Stringy

from an original design by James Hutchby

Stringy is a guitar synthesiser, a remix of the MadLab kit *Funky Guitar*, which uses a cheap 8-bit PIC microcontroller and a software model of a plucked guitar string. It is a minimal and somewhat lo-fi design.

The keyboard plays the twelve sharps, flats and natural notes in an octave. The two pushbuttons shift the range of notes up or down by an octave over five octaves in total. If both pushbuttons are pressed and released then the voice is toggled between an acoustic and an electric-sounding guitar. If both pushbuttons are pressed and held down for about a second then a demo tune is played (in the current voice).

Hardware

The keyboard consists of a ladder of 12 resistors connected in series with 5V and ground at opposite ends thus forming a potential divider. The stylus taps a voltage in this ladder between 5/12V and 5V in 5/12V increments. The stylus is connected to an analogue input pin on the processor which digitises the voltage and maps it to one of 12 notes.

A resistor pulls the voltage on the stylus to ground when no note is being touched. Thus a total of 13 discrete voltages appear on the analogue pin allowing the firmware to differentiate between the 12 notes and no note. The pull-down resistor is much larger than the ladder resistors so has minimal effect on the measured voltage.

Metal film resistors of 1% tolerance are used in the ladder to ensure the firmware can reliably differentiate between the 12 notes.

An NPN transistor is used as an output driver. The transistor is driven by a 5V square wave so is fully saturated and effectively acts as a Class D amplifier with unit gain.

A low-pass filter is connected to the output from the driver stage to remove the quantisation artefacts from the output signal. This is a simple passive RC filter but is sufficient for the lo-fi minimal design.

The low-pass filter also attenuates the pass band to some extent which limits the output volume to a comfortable level for headphones.

A 220u electrolytic capacitor protects the headphones by blocking any DC.

RA0 (pin 7) is unused and is fed to the header pads with a pull-down resistor to ground. This is available for adding extra features to the firmware. Perhaps a MIDI or UART input for driving the board with an external sequencer. Or perhaps connected to a variable resistor to control the rate of decay of the notes.

Firmware

The firmware implements the Karplus-Strong algorithm as a means of synthesising realistic plucked-string sounds. This is computationally cheap so is achievable with a modest microcontroller. The principal characteristic of a plucked-string sound is that the higher frequencies decay faster than the lower ones. This is implemented in the firmware with a delay line, initially filled with random values, and a low-pass filter (the simple averaging of successive pairs of values in the delay line).

There is no digital-to-analogue hardware in the microcontroller but there is a hardware pulse-width modulation (PWM) module. A PWM will serve the same purpose as a DAC if the PWM frequency is high enough.

A range of 256 duty cycles is available corresponding to an 8-bit output resolution.

The pitch of the output note is both determined by the length of the delay line and the PWM frequency. For low pitch notes (the first three octaves) the maximum length of delay line is used (128 bytes) with the PWM frequency gradually increasing as the pitch increases. At a certain point the maximum PWM frequency is reached (31250Hz, limited by the maximum CPU clock speed). For the last two octaves the PWM frequency is kept at the maximum and the length of the delay line is reduced instead.

Refer to the spreadsheet for the full set of delay line lengths and PWM frequencies over the five octaves.

The firmware is interrupt driven to ensure accurate timing with an interrupt service routine (ISR) updating the PWM parameters and executing the low-pass filter.

A pair of multiplications is required in the ISR. Because the microcontroller doesn't support hardware multiplication this is implemented in software, and the software multiplier needs to be efficient as time is at a premium in the interrupt handler. A couple of tricks are used to speed up the multiplier. Firstly, the loop is unrolled which avoids the overhead of a loop counter and branch instruction (at the expense of more program memory). Secondly, because the multiplication is by a constant (i.e. known at compile time), iterations of the loop with a corresponding multiplicand bit of zero can skip the addition without a run-time bit test.

A compact format is used to represent the demo tune in program memory. One byte represents a single note with bit-packed fields for the note value, its duration and the rest period after the note. Fifteen notes can be expressed directly in a 4-bit field, with escape codes that allow for the range to be shifted up or down by one octave. Further escape codes allow for longer rests and the ability to call and return from a sub-tune. A set of symbolic constants and helper macros simplify the process of encoding a tune.

About 75% of the program flash memory is unused and is available for longer or multiple tunes.

E: james@madlab.org

W: www.madlab.org

T: @clubmadlab