap was detected or being executed (gShouldPlayFill = false), check the orientation of the board
5. Check if the timeout interval was already reached. If it is, start the next event
6. Execute the state machine that executes the drums of the actual event (up to 16)
7. Output the result, getting all the 16 slots, summing up and dividing by 16

As is possible to conclude, almost everything inside the render() are check-ups which will influence the way that the drums will be played. The only exception is the output (already explained above) and the state machine. The state machine is used inside a loop that checks all 16 slots, where each slot can have an assigned drum to play.

The first state (wait state) checks if the read pointer and the drum buffer are not set to -1, where -1 mean that they would be empty. If they are not, then it goes to the play state. When the read pointer reaches the drum buffer length, it goes to the end state, which will do some cleanups and then go to the beginning again.

Bonus feature

The bonus feature was implemented using a butterworth 2nd order filter, which is implemented at the Filter class. To use the class, the cut-off frequency and the filter type (high or low pass) have to be specified. After that, the developer just need to call the run() function, passing the actual samples to it.

The high-pass filter is used because, as the tap will generate only a few high value samples on the accelerometer output in a small fraction of time, this will generate a high frequency component on the signal. This high-pass filter is using a 5kHz cut-off frequency, which was chosen after a try-error session.

To get the tap, a function called getBoardTap() (implemented on utils.cpp) is called. Inside of it, the program checks the board orientation and then read the respective axis, passing the accelerometer outputs to the filter. After passing the high-pass filter, it goes to a comparator, where the threshold is 0.2 (got in a try-error session, too). If it is higher than this value, then the function returns true and this will trigger some variable setups at render(), which will lead to the fill playback.