Lockdown Project Number One – The 695 Waffle Meter

Lack of modulation depth is a bit like snoring – you never hear it yourself, so it falls to others to tell you about it.

The lockdown net has been great fun, but it usually fell to Grant to point out to the miscreants, including yours truly, that their modulation depth was poor.

We lamented the lack of a "smart microphone", one that gave a red light when the modulation was too low, and green when things were good.

Well… here it is.

I hope the project is interesting for anyone who is getting started with MCUs like the Arduino. There are many resources online to show you how to get an Arduino to blink a LED, and many to help you solve advanced problems that are usually components of a bigger project. This project aims to fill part of the gap between – it is a small, fully functioning real time signal processor that can be used as the base of larger projects.

The traditional method for indicating modulation depth is to drive a VU meter from somewhere in the audio signal path. It is difficult to open a Bowfinangledangle Handheld (or equivalent) let alone find the right part of the signal path. It is easier to drive an indicator direct from the microphone.

A very elegant Sound Level Meter can be found here: http://www.technoblogy.com/show?ZU7

I decided to take a slightly different path for several reasons. This was a lockdown project, so it had to be made from parts on hand. The Technoblogy project uses a rather nice bar graph display, but I did not have one. The input impedance may also be lower than is desirable, it was hard to know. Finally, the Technoblogy code is very elegant but involves direct manipulation of the processor registers, so is not the best example for these purposes.

All good reasons to try a different, and perhaps slightly more traditional, approach.

Diagram one shows the circuit diagram. U1A is used as a preamplifier and a bit of a low pass filter. If you are interested in the analysis of the circuit it is generally based on a design from the TI Analog Engineer's Circuit Cookbook pg 274 (see http://www.ti.com/design-resources/design-tools-simulation/analog-circuits/overview.html) . Being a lockdown project, most of the components and the op amp are a little different, but you will get the general idea. UA2 is just a useful amplifier because the part I used had two op amps. Only one is really needed.

The output of the amplifier is fed into the ADC input of the ATTINY85. R7 needs to be adjusted so that with the microphone you are using the voltage into the ADC swings between both supply rails.

The ATTINY samples the ADC approximately every 200uS, or at 5kHZ.

The other pins on the ATTINY are connected to two LEDs (red and green) and to two switches (SET and PTT). Provision is also made to connect a serial port instead of switches, but this option is not used for now.

I have made a PCB for the design. The PCB is laid out so that the switches, LEDs and microphone can be mounted on the board or wired to the board. The board is yellow because I wanted to see what this colour looked like.

The Eagle files and pdf of the schematic have been loaded onto GitHub at https://github.com/JohnGENZ/695-Waffle-Meter .

The signal processing is carried out by the ATTINY software.

When the SET button is held down (PTT can be open or closed), both lights flash and input samples are continuously recorded to compute and store a "target value", referred to in the software as the GSI. When the PTT button is pushed, the ADC input is continuously sampled, and a computed value is compared with the stored GSI. If the input value exceeds the GSI the green LED goes on. If the input is just below, but close to the GSI, neither light goes on. If the input is significantly below the GSI, then the red light goes on. The GSI is stored in EEPROM, so once set it will be used until it is set again.

So, in use, start by pushing PTT and SET. Ask someone if you sound OK – if yes, then the GSI is set correctly, otherwise, have another go. After that, just speak loud enough to keep the green light on while PTT is pushed. What does GSI stand for? It is the "Grant Satisfaction Index".

The software is a very simplified cooperative scheduler. In a true real-time operating system (RTOS) the processor is interrupted to switch between tasks (known as context switching). Context switching carries an overhead of processor resources (memory and time) and also requires programming disciplines that acknowledge that the context switch can occur at any time. The most popular MCU RTOS is FreeRTOS. FreeRTOS is an exceptionally powerful system, but it also has a significant learning curve.

Our cooperative scheduler is much simpler than FreeRTOS. It has a small footprint of time and memory and so can be used on very small processors. For those who are familiar with MCUs and RTOS systems, to lighten the weight, this version has been stripped of both inter process queues and semaphores. It only executes integer arithmetic. The software data structures have also been simplified for clarity.

When using a cooperative scheduler, an application is divided into a set of more or less independent tasks. Each task is short, simple to write, and simple to maintain.

All tasks need to follow certain rules, in other words, to cooperate. The most important of these rules is to avoid long loops or delays. Instead of looping or delaying, tasks must yield to the operating system and start again later.

This system has five tasks and a timer interrupt service routine (ISR). Only three of the tasks are used in the "live" system. The running tasks read the buttons, flash the lights, and update the GSI values while the timer ISR reads the ADC. The additional two tasks are diagnostic tasks that are not normally enabled. One diagnostic task flashes the LEDs. It is useful for debugging the hardware. The other diagnostic task is called "InsideOut". It sends internal data values to the LED pins one bit at a time. It can be used with a logic analyser to examine internal data values and to debug the software.

To follow the operation of the software, look at the files loaded onto GitHub. Start with BiscuitV280.ino, BiscuitKernelData_V280 and BiscuitKernelCode_V280. Why Biscuit? It is a long

story involving a packet of TimTams. If there is interest in how to use this software, I will cover it in more detail later in the year.

This project has a working prototype and a PCB. Later in the year, I will work on populating the PCB, 3D printing a housing for the microphone and maybe enhancing the software. The calculation of the GSI is still quite primitive, and there is scope to improve the sample rate by more aggressive optimisation of the ISR, but these are all jobs for another day right now. I am very happy to share with anyone who is working on something similar.

John Errington, ZL3TIL