Real-Time DSP Assignment 2: Drum Machine

Kranthi Yanamandra (170797647)

Overview:

The aim of this assignment was to implement a drum machine by playing different drum samples present in preloaded buffers. An accelerometer was used to detect the orientation of the breadboard such that each orientation plays a different drum pattern. Other components of the circuit include a potentiometer to adjust the tempo of the drum pattern, an LED that blinks in time with the selected tempo and a button that starts and stops playback. The following sections illustrate the design, implementation and state diagram of the drum machine.

Design and Implementation:

General structure:

The design approach adopted for the drum machine follows the instructions provided in the assignment brief. In the *setup()* function, the GPIO pins are initialised for the button and LED and the filter coefficients are set to implement a high pass filter at a later stage for tap detection. In order to keep track of the states of the button (on/off), the LED (blinking/stationary), and the number of samples that have gone by, their corresponding state variables are declared and initialised to 0. 16 read pointers are defined to be used to traverse through drum samples. Inside the main for loop of the *render()* function, the button state is read and a check occurs to know whether the button is pressed or not. This state is then preserved for the next call to render. All the audio processing code goes into the *render()* function.

Accelerometer calibration and reading:

A function called *updateOrientation()* is defined which checks for the orientation of the accelerometer depending on two threshold values (low and high thresholds). It checks the states of the x, y and z axes against these threshold values and if the accelerometer is facing upside down, the drums start to play backwards. The voltage values of the three axes of the accelerometer are read for the first 10 runs of the *render()* function to be used as a reference. A second order high pass butterworth filter was implemented to detect taps on the board to play a drum fill. By filtering out low frequency data, a static threshold was set to detect sharp movements in the z axis. The filter coefficients were adopted from the code written for assignment 1.

The voltages of the three axes on the accelerometer are calculated as a difference of the analog input readings of each of its x, y, and z pins and the reference voltage mentioned earlier. Then,

the magnitude of the acceleration is calculated as the square root of the sum of squares of each of these voltages. The filtered acceleration (to filter out gravity) is then calculated by implementing the standard transfer function of an IIR filter using the previously calculated filter coefficients. To implement a drum fill when the board is tapped, a check is performed to see if the filtered acceleration is above a specified threshold. If so, the drum fill flag is set to 1. To read each axis and to know the board's orientation, the accelerometer is checked 128 times per second and the *updateOrientation()* function is invoked.

startPlayingDrum() and startNextEvent() functions:

In the *startPlayingDrum()* function, an available read pointer is searched from the 16 read pointers. In the *startNextEvent()* function, the *startPlayingDrum()* function is called to start playing the indexed drum sound. The index is set to 0 again when it reaches the end of drum pattern. Also, if the current pattern being played is a fill pattern, it is updated for the next iteration and the variable that keeps track of fill pattern is set to 0.

Potentiometer mapping and LED toggling:

To fit the full range of the potentiometer values to an interval range of 50 - 1000 ms, the *map()* function was used. The output was stored in the variable *gEventIntervalMilliseconds*. After checking that enough samples have passed, the *startNextEvent()* function is called and the sample counter is reset to 0. The convert from the interval in milliseconds to the number of samples needed, the time interval is multiplied by the sampling rate. At this point, the state of the LED is checked and is toggled according to the selected tempo. If the pattern has to be played backwards, the read pointers are counted downwards instead of upwards and when its value reaches 0, the samples are sent to the output buffer. The output is multiplied by 0.25 to prevent clipping.

State Diagram Explanation:

The state diagram of the drum machine is illustrated below. The 'Toggle Active' and 'Toggle Inactive' here refer to the button being in ON and OFF states respectively. The drum machine is in a 'Not Playing' state initially when the button is not pressed and continues to be so till it is pressed. Once it is pressed i.e. the toggle is active, the tempo 'T' is set to the corresponding potentiometer value. Next, the orientation of the board is checked and if it is flat/left/right or front facing, the drum pattern for those particular orientations are played respectively at the previously selected tempo 'T'. If the board is upside down, the drum pattern is played backwards at tempo 'T'. During both these states, the toggle value is checked continuously. If it is active, it keeps checking for the tempo. If it is inactive, it transitions to the 'Not Playing' state. The following figure illustrates the state diagram of the drum machine.

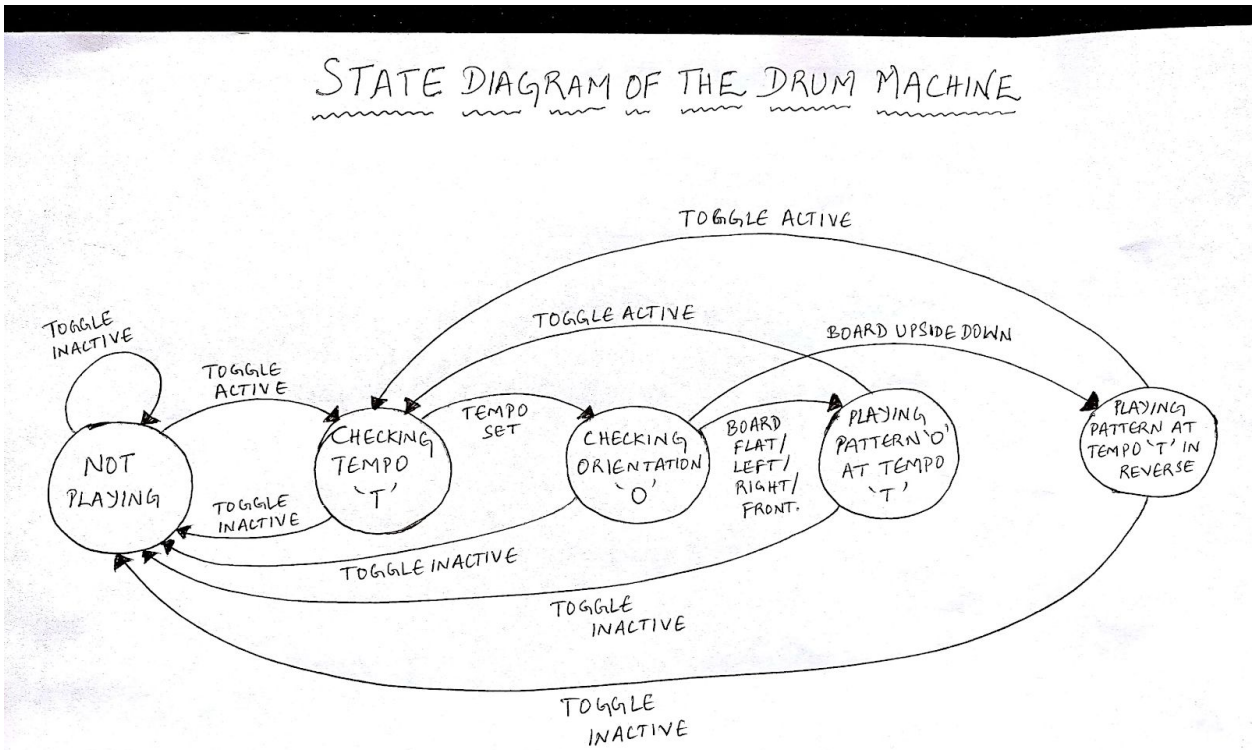

Figure: State Diagram of the Drum Machine

References:

1. Will Pirkle, 'Designing Audio Effect Plugins With C++'.

2. Dimension Engineering - A beginner's guide to accelerometers.
https://www.dimensionengineering.com/info/accelerometers

3. Accelerometers - Measuring Acceleration and Tilt
https://wpilib.screenstepslive.com/s/currentCS/m/cpp/l/241870-accelerometers-measuring-acceleration-and-tilt

4. Lee Copeland - State Transition Diagrams
https://www.stickyminds.com/article/state-transition-diagrams