A Multitone Current Sink For Measuring Impedance of In-Use Batteries

Christopher Dunn
Faculty of Science and Engineering
University of Waikato
Hamilton, New Zealand
cjdunn@xtra.co.nz

Jonathan Scott
Faculty of Science and Engineering
University of Waikato
Hamilton, New Zealand
scottj@waikato.ac.nz

Vance Farrow
Faculty of Science and Engineering
University of Waikato
Hamilton, New Zealand
vf14@students.waikato.ac.nz

***Abstract*—Measurement of impedance of rechargeable batteries at very low frequencies reveals valuable information relating to battery behaviour and condition. Until now, successful measurements have been made in the laboratory with bench analysers. This requires batteries to be taken out of service for periods ranging from hours to days, which is impractical for units that are needed for continual use or that are difficult to remove for analysis (e.g. electric vehicles), or that are difficult or impossible to access (e.g. satellites, implanted medical devices, distant data centres). We describe a simple microprocessor-based circuit that draws current at a number of frequencies and provides voltage and current data that can be logged and used to calculate the impedance of batteries *in situ*.**

***Index Terms*—Batteries, battery powered vehicles, impedance measurement, frequency domain analysis, embedded systems**

I. Introduction

Recent research has demonstrated the value of very low (of the order of microhertz) frequency measurements on rechargeable batteries [1]–[3]. Researchers making measurements in this region are able to infer characteristics arising from fractional capacitor behaviour that are not apparent at higher frequencies [1], [2], [4].

The circuit model of a battery is key to predicting its behaviour, particularly with reference to its state of charge (SoC) and state of health (SoH) [5], [6]. Most common models are based on an RC network with a voltage source [5], [7]–[10], but there has been increasing interest in and a return to the idea of a fractional model as first hinted at by Randles in 1947 [11]. Recently, Hasan and Scott have presented persuasive empirical evidence that a fractional derivative model based on constant phase elements is required for the modelling of rechargeable batteries, and have demonstrated the utility of this model by measuring impedance in NiMH batteries at frequencies ranging from 10μHz to 10Hz [4]. Impedance magnitude was found to be flat at higher frequencies, but to increase below 10mHz, with a phase shift from around 0 to –80 degrees [2], [4] (Fig. 1).

These insightful measurements have been repeated successfully in the laboratory with impedance analysers capable of measuring down to very low frequencies, and have led to the proposal of a novel algorithm that provides robust and repeatable data [1], [2]. Measurements entail the generation of a sinusoidal current waveform at each frequency of interest, with each measurement carried out over several (ideally six) cycles, as multiple cycles facilitate Fourier post-processing. Data files containing current and voltage data in the time domain are then analysed for magnitude and phase using a discrete Fourier transform (DFT) at the frequency or frequencies of interest. A suitable method is that described by Scott and Parker [12], [13]. Following the presentation of magnitude and phase in the frequency domain, complex impedance may be calculated by taking the quotient of voltage and current at the frequencies of interest.

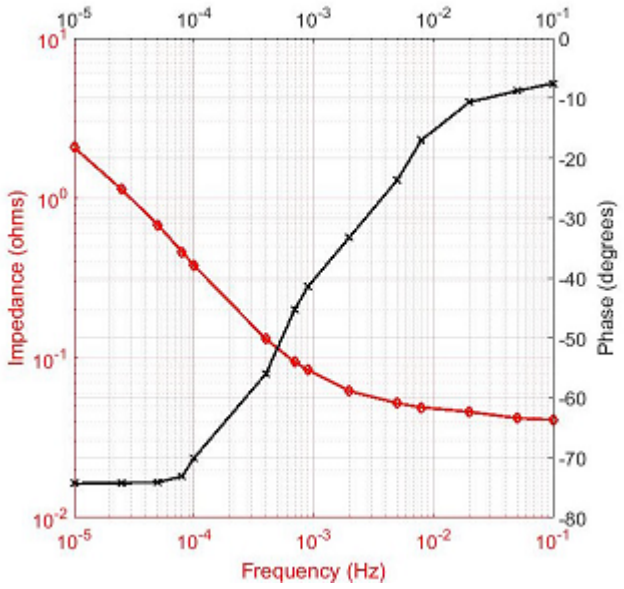


Fig. 1. Impedance and phase as measured using a Solartron 1260A automated impedance analyser: single 55123 1850mAh NiMH cell [2].

As described above, until now these measurements have been taken on the laboratory bench with costly impedance analysers. This begs the question as to what is to be done if the battery to be measured cannot be easily removed from the device it is powering, or is difficult or impossible to access. Examples include batteries powering satellite systems, those in

The authors thank WaikatoLink Ltd (www.waikatolink.co.nz) for assistance with research funding

Fig. 2. ‘eGO’ 24V electric scooter.

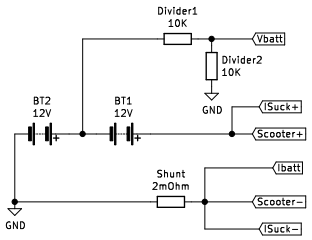


Fig. 3. LabJack and current sink connections. Note the system ground reference point (GND) on the negative battery side; this is connected to the common ground of the LabJack (sgnd). ISuck+ = 24V supply to the current sink; ISuck– = current sink negative terminal. Scooter+ and Scooter– show connections to the scooter’s circuitry and ultimately the motor. Vbatt and Ibatt are LabJack analog inputs.

medical devices implanted in tissue, and those in data centres or similar settings far from the testing laboratory. Moreover, measurements at the frequencies described are lengthy: the period of a 10μHz sine wave is over 27 hours. Six cycles (as required ideally for DFT processing) take over 166 hours (nearly a week) to record. It is neither convenient nor practical to take batteries that are required for use out of service for this long for testing.

There is therefore an as-yet unmet need for means of carrying out informative complex impedance measurements on rechargeable batteries while allowing them to remain in use. We describe the development of a simple and inexpensive circuit, based on a PIC microprocessor, that is capable of drawing current from batteries at multiple low frequencies in order to provide voltage and current data that may be logged for Fourier analysis and subsequent impedance and phase calculations.

II. Measurement System

Measurements were taken from one of a pair of 12V lead-acid (nominal 33Ah) traction batteries wired in series to provide the 24V supply driving an ‘eGO’ electric scooter [14] (Fig. 2).

Battery voltage (Vbatt) was measured from the positive terminal of a single battery (12V nominal). Current drawn from the battery (Ibatt) was based on the voltage measured across a shunt with a known resistance of 0.002Ω placed between the scooter chassis and the system’s ground reference point at the negative side of the supply (Fig. 3).

III. Data Logger (‘LabJack’)

Data were logged using a proprietary ‘LabJack’ U6 device [15]. The LabJack’s analog inputs offer 16 bits of resolution at maximum sampling speed (approximately 20μs conversion time), increasing to 18+ bits at slower speeds.

Data were transferred to an HP Stream laptop computer running Microsoft Windows 10 and loaded with LJStreamUD data acquisition software as supplied for use with the LabJack. LJStreamUD is an executable that streams up to 16 channels; selected channels may be displayed on a graph while data are written to tab-delimited ASCII files. These are then separated into time/voltage and time/current files and transformed for frequency domain analysis using DFT processing as described by Scott and Parker [12], [13].

IV. Current Sink (‘ISuck’)

See Fig. 4 for the PCB layout; Fig. 5 shows the full circuit. The current sink was designed to draw current via the 24V scooter supply, with the negative terminal connected to the scooter chassis. The 24V supply is passed through a low value 10W resistor to provide initial heat dissipation, and then to an LM317 voltage regulator (with further heat sinking) configured to provide a 5V supply to power a PIC16F690 microprocessor [16]. Current is drawn by the circuit through this supply by a pulse width modulation (PWM) signal from pin 5 of the PIC microprocessor. The PWM signal has 10 bits of resolution, and the timestamp integer used to calculate the value of its duty cycle is updated every 200ms by counting in an interrupt that is entered every 64μs. This drives a Sziklai (complementary feedback) pair of bipolar junction transistors (BD139 and BD140). This configuration was chosen as it provides current gain based on the product of the two transistors, but with a single V_{BE} (i.e. half the turn-on voltage of a Darlington pair) [17].

An external oscillator (16MHz) was chosen in preference to the PIC’s on-board oscillator to permit higher clock speeds ($F_{OSC}/4$; the PIC16F690 has an 8MHz internal oscillator) and increased accuracy (depending on supply voltage and temperature, the PIC’s internal oscillator is stated to be subject to an error ranging from $\pm$1% to $\pm$5%) [18].

An additional three-way terminal block has been included on the circuit to allow for attachment of a temperature sensor for future use (LM19 as shown in Fig. 5).

The circuit also includes a number of features that provide information to the user and that facilitate booting into one of two modes, test and measurement. There are two pushbutton switches (SW1 and SW2; Fig. 5): SW1 applies a scaling factor to the multitone sinewave calculations which can be

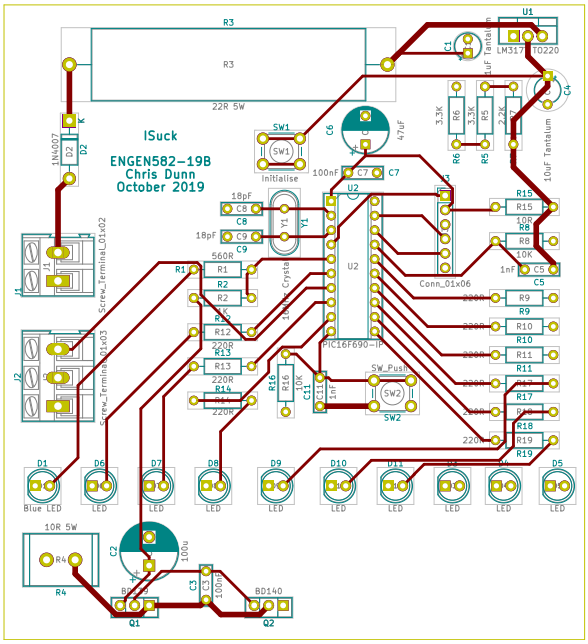


Fig. 4. PCB layout.

used to speed up waveforms for visualisation and testing purposes. If this button is not pressed at power-up, unscaled (i.e. microhertz) values will be used for the frequencies of interest. SW2 reboots the PIC if held down for 5 seconds.

There are 10 diagnostic LEDs:

- a blue LED on pin 5 (the PWM output, see Fig. 5) varies in intensity with the PWM duty cycle;
- five LEDs (D3–D5, D10–D11) show how many frequencies are being tested and when they are in the positive parts of their cycles;
- D7 indicates the entering and leaving of the interrupt service routine (ISR);
- D6 blinks every time the PWM duty cycle is recalculated;
- D9 toggles when the PWM changes;
- D8 is a 'heartbeat' that toggles every time the timestamp is updated in the ISR (see code description below).

V. Current Sink Code

The complete code for the current sink is shown in Appendix 1.

The base frequency from which frequencies of interest will be calculated (f_0) and the number of frequencies (nSines) are declared together with an array (mult[]) to hold the multipliers. These are set as primes to prevent collision of harmonics. Calculation of the amplitude of each sinewave component at each new timepoint takes place in a 'for' loop that repeats 'nSines' times, and the components are added together.

The general equation describing the multitone signal is

$$y_{sum} = \left(\sum_{i=1}^{n} \frac{1}{n} sin(2\pi N_i f_0 t + \phi_i) + 1 \right) \div 2 \qquad (1)$$

where n is the number of sine components, N_i is the ith prime number in the array, f_0 is the base frequency, t is time elapsed, and ϕ is a phase shift calculated as $\frac{2\pi}{3}$, $\frac{2\pi}{5}$, etc.

Note that 1 is added for each sine component to maintain a value greater than 0; division by $2n$ (i.e. $2 \times$ nSines) maintains y_{sum} in the range $0 < y_{sum} < 1$.

Time is kept in the interrupt as an integer number of multiples of 200ms. This number is atomically copied from the ISR every time a new calculation of output signal level is carried out. This is achieved by suspending interrupts and copying a long integer (tsi), an operation that uses much less time than the duration between interrupts (64μs). Once copied into the mainline routine, this number is converted to a floating-point value in seconds. In order to compute the value of each sinewave this number of seconds must be multiplied by the frequency of the sinewave.

When the total time is multiplied by the frequency, the resulting phase soon reaches a large integer multiple of $(2\times\pi)$. A float (the largest precision available on the PIC platform) has a 24-bit mantissa, and thus about 7.5 digits. Once the product of time and frequency exceeds a few tens of thousands, the precision of the phase actually computed by $\sin(2\pi\omega t)$ becomes imprecise, since the $\sin$ function operates on the result modulo 2π, discarding the integer part of ωt.

This problem is dealt with as follows. Observing that t and ω can be written

$$t = t_I + t_f \qquad (2)$$

and

$$\omega = \omega_I + \omega_f \qquad (3)$$

where the subscript $_I$ is the integer part, and $_f$ the fractional or decimal part, of the variable. The product can then be written

$$\omega t = (\omega_I + \omega_f) \times (t_I + t_f) \qquad (4)$$

but as we require only the fractional part of the product, we may immediately discard any integer part of the product:

$$(\omega t)' = \omega_f t_I + \omega_I t_f + \omega_f t_f \qquad (5)$$

and computing

$$\sin(2\pi\omega t) = \sin(2\pi(\omega t)') \qquad (6)$$

without loss of precision. For our calculation, t can be very large, and ω very small. The above algorithm works best when both numbers have substantial integer parts. Thus observing that

$$\omega t = q\omega \times t/q \qquad (7)$$

for any number q, we prefix the above process with a loop that doubles the smaller number and halves the larger until they are comparable in magnitude.

PWM is calculated as

$$pwm = 130.0 + ((1024.0 - 130.0) \times y_{sum}) \qquad (8)$$

which ensures that the value of pwm remains at or above 130. As $\frac{130.0}{1024.0} \times 5V = 0.63V$, this ensures that V_{BE} across Q1 remains 'on', thus facilitating smooth current draw through

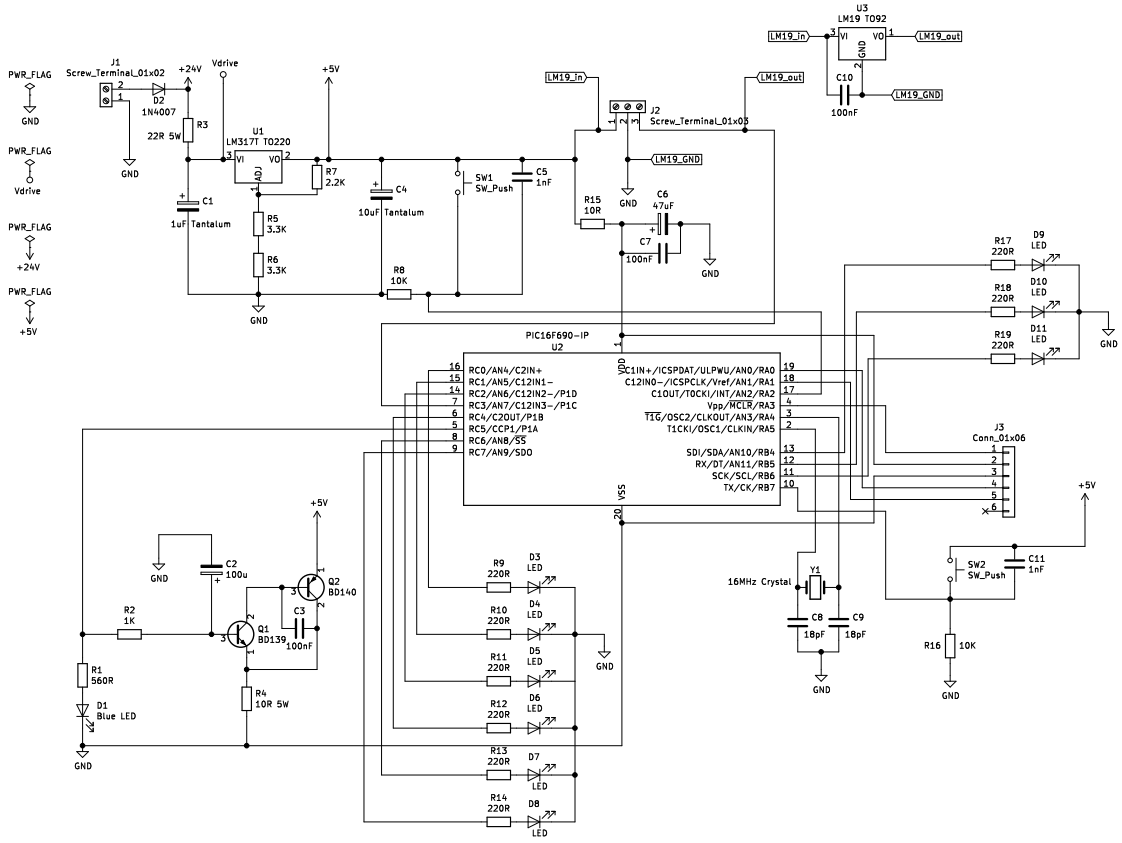


Fig. 5. Current-sinking 'ISuck' circuit.

the Sziklai pair. The raising of the PWM 'floor' involves some unavoidable loss of PWM resolution, but this is of no practical consequence.

VI. Measurement and Verification of Test Signal

A. Setup and Programming

The circuit was programmed with the code shown in Appendix 1. f_0 was set at 30μHz, with multipliers 3, 7, 13, 29 and 43 (corresponding to five frequencies: 90, 210, 390, 870 and 1290μHz). Upon booting, SW1 was held down so that a scaling factor of 100 would be applied to these frequencies, thus providing a test range of 9–129mHz. As six periods of the lowest of these frequencies are equal to just over 11 minutes, the system was left to run over a lunchtime to provide a generous data file.

Inputs to the LabJack were ranged at $\pm$10V for the Vbatt input and at $\pm$1V and $\pm$0.1V for Ibatt (i.e. the voltage at the head of the shunt relative to ground; Fig. 3). Two analog inputs were used for Ibatt to explore the ranging capabilities and behaviour of the LabJack. For interest, a thermal image of the circuit while running is shown in Fig. 6. Oscilloscope outputs showing the use of the diagnostic LEDs to verify operational timing around the circuit are shown in Appendix 2.

B. Time and Frequency Domain Results

Fig. 7 shows voltage as recorded on the Q1 emitter of the Sziklai pair. For comparison, Ibatt as recorded by the LabJack and plotted using GnuPlot version 5.2 is shown in Fig. 8. Inspection of voltages in the time domain in the presence of the 10Ω resistor between the emitter of Q1 and ground suggests that the circuit is drawing up to approximately 400mA.

Frequency spectra after applying the DFT for Ibatt and Vbatt are shown in Fig. 9 and Fig. 10. Note that these data were obtained with the scooter at rest (i.e. not charging and with no load being applied by the motor).

Fourier peaks appear at positions corresponding to the frequencies of interest, at least one order of magnitude above the scooter's resting noise floor. Moreover, Ibatt peaks appear at the same level, whereas Vbatt peaks show a decrease with increasing frequency, as expected. In a constant phase element, impedance (Z_ω) falls as frequency increases (as in a capacitor) [1], [19], while current draw by 'ISuck' is equal for all tones. Thus, V = $Z_\omega \times$ I falls with increasing frequency.

Fig. 6. Thermal image of the 'ISuck' circuit while in test mode. Note the maximum temperature at the input resistor and voltage regulator of approximately 62°C.

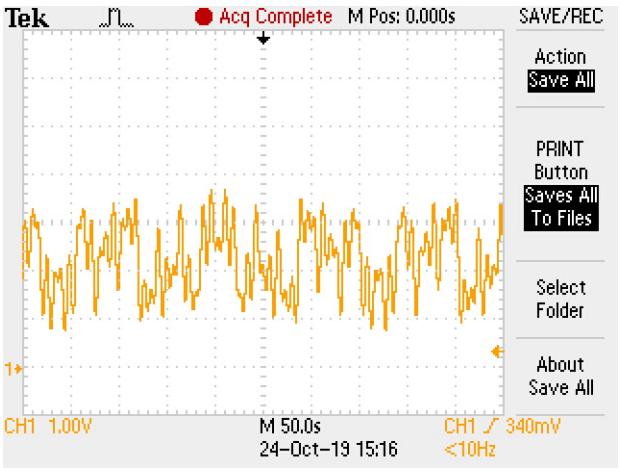


Fig. 7. Q1 Emitter voltage as shown by a Tektronix TDS2014 digital oscilloscope.

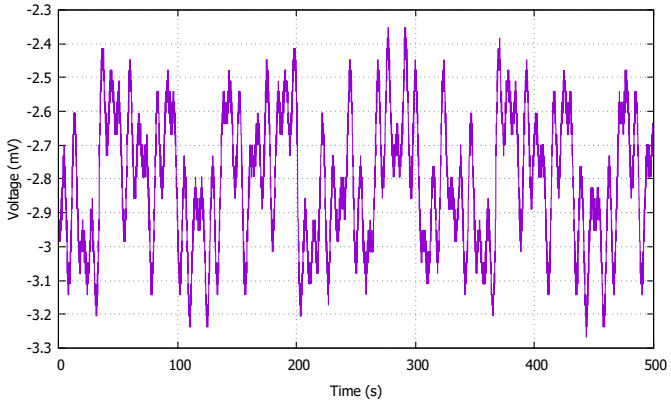


Fig. 8. Shunt voltage ('Ibatt') as recorded by LabJack U6 (±1V range input). Note that LJStreamUD can be set up to apply equations (for calculation of current across the shunt, for example) to raw data such as these for output on another channel.

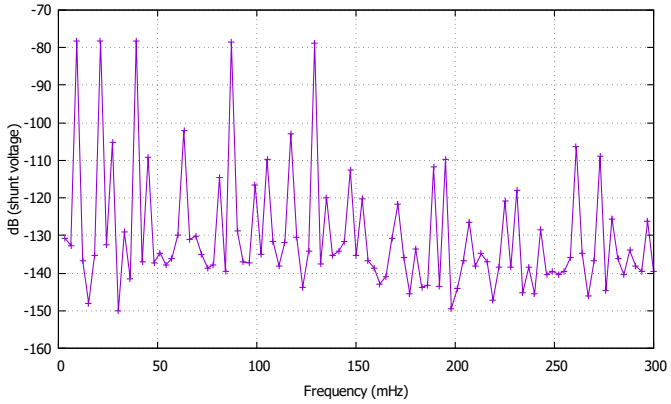


Fig. 9. Fourier spectrum of shunt voltage (Ibatt). Fundamental = 3mHz; 100 harmonics; Hann window.

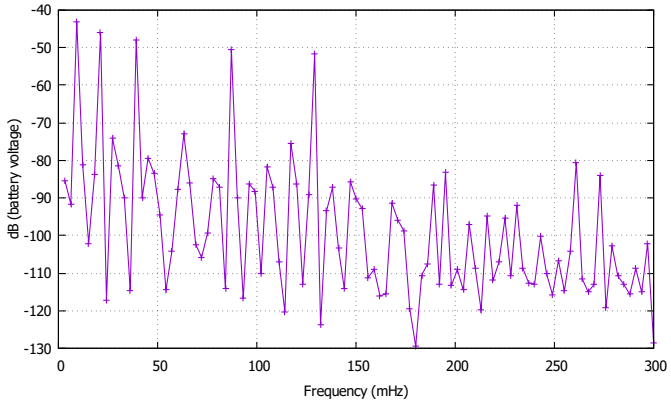


Fig. 10. Fourier spectrum of battery voltage (Vbatt). Fundamental = 3mHz; 100 harmonics; Hann window.

VII. Conclusions

The above results demonstrate that the current-sinking device described can be used to draw current at multiple frequencies from an in-use battery, and thereby provide data in the time domain that can be successfully transformed into the frequency domain. The system has sufficient resolution to generate spectra with clear peaks at the frequencies of interest, more than an order of magnitude above the noise floor, from which it is likely to be possible to derive impedance and phase measurements. This would be carried out with the circuit booted into measurement rather than test mode, as the frequencies used here are likely to be too high to generate impedance data of interest (i.e. in the region where the partial derivative characteristic becomes clearly apparent). We note also that these measurements were taken with the scooter at rest, and have yet to ascertain the effects on measurements of charging and riding. Nevertheless, these results show the feasibility and potential applicability of a measurement system based on the 'ISuck' circuit that would be capable of providing reliable complex impedance data for use in deducing battery SoH.

References

[1] J. Scott and R. Hasan, "New results for battery impedance at very low frequencies", IEEE Access, vol. 7, pp. 106925–106930, 2019.

[2] J. Scott and R. Hasan, "Measurement of battery impedance extending to extremely low frequencies", University of Waikato, Hamilton, New Zealand, 2018.

[3] R. Hasan and J. Scott, "Impedance measurement of batteries under load", Proceedings of the 2019 IEEE International Instrumentation and Measurement Technology Conference, Auckland, New Zealand.

[4] R. Hasan and J. Scott, "Fractional behaviour of rechargeable batteries", Proceedings of the 2016 Electronics New Zealand Conference (ENZCon), November 17–18, 2016, Victoria University of Wellington, pp111–114. https://ecs.victoria.ac.nz/foswiki/pub/Events/ENZCon2016/WebHome/ENZCon 2016 Proceedings.pdf

[5] G. Nobile, M. Cacciato, G. Scarcella, and G. Scelba, "Performance Assessment of Equivalent-Circuit Models for Electrochemical Energy Storage Systems", 43rd Annual Conference of the IEEE Industrial Electronics Society (IECON), 2017, pp. 2799–2806.

[6] C. Zhang, K. Li, S. Mcloone, and Z. Yang, "Battery modelling methods for electric vehicles—A review", Proc. Eur. Control Conf. (ECC), pp. 2673-2678, Jun. 2014.

[7] J. Chiasson and B. Vairamohan, "Estimating the state of charge of a battery", IEEE Transactions on Control Systems Technology, vol. 13, no. 3, May 2005, pp. 465–470.

[8] T. Wu, L. Liu, Q. Xiao, Q. Cao, and X. Wang, "Research on SoC Estimation Based on Second-order RC Model", Telkomnika, vol. 10, no. 7, Nov 2012, pp. 1667–1672.

[9] Z. Cheng, J. Lv, Y. Liu, and Z. Yan, "Estimation of State of Charge for Lithium-Ion Battery Based on Finite Difference Extended Kalman Filter", Journal of Applied Mathematics, vol. 2014, Article ID 348537, 10 pages, 2014.

[10] M. Chen and G. A. Rincón-Mora, "Accurate Electrical Battery Model Capable of Predicting Runtime and I/V Performance", IEEE Transactions on Energy Conversion, vol. 21, no. 2, June 2006.

[11] J. E. B. Randles, "Kinetics of rapid electrode reactions", Discussions of the Faraday Society, vol. 1, pp. 11–19, 1947.

[12] J. Scott and A. Parker, "Modern guide to spectral analysis with SPICE", IEEE Circuits and Devices Magazine, vol. 11, no. 5, pp. 10–16, September 1995.

[13] J. Scott and A. Parker, "Distortion analysis using SPICE", J. Audio Eng. Soc., vol. 43, no. 12, pp. 10–16, December 1995.

[14] EGO Vehicles LLC, Cambridge, MA, USA; http://www.egovehicles.com/.

[15] LabJack Corporation, Lakewood, CO, USA; https://labjack.com/products/u6.

[16] Microchip Technology Inc., Chandler, AZ, USA; https://www.microchip.com/.

[17] R. Elliott, "Compound pair vs. Darlington pairs", Elliott Sound Products, 6 January 2011, available from: https://sound-au.com/articles/cmpd-vs-darl.htm.

[18] "Microchip PIC16F631/677/685/687/689/690: 20-pin flash-based, 8-bit CMOS microcontrollers", ©2005–2015 Microchip Technology Inc., DS40001262F.

[19] M. E. Orazem, I. Frateur, B. Tribollet, V. Vivier, S. Marcelin, et al., "Dielectric Properties of Materials Showing Constant-Phase-Element (CPE) Impedance Response", Journal of The Electrochemical Society, vol. 160, no. 6, pp. C215–C225, 2013.

APPENDIX 1. 'ISUCK' CODE.

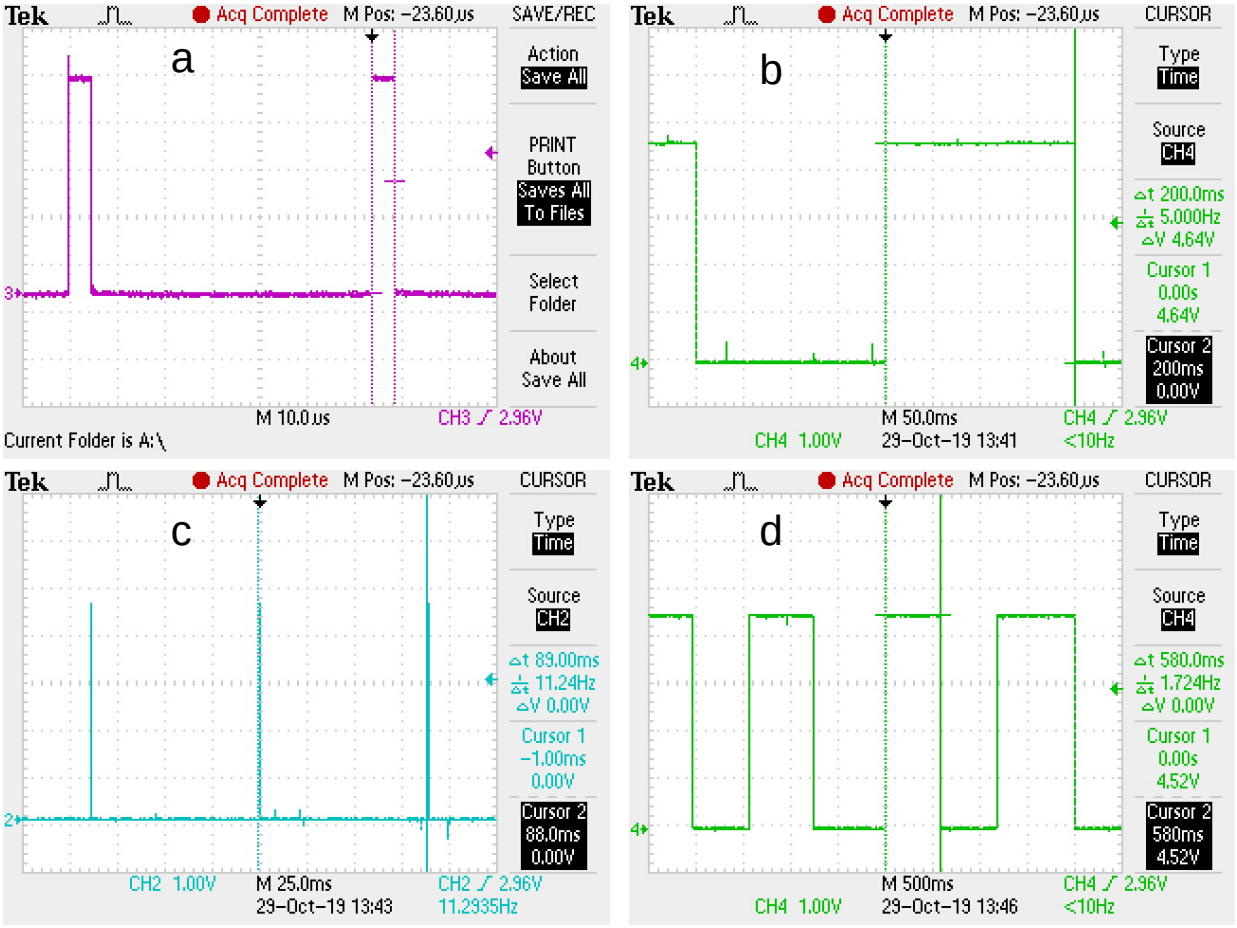
```
/*
 * File:   ISuck_III.c
 * Author: Chris Dunn, Vance Farrow & Jonathan Scott
 *
 * Created on 20 October 2019, 4:39 PM
 * Edited by JBS 20 Oct 2019
 */


#include <xc.h>
#include <math.h>
#include <stdint.h>

// CONFIG
#pragma config FOSC = HS        // Oscillator Selection bits (HS oscillator: Hi-speed crystal/resonator on RA4/OSC2/CLKOUT and RA5/OSC1/CLKINT)
#pragma config WDTE = OFF       // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = OFF      // Power-up Timer Enable bit (PWRT disabled)
#pragma config MCLRE = OFF      // MCLR Pin Function Select bit (MCLR pin function is digital input, MCLR internally tied to VDD)
#pragma config CP = OFF         // Code Protection bit (Program memory code protection is disabled)
#pragma config CPD = OFF        // Data Code Protection bit (Data memory code protection is disabled)
#pragma config BOREN = ON       // Brown Out Detect (BOR enabled)
#pragma config IESO = ON        // Internal External Switchover bit (Internal External Switchover mode is enabled)
#pragma config FCMEN = ON       // Fail-Safe Clock Monitor Enabled bit (Fail-Safe Clock Monitor is enabled)


//Some intuitive definitions to clarify the code
#define SET 1
#define CLEAR 0
#define TRUE 1
#define FALSE 0
#define ON 1
#define OFF 0

//Hardware definitions
#define PUSHBUTTON RA2
#define SW2 RB7

#define D3 RC0
#define D4 RC1
#define D5 RC2
#define D6 RC4
#define D7 RC6
#define D8 RC7
#define D9 RB4
#define D10 RB5
#define D11 RB6

//numerical definitions
#define pi2 6.28318530718
#define f0 30e-6

volatile uint16_t timer1 @0x0E; //Variable to handle Timer 1
//Note: use 'volatile' to tell compiler that its value may change at any time
//Note also the use of 'construct @ address' to declare 'timer1' located at 0Eh
uint32_t timeElapsedFifthSec = 0; //Counter for 1/5 second
uint16_t ticks = 0; //Counter to space timeElapsedFifthSec
float scale = 1.0; // frequency adjust variable
volatile uint8_t PBcnt = 0; //Counter for reboot pushbutton (SW2))
volatile uint8_t run = 1; // allows reboot
bit plus; // am I >0

void init(void) {
    //Initialize 16F690 Microcontroller
    run = 1;
    VRCON = CLEAR; //Turn Off Voltage Reference Peripheral
    CM1CON0 = CM2CON0 = 0x00; //Turn Off Comparator Peripheral
    PORTA = CLEAR; //Clear the I/O registers (two lots)
```

```
68      PORTB = CLEAR;
69      PORTC = CLEAR; //We read and write to these; TRISA and C set I/P and O/P
70      TRISA = 0xFF; //Set inputs and outputs: RA0-RA4=1 (input); RA5=1 (input for clock)
71      TRISB = 0x80;
72      WPUB = 0x00;
73      TRISC = 0x00; //Set all as outputs
74      OSCCON = 0x00; //16MHz, external oscillator
75
76      //ANSEL, ADCON0 and ADCON1 control the functionality of the A/D module
77      ANSEL = 0x00; //
78      ANSELH = 0x00; //
79
80      ADCON0 = 0b10001101; //Rt-justified; VddRef; channel select; Go_nDone; enable
81      ADCON1 = 0b0010000; //Set conversion clock: 001 for Fosc 4MHz, 101 for 8MHz
82
83      /*Timers - set up individual bits in OPTION_REG*/
84
85      //Timer 0 setup in OPTION_REG
86      OPTION_REG = 0x08; //Prescaler assigned to WDT; count instructions
87      TMR0 = CLEAR; //Clear TIMER0 register (8 bits: counts from 0-255)
88
89      /*Timer1 control register*/
90      T1CON = 0x01; //enable Timer1; int clock, 1:1 prescale
91      TMR1 = CLEAR; //Clear Timer1 register (16 bits: counts from 0-65535)
92
93      /*Set bits in INTCON register*/
94      T0IE = ON; //Timer0 Overflow Interrupt enabled
95      T0IF = CLEAR; //Clear Timer0 Overflow Interrupt Flag
96      //    RAIF = CLEAR; //Clear PORTA Change Interrupt Flag bit
97      //    INTF = CLEAR; //Clear RA2/INT External Interrupt Flag bit
98
99      GIE = CLEAR; //Enable All Interrupts
100
101     /*PWM*/
102     CCP1CON = 0b00001100; //PWM single o/p; P1A-P1D active high; DC1B bits await
        values
103     PR2 = 0xff; //Load PR2 register (255 gives 1024 duty cycles - max resolution)
104     /*See p83 of datasheet for full details*/
105     CCPR1L = CLEAR; //Clear ready for the 8 MSb of the duty cycle
106     /*CCPR1H read-only in PWM mode. The 2 LSb are put into CCP1CON <5:4> (DC1B bits)*/
107     /*Configure and start timer 2*/
108     TMR2 = CLEAR; //Clear timer 2 register
109     TMR2IE = SET; //Enable the Timer 2 to PR2 match interrupt in PIE1 register
110     PIR1 = CLEAR; //Clear TMR2IF and TMR1IF interrupt flags
111     T2CON = 0b00000100; //Postscaler not used; enable timer 2; prescale 1:1
112
113     //Flash an LED when turning on
114     for (int i = 0; i < 30000; i++) { //Every 0.12 seconds
115         if (i % 10000 < 5000) {
116             D5 = D4 = D3 = ON;
117         } else {
118             D5 = D4 = D3 = OFF;
119         }
120     }
121     /*Button push at startup - determines whether to use test frequencies or
122      1000 times these to speed up for testing/observation purposes*/
123     if (PUSHBUTTON) {
124         scale = 100.0; // all frequencies will be 'scale' times faster
125         for (int i = 0; i < 30000; i++) { // more flashes to show PB picked up
126             if (i % 10000 < 5000) { // shows longer booty-flash
127                 D5 = D4 = D3 = ON;
128             } else {
129                 D5 = D4 = D3 = OFF;
130             }
131         }
132     } else {
133         scale = 1.0;
134     }
135 }
136
137 void main(void) {
138
139     init();
```

```
140
141         float ysum; //Variable to hold the sum of successive sine calculations
142         float y; //Variable to hold the results of successive sine calculations
143         uint8_t nSines = 5; //Number of individual sine frequencies to be included
144         uint8_t i; //Counter for sine frequencies
145         float f, timeStamp; //floats containing time and frequency in Hertz
146         float fq, tz; //floats to be multiplied without loss of lower digits
147         unsigned char mult[5] = {3, 7, 13, 29, 43}; // prime multiples prevents
            harmonics colliding
148         //    unsigned char pha[5] = {pi2/7, pi2/2, pi2/5, pi2/11, 0}; // prime
            multiples prevents harmonics colliding
149
150         uint16_t pwm, lastpwm; //Variable to control PWM duty cycle
151         uint32_t fqi, tzi, tsi; //temp store of long int (as cheap compiler does not
            reuse memory)
152
153
154
155         while (run) {
156
157             /*Pause the interrupt, take the number of fifths of seconds counted
158              and divide by 5 to convert the counter value back into actual seconds.
159              Assign this to 'timeStamp'*/
160             D6 = 1; // floatLED: Blink every time we recalculate
161             GIE = 0; //Disable all interrupts
162             tsi = timeElapsedFifthSec; // atomic grab of time from ISR
163             GIE = 1; // ISR back on QUICK!
164             timeStamp = tsi / 5.0; // timestamp in secs
165             D6 = 0;
166
167             for (ysum = 0, i = 0; i < nSines; i++) { //For each sine frequency
168                 f = mult[i] * f0; // frequency of interest
169                 f *= scale; // optionally scaled for testing
170                 // we seek the fractional part of f*timeStamp
171                 // Let f*timeStamp = (f*v)*(timeStamp/v) for some v
172                 // put fq = f*v and tz = timeStamp/v
173                 // fq=i+q where i is an integer and 0<=q<1, likewise tz=j+z;
174                 // The product fq*tz = ij + iz + qj + qz;
175                 // The fractional(decimal) part will not include ij, so it can be thrown
                    away.
176                 fq = f;
177                 tz = timeStamp;
178                 while (tz > fq) { // time bigger than frequency
179                     tz /= 2; // time 2x smaller
180                     fq *= 2; // freq 2x bigger
181                 } // now these two numbers are comparable
182                 // now carry out multiplication but discarding integer product term
183                 fqi = (int) fq; // break f into integer and decimal parts
184                 fq = fq - fqi; // subtract integral part so fq is now the decimal part...
185                 tzi = (int) tz; // break f into integer and decimal parts
186                 tz = tz - tzi; // subtract integral part so tz is now the decimal part...
187                 f = fq*tz; // two fractional parts
188                 f += fq*tzi; // frac freq * int time part
189                 f += tz*fqi; // frac time * int freq part
190                 y = sin(pi2 * f); /*Calculate amplitude*/
191
192                 /*Turn LEDs on when sine components are in +ve half of cycle*/
193                 if (y > 0.0) {
194                     plus = 1;
195                 } else {
196                     plus = 0;
197                 }
198                 if (i == 0) {
199                     if (plus == 1) {
200                         D5 = 1;
201                     } else {
202                         D5 = 0;
203                     }
204                 }
205                 if (i == 1) {
206                     if (plus == 1) {
207                         D4 = 1;
208                     } else {
```

```
                    D4 = 0;
                }
            }
            if (i == 2) {
                if (plus == 1) {
                    D3 = 1;
                } else {
                    D3 = 0;
                }
            }
            if (i == 3) {
                if (plus == 1) {
                    D11 = 1;
                } else {
                    D11 = 0;
                }
            }
            if (i == 4) {
                if (plus == 1) {
                    D10 = 1;
                } else {
                    D10 = 0;
                }
            }
            ysum += y; //Add amplitudes together
        } // end for i < nSines
        ysum += nSines; //Add number of frequencies so always >0
        ysum /= 2 * nSines; //Divide by 2 & each frequency to keep x in range 0<x<1

        pwm = (130.0 + ((1024.0 - 130.0) * ysum));
        if (pwm != lastpwm) { // value changed, new pwm level this time around
            D9 ^= 1; // thinkLED toggles
            //Set PWM register contents: 10 bits available
            CCPR1L = (pwm >> 2);
            if (pwm & 0x02)
                DC1B1 = 1;
            else DC1B1 = 0;
            if (pwm & 0x01)
                DC1B0 = 1;
            else DC1B0 = 0;
        }
        lastpwm = pwm;
    }
}

/*Fclock = Fosc/4 = 4mips*/

/*TMR0 (8-bit timer, 0-255) overflows every 64us (256*(1/4e6))*/
void interrupt ISR() {
    D7 = 1; // ISR LED, check duration & frequency on scope
    if (T0IF) { //If TMR0 is the source of the interrupt

        if (++ticks > 3125) { //3125*64us = 0.2s
            D8 ^= 1; // heartbeat LED, 0.4s period, 2.5Hz
            if (SW2) { // if second button held down
                PBcnt++;
                if (PBcnt > 25) { // for 5 seconds...
                    run = 0; // reboot by exiting main while
                }
            } else {
                PBcnt = 0;
            }
            timeElapsedFifthSec += 1; //Increment every 1/5 second
            ticks = 0;
        }
        T0IF = CLEAR; // clear ISR flag
    }
    D7 = 0; // clear ISR LED
    return;

}
```

APPENDIX 2. DIAGNOSTIC LED OSCILLOSCOPE TRACES (TEKTRONIX TDS2014).



a. D7 'ISR LED': turns on when entering the interrupt service routine (ISR) and off again when leaving. Staying on as shown here for approximately 5μs indicates execution of around 20 instructions, which is consistent with timeElapsedFifthSec not being incremented on this occasion (i.e. ++ticks $\not>$ 3125).
b. D8 'heartbeat LED': shows incrementing of timeElapsedFifthSec every 0.2s (3125 $\times$ 60μs) in the interrupt.
c. D6 'floatLED': comes on for 20μs approximately every 90ms, i.e. every time 'timeStamp' is recalculated at the top of while(run).
d. D9 'thinkLED': toggles every time the PWM value changes (i.e. when pwm != lastpwm). The rate of toggling varies according to the slope of the signal, and reassures the user that the processor is computing sufficiently rapidly. This would not be the case if D9 were to catch up with D8 'heartbeat'.