Application Note 112

March 2007

Developments in Battery Stack Voltage Measurement

A Simple Solution to a Not So Simple Problem

Jim Williams and Mark Thoren

Automobiles, aircraft, marine vehicles, uninterruptible power supplies and telecom hardware represent areas utilizing series connected battery stacks. These stacks of individual cells may contain many units, reaching potentials of hundreds of volts. In such systems it is often desirable to accurately determine each individual cell's voltage. Obtaining this information in the presence of the high "common mode" voltage generated by the battery stack is more difficult than might be supposed.

The Battery Stack Problem

The "battery stack problem" has been around for a long time. Its deceptively simple appearance masks a stubbornly resistant problem. Various approaches have been tried, with varying degrees of success.[1]

Figure 1's voltmeter measures a single cell battery. Beyond the obvious, the arrangement works because there are no voltages in the measurement path other than the measurand. The ground referred voltmeter only encounters the voltage to be measured.

Figure 2's "stack" of series connected cells is more complex and presents problems. The voltmeter must be switched between the cells to determine each individual cell's voltage. Additionally, the voltmeter, normally composed of relatively low voltage breakdown components, must withstand input voltage relative to its ground terminal. This "common mode" voltage may reach hundreds of volts in a large series connected battery stack such as is used in an automobile. Such high voltage operation is beyond the voltage breakdown capabilities of most practical semiconductor components, particularly if accurate measurement is required. The switches present similar problems. Attempts at implementing semiconductor based switches encounter difficulty due to voltage breakdown

LT, LTC and LTM are registered trademarks of Linear Technology Corporation. All other trademarks are the property of their respective owners.

[1]See Appendix A, "A Lot of Cut Off Ears and No Van Goghs" for detail and commentary on some typical approaches.

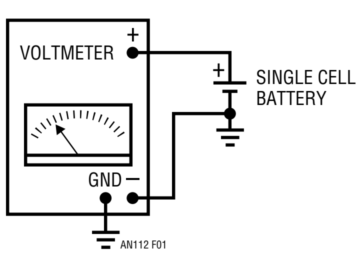


Figure 1. Voltmeter Measuring Ground Referred Single Cell is Not Subjected to Common Mode Voltage

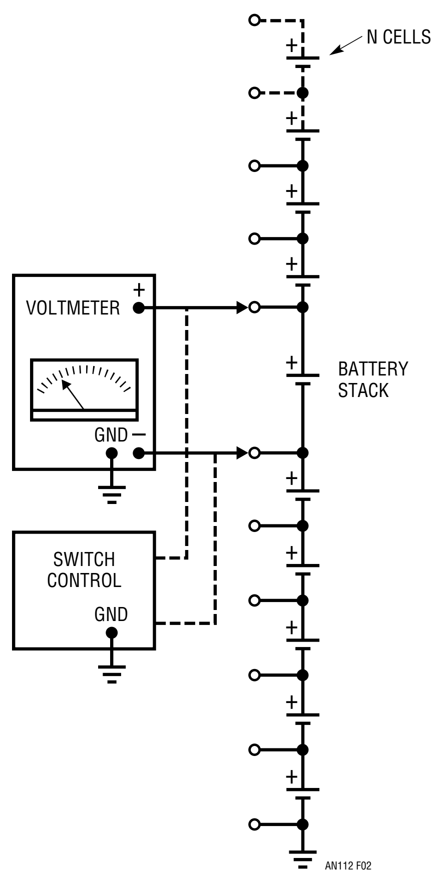


Figure 2. Voltmeter Measuring Cell in Stack Undergoes Increasing Common Mode Voltage as Measurement Proceeds Up Stack. Switches and Switch Control Also Encounter High Voltages

and leakage limitations. What is really needed is a practical method that accurately extracts individual cell voltages while rejecting common mode voltages. This method cannot draw any battery current and should be simple and economically implemented.

Transformer Based Sampling Voltmeter

Figure 3's concept addresses these issues. Battery voltage ($V_{BATTERY}$) is determined by pulse exciting a transformer (T1) and recording transformer primary clamp voltage after settling occurs. This clamp voltage is predominately set by the diode and $V_{BATTERY}$ shunting and similarly clamping T1's secondary. The diode and a small transformer term constitute predictable errors and are subtracted out, leaving $V_{BATTERY}$ as the output.

Detailed Circuit Operation

Figure 4 is a detailed version of the transformer based sampling voltmeter. It closely follows Figure 3 with some minor differences which are described at this section's conclusion. The pulse generator produces a 10µs wide event (Trace A, Figure 5) at a 1kHz repetition rate. The pulse generator's low impedance output drives T1 via a 10k resistor and also triggers the delayed pulse generator. T1's primary (Trace B) responds by rising to a value representing the sum of $V_{DIODE} + V_{BATTERY}$ along with a small fixed error contributed by the transformer. T1's primary clamps at this value. After a time (Trace C) dictated by the delayed pulse generator a pulse (Trace D) closes S1, allowing C1 to charge towards T1's clamped value. After a number of pulses C1 assumes a DC level identical to T1's primary clamp voltage. A1 buffers this potential and feeds differential amplifier A2. A2, operating at a gain near unity, subtracts the diode and transformer error terms, resulting in a direct reading $V_{BATTERY}$ output.

Accuracy is critically dependent on transformer clamping fidelity over temperature and clamp voltage range. The carefully designed transformer specified yields Figure 6's waveforms. Primary (Trace A) and secondary (Trace B) clamping detail appear at highly expanded vertical scale. Clamping flatness is within millivolts; trace center aberrations derive from S1 gate feedthrough. Tight transformer clamp coupling promotes good performance. Circuit accuracy at 25°C is 0.05% over a 0V to 2V battery range with 120ppm/°C drift, degrading to 0.25% at $V_{BATTERY}$ = 3V.[2]

[2]Battery stack voltage monitor development is aided by the floating, variable potential battery simulator described in Appendix B.

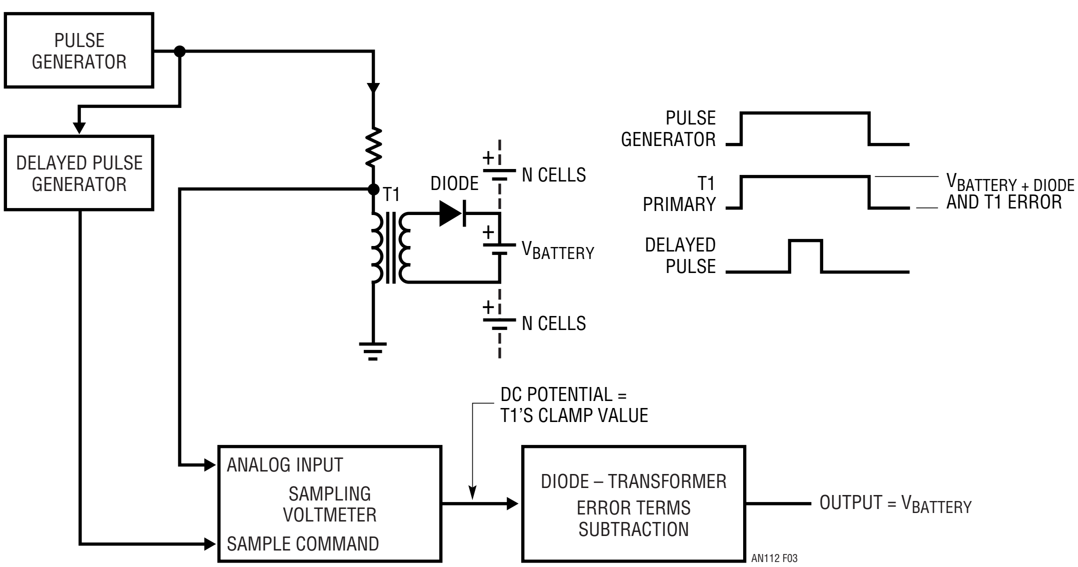


Figure 3. Transformer-Based Sampling Voltmeter Operates Independently of High Common Mode Voltages. Pulse Generator Periodically Activates T1. Delayed Pulse Triggers Sampling Voltmeter, Capturing T1's Clamped Value. Residual Error Terms are Corrected in Following Stage

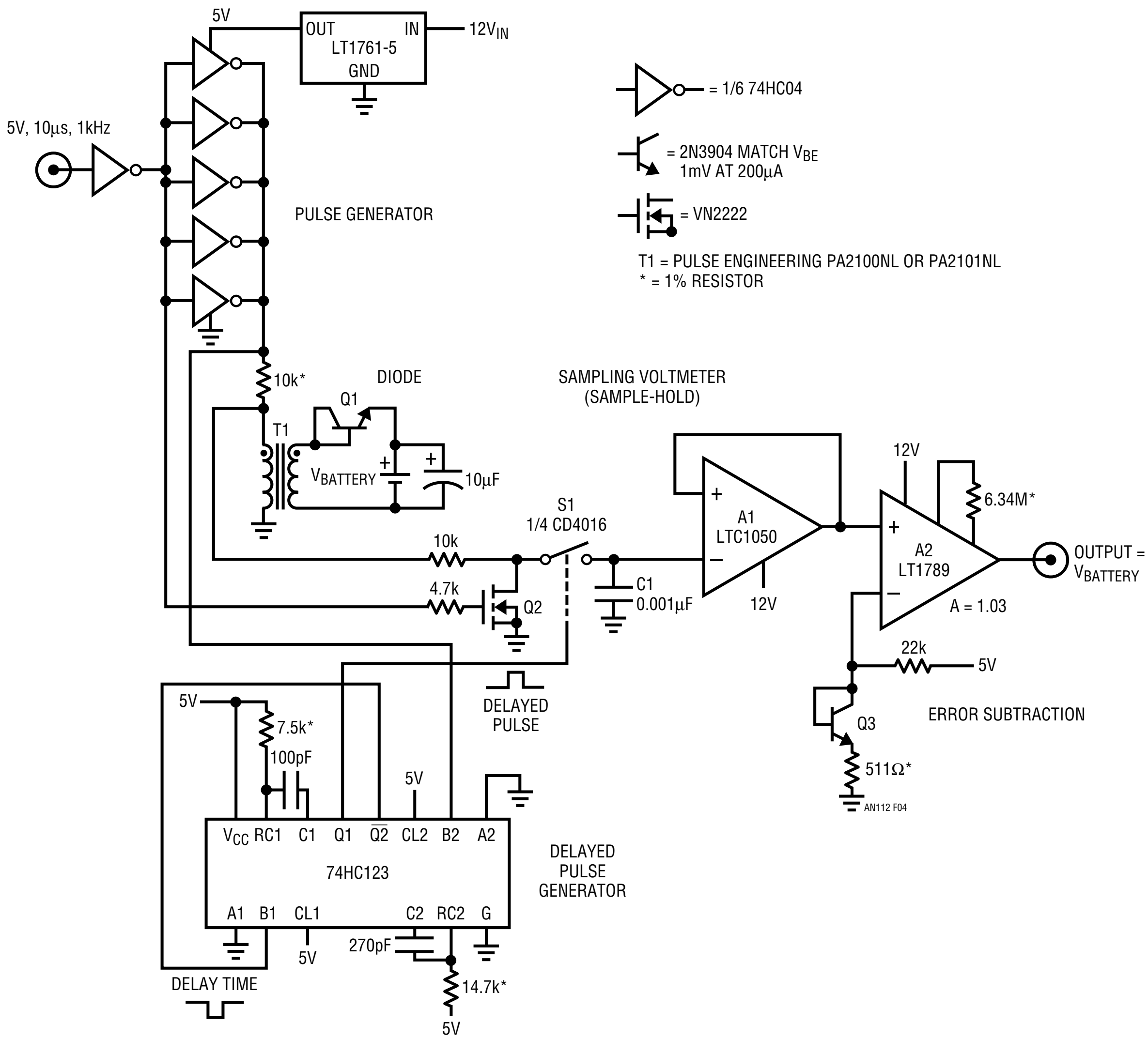
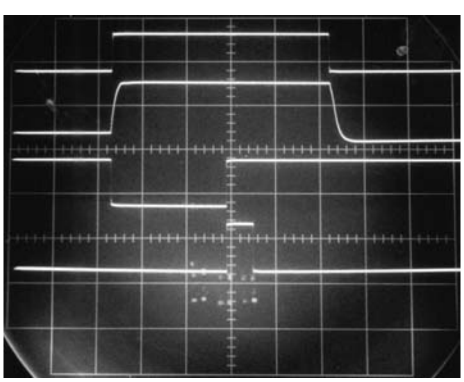


Figure 4. Transformer Fed Sampling Voltmeter Schematic Closely Follows Figure 3's Concept. Error Subtraction Terms Include Q3 Compensating Q1 and Resistor/Gain Corrections for Errors in T1's Clamping Action. Q1-Q3 Transistors Replace Diodes for More Consistent Matching. Q2 Prevents T1's Negative Recovery Excursion from Influencing S1

A = 5V/DIV
B = 2V/DIV
C = 5V/DIV
D = 5V/DIV



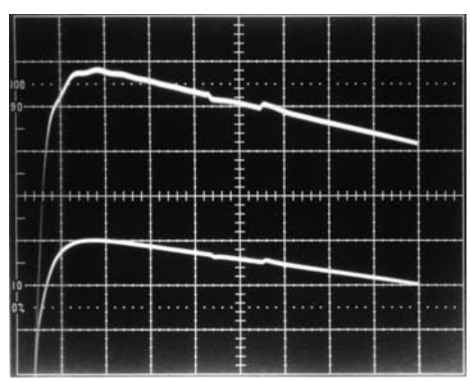
2µs/DIV

AN112 F05

Figure 5. Figure 4's Waveforms Include Pulse Generator Input (Trace A), T1 Primary (Trace B), 74HC123 $\overline{Q2}$ Delay Time Output (Trace C) and S1 Control Input (Trace D). Timing Ensures Sampling Occurs When T1 is Settled in Clamped State

A = 2mV/DIV
ON 2.2V STEP

B = 2mV/DIV
ON 2.2V STEP



2µs/DIV

AN112 F06

Figure 6. T1 Primary (Trace A) and Secondary (Trace B) Clamping Detail. Highly Expanded Vertical Scale Shows Primary and Secondary Clamping Flatness Within Millivolts. Trace Center Aberrations Derive from S1 Gate Feedthrough

Several details aid circuit operation. Transistor V_{BE}'s, substituted for diodes, provide more consistent initial matching and temperature tracking. The 10µF capacitor at Q1 maintains low impedance at frequency, minimizing cell voltage movement during the sampling interval. Finally, synchronously switched Q2 prevents T1's negative recovery excursion from deleteriously influencing S1's operation.

This approach's advantage is that its circuitry does not encounter high common mode voltages—T1 galvanically isolates the circuit from common mode potentials associated with $V_{BATTERY}$. Thus, conventional low voltage techniques and semiconductors may be employed.

Multi-Cell Version

The transformer-based method is inherently adaptable to the multi-cell battery stack measurement problem previously described. Figure 7's conceptual schematic shows a multi-cell monitoring version. Each channel monitors one cell. Any individual channel may be read by biasing its appropriate enable line to turn on a FET switch, enabling that particular channel's transformer. The hardware required for each channel is typically limited to a transformer, a diode connected transistor and a FET switch.

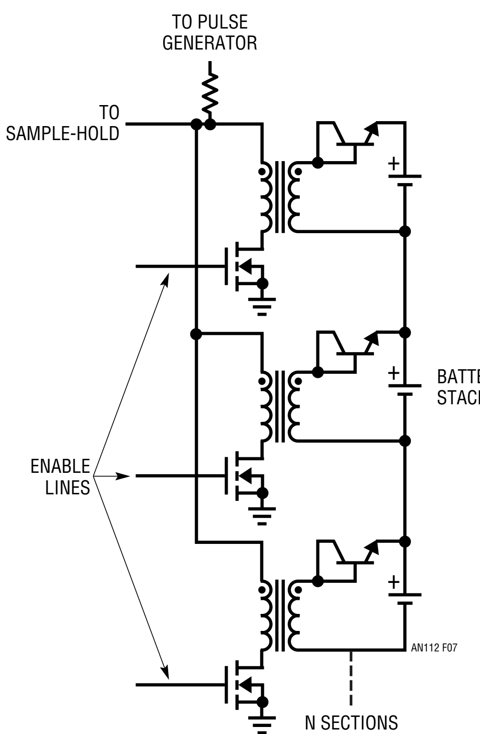


Figure 7. Multiple Channels are Facilitated by Adding Enable Lines and Transistor Switches

Automatic Control and Calibration

This scheme is suited to digitally based techniques for automatic calibration. Figure 8 uses a PIC16F876A microcontroller, fed from an LTC1867 analog to digital converter, to control the pulse generators and channel selection. As before, even though the cell stack may reach hundreds of volts, the transformer galvanic isolation allows the signal path components to operate at low voltage.

A further benefit of processor driven operation is elimination of Figure 4's V_{BE} diode matching requirement. In practice, a processor-based board is tested at room temperature with known voltages applied to all input terminals. The channels are then read, furnishing the information necessary for the processor to determine each channel's initial V_{BE} and gain. These parameters are then stored in nonvolatile memory, permitting a one-time calibration that eliminates both V_{BE} mismatch and gain mismatch induced errors.

Channels 6 and 7 provide zero and 1.25V reference voltages representing cell voltage extremes. The room-temperature values are stored to nonvolatile memory. As temperature changes occur, readings from channel 6 and 7 are used to calculate a change in offset and a change in gain that are applied to the six measurement channels. The calibration is maintained as temperature varies because each channel's –2mV/°C V_{BE} drift slopes are nearly identical. Similarly, gain errors from channel to channel are nearly identical.

Since the gain and offset are continuously calibrated, the gain and offset of the LTC1867 drop out of the equation. The only points that must be accurate are the 0V measurement (easy, just short the channel 6 inputs together) and the 1.25V reference voltage, provided by an LT®1790-1.25. The LTC®1867 internally amplifies its internal 2.5V reference to 4.096V at the REFCOMP pin, which sets the full scale of the ADC (4.096V when it is configured for unipolar mode, ±2.048V in bipolar mode). Thus the absolute maximum cell voltage that can be measured is 3.396V. And since the offset measurement is nominally 0.7V at the ADC input it is never in danger of clamping at zero. (A zero reading will result if a given LTC1867 has a negative offset and the input voltage is any positive voltage less than or equal to the offset.)

Accuracy of the processor-driven circuit is 1mV over a 0V to 2V input range at 25°C. Drift drops to less than 50ppm/°C—almost 3x lower than Figure 4.

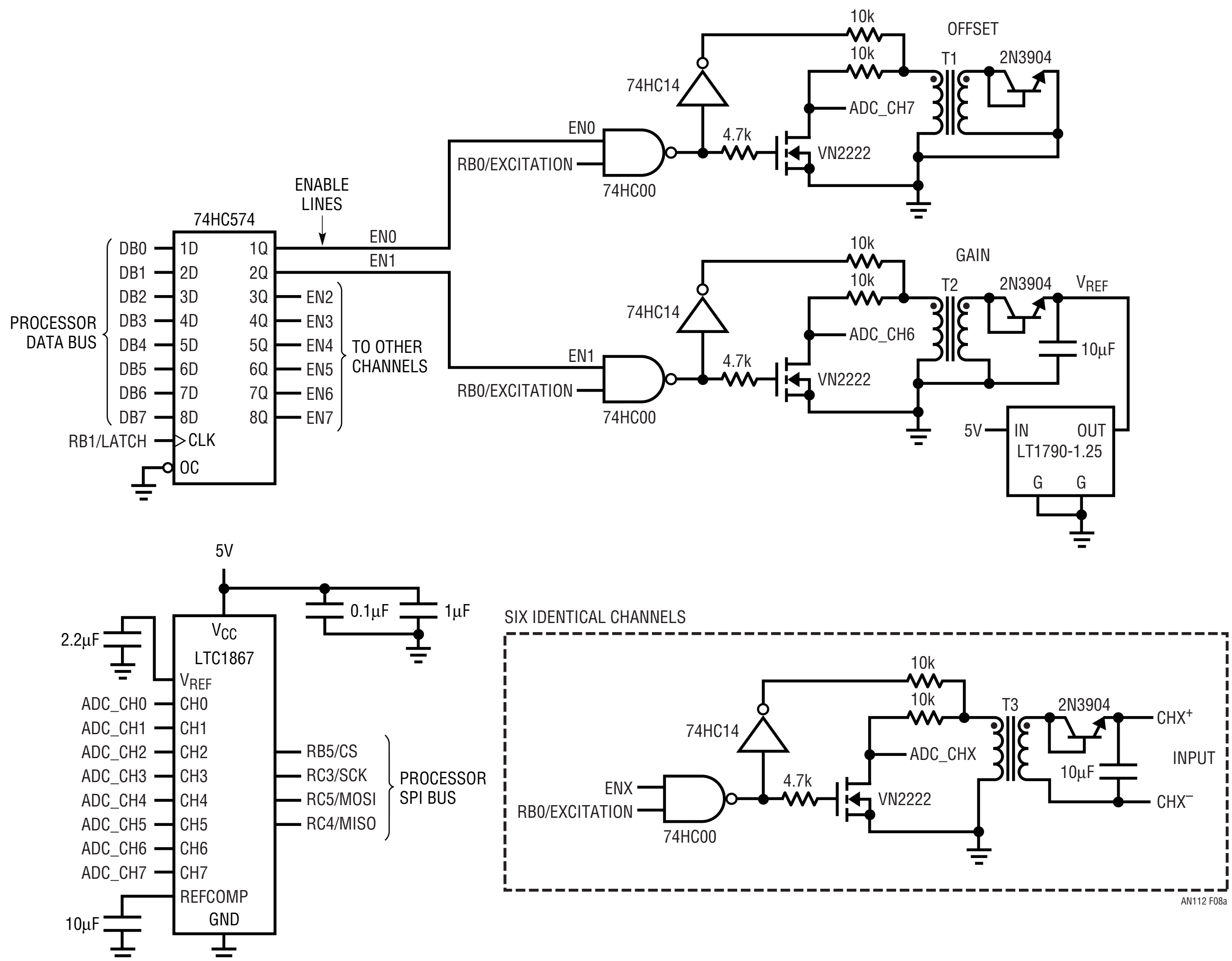


Figure 8a. Pulse Generators, Calibration Channels, Measurement Channels. ADC Calibration Channels Eliminate V_{BE} Matching Requirement and Compensate for Temperature Dependent Errors

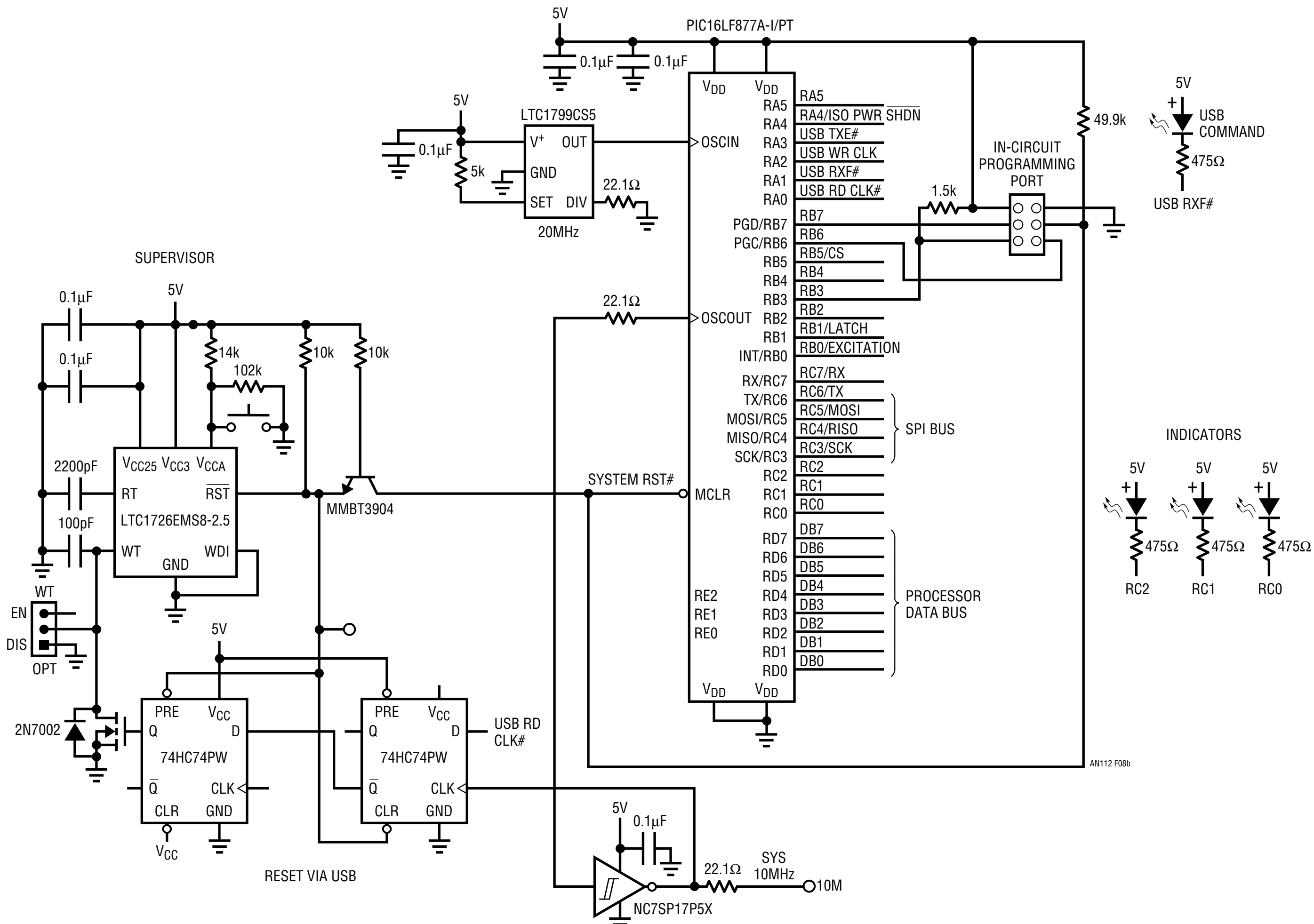


Figure 8b. Microcontroller and Reset

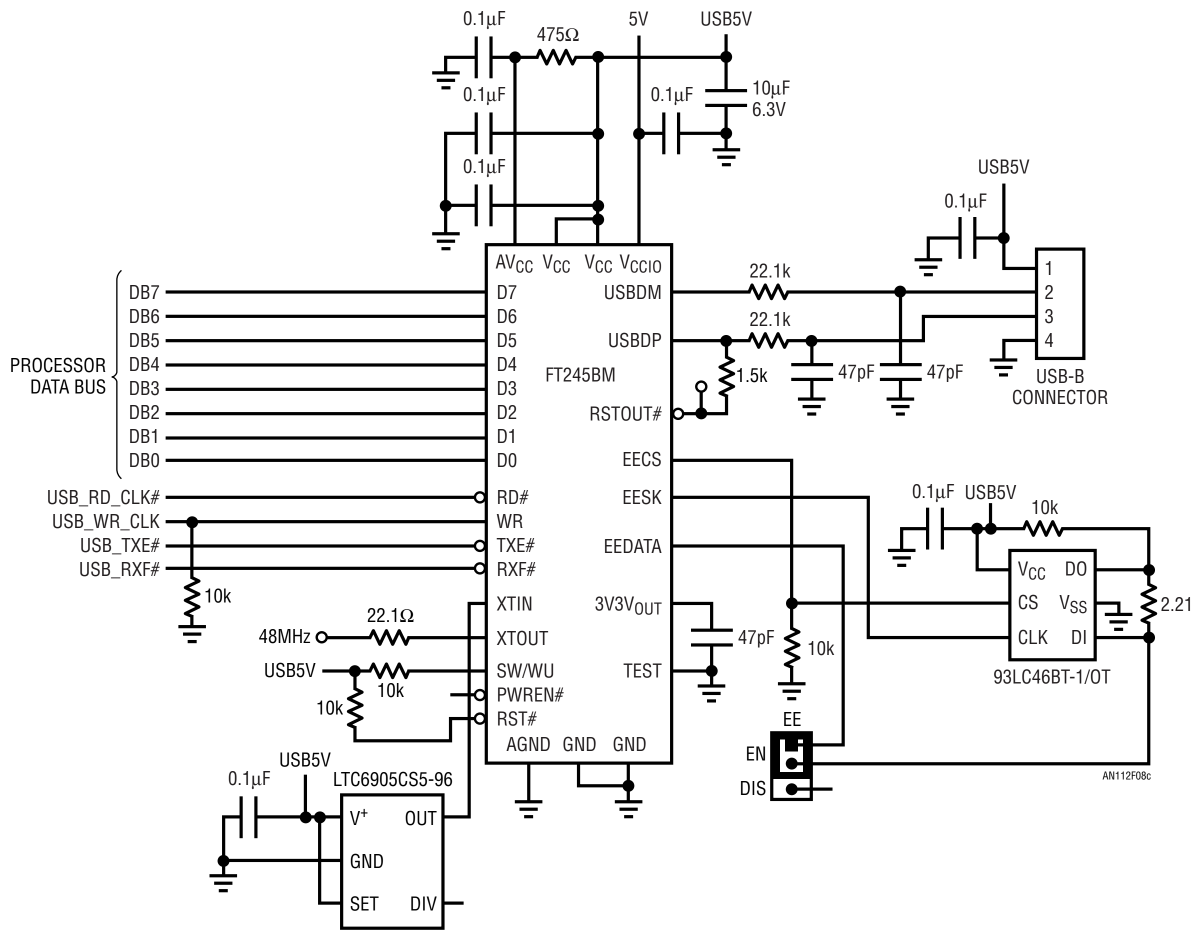


Figure 8c. USB Interface (for Development Only)

Firmware Description

The complete firmware code listing is in Appendix C. The code for this circuit is designed to be a good starting point for an actual product. Data is displayed to a PC via an FTDI FT242B USB interface IC. The PC has FTDI's Virtual Com Port drivers installed, allowing control through any terminal program. Data for all channels is continuously displayed to the terminal, and simple text commands control program operation.

A timer interrupt is called 1000 times per second. It controls the pulse generators and ADC, and stores the ADC readings to an array that can be read at any time. Thus if the main program is reading the buffer, the most out-of-date any reading will be is 1ms.

Automatic calibration routines are also included. Two functions store a zero reading and a full-scale reading for all channels, including the calibration voltages applied to channels 6 and 7, to nonvolatile memory. These are subsequently used to calibrate out the initial gain and offset errors as well as temperature dependent errors. The entire procedure is to apply zero volts to all inputs and issue a command to store the zero calibration, then apply 1.25V to all inputs and issue a command to store the full-scale calibration. Note that this is no more complicated than a basic functionality test that would be part of any manufacturing process. The 1.25V factory calibration source can be from a voltage calibrator, or from a selected "golden" LT1790-1.25 that is kept at a stable temperature.

A digital filter is also included for testing purposes. The filter is a simple exponential IIR (infinite impulse response) filter with a constant of 0.1. This reduces the noise seen in the readings by a factor of $\sqrt{10}$.

Measurement Details

To take a reading from a given channel, the processor must apply the excitation to the transformer, wait for the voltage signal to settle out, take a reading with the ADC, and then remove the excitation. This is driven by an interrupt service routine that is called once every millisecond. Refer to Appendix C for the code listing. Figure 9 shows the digital signals, excitation pulse, and clamp voltage at the ADC input along with the C code that performs these operations.[3]

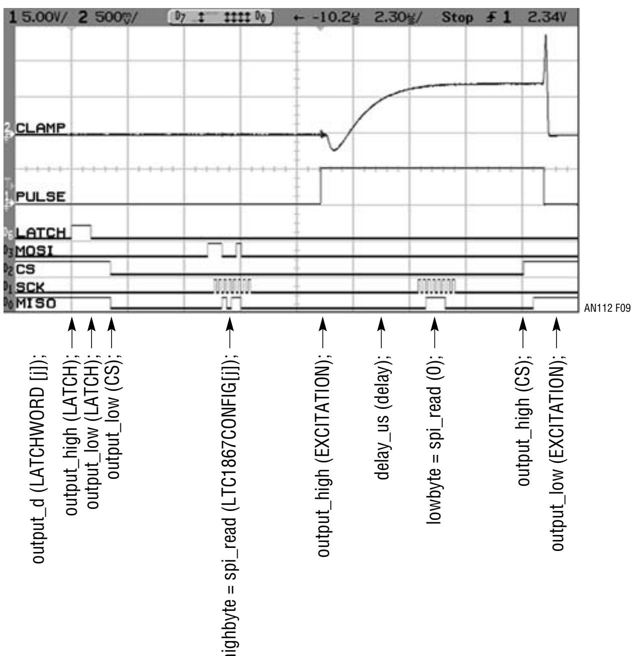


Figure 9. Pulse Generator and ADC Sequencing

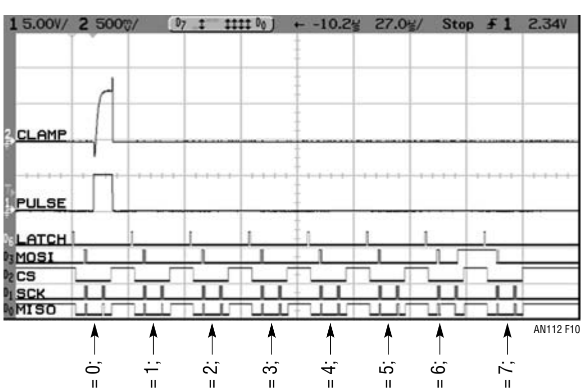


Figure 10. ISR Scanning 8 Channels

Individual channels are enabled by loading an 8 bit byte with one bit set high into the 74HC574 latch.

Note that the excitation is applied after 8 bits of the LTC1867 data are read out. This is perfectly acceptable, since there is no conversion taking place and all of the data in the LTC1867 output register is static. Depending on the specific timing of the processor being used, excitation may be applied before reading any data, in the middle of reading data, or after reading the data but before initiating a conversion. If the serial clock is very slow—1MHz for instance, applying excitation before reading any data would result in the excitation being applied for 16µs which is too long. The only constraint is that the voltage at the ADC input must have enough time to settle properly and that the excitation is not left on for too long. Figure 10 shows the same signals over the entire interrupt service routine. There are similar analog signals at each transformer and the other LTC1867 inputs.

Adding More Channels

There are lots of ways to add more channels to this circuit. Figure 11 shows a 64 channel concept. Figure 11 decodes the 64 channels into eight banks of eight channels using 74HC138 address decoders. The selected bank corresponds to one LTC1867 input that is programmed through the SPI interface. The additional analog multiplexing is done with 74HC4051 8:1 analog switches. A single 74HC4051 feeding each LTC1867 input gives 64 inputs. The LTC1867 is still a great choice in high channel count applications, rather than a single channel ADC, because it is good idea to break up multiplexer trees into several stages to minimize total channel capacitance. The LTC1867 takes care of the last stage. And with a maximum sample rate of 200ksps, it can digitize up to 200 channels at the maximum 1ksps limitation of the sense transformer. That's a lot of batteries.

[3] Sometimes a jack-of-all-trades is exactly what you need. A high speed digital designer would never dream of trading a good logic analyzer for a mixed-signal oscilloscope to test signal integrity across a complicated backplane. And its 100MHz analog channels pale in comparison to a good four channel, half-gig scope. But for testing a circuit with a microcontroller and data converters up to a few megasamples per second, a good mixed signal oscilloscope is the master of the trade.

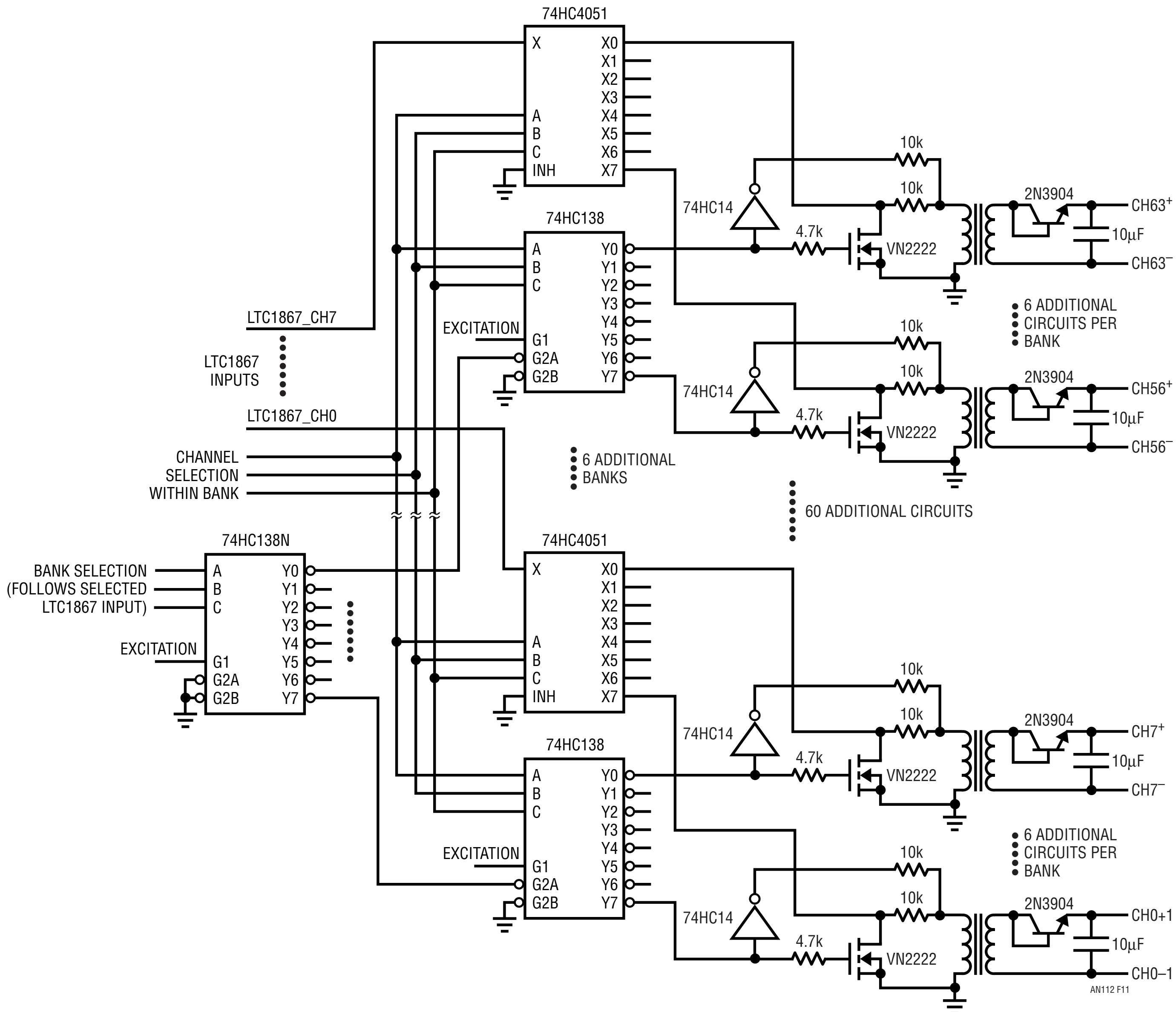


Figure 11. 64-Channel Concept

REFERENCES

1. Williams, Jim, “Transformers and Optocouplers Implement Isolation Techniques,” “Isolated Temperature Measurement,” pp.116-117. EDN Magazine (January 1982)

2. Williams, Jim, “Isolated Temperature Sensor,” LT198A Data Sheet. Linear Technology Corporation (1983)

3. Dobkin, R. C., “Isolated Temperature Sensor,” LM135 Data Sheet. National Semiconductor Corporation (1978)

4. Williams, Jim, “Isolation Techniques for Signal Conditioning,” “Isolated Temperature Measurement,” pp.1-2. National Semiconductor Corporation, Application Note 298 (May 1982)

5. Sheingold, D. H., “Transducer Interfacing Handbook,” “Isolation Amplifiers,” pp. 81-85. Analog Devices Inc. (1980)

6. Williams, Jim, “Signal Sources, Conditioners and Power Circuitry,” “0.02% Accurate Instrumentation Amplifier with 125 V_{CM} and 120dB CMRR,” pp. 11-13. Linear Technology Corporation, Application Note 98 (November 2004)

APPENDIX A

A Lot of Cut Off Ears and No Van Goghs

Things That Don't Work

The "battery stack problem" has been around a long time. Various approaches have been tried, with varying degrees of success. The problem appears deceptively simple; technically and economically qualified solutions are notably elusive. Typical candidates and their difficulties are presented here.

Figure A1 presumably solves the problem by converting cell potentials to current, obviating the high common mode voltages. Op amps feed a multiplexed input A/D; the decoded A/D output presents individual cell voltages. This approach is seriously flawed. Required resistor precision and values are unrealistic, becoming progressively more unrealistic as the number of cells in the stack increases. Additionally, the resistors drain current from the cells, a distinct and often unallowable disadvantage.

An isolation amplifier based approach appears in Figure A2. Isolation amplifiers feature galvanically floating inputs, fully isolated from their output terminals. Typically, the device contains modulation-demodulation circuitry and a floating supply which powers the signal input section[4]. The amplifier inputs monitor the cell; its isolation barrier prevents battery stack common mode voltage from corrupting output referred measurement results. This approach works quite well but, requiring an isolation amplifier per cell, is complex and quite expensive. Some simplification is possible; e.g., a single power driver servicing many amplifiers, but the method remains costly and involved.

Figure A3 employs a switched capacitor technique to measure individual cell voltage while rejecting common mode voltage. The clocked switches alternately connect the capacitor across its associated cell and discharge it into an output common referred capacitor.[5] After a number of such cycles the output capacitor assumes the cell voltage. A buffer amplifier provides the output. This arrangement rejects common mode voltages but requires many expensive high voltage switches, a high voltage level shift and nonoverlapping switch drive. More subtly, switch leakage degrades accuracy, particularly as temperature

[4]See reference 5 for details on isolation amplifiers.

[5]Old timers amongst the readership will recognize this configuration as a derivative of the venerable reed switched "flying capacitor" multiplexer.

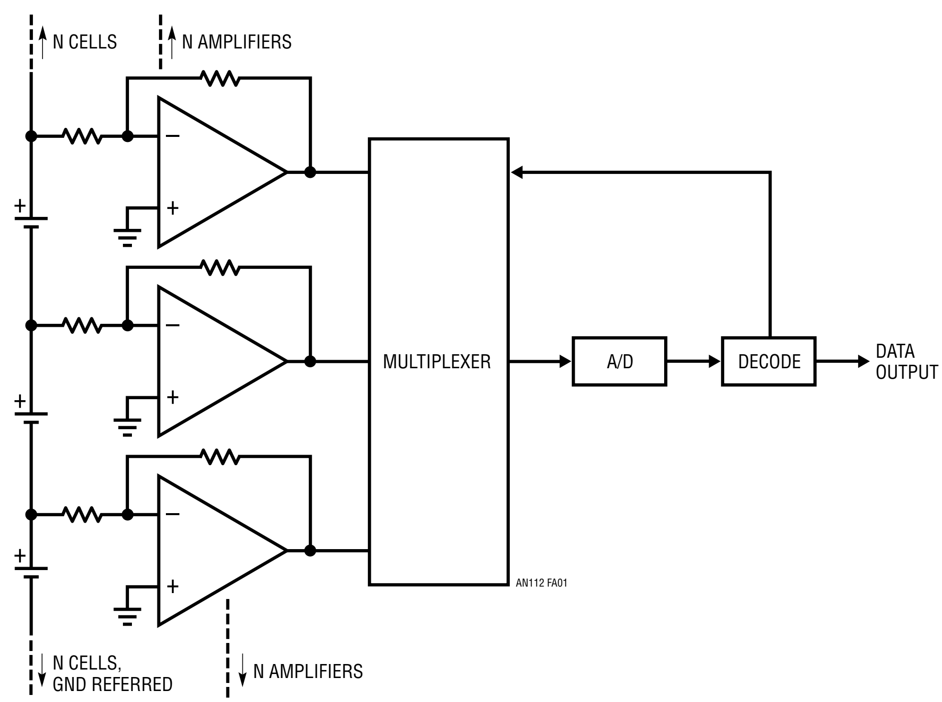


Figure A1. Unworkable Scheme Suppresses High Common Voltages by Converting Cell Potentials to Current. Circuit Decodes Amplifier Outputs to Derive Individual Cell Voltages. Required Resistor Precision and Values are Unrealistic. Resistors Draw Current from Cells

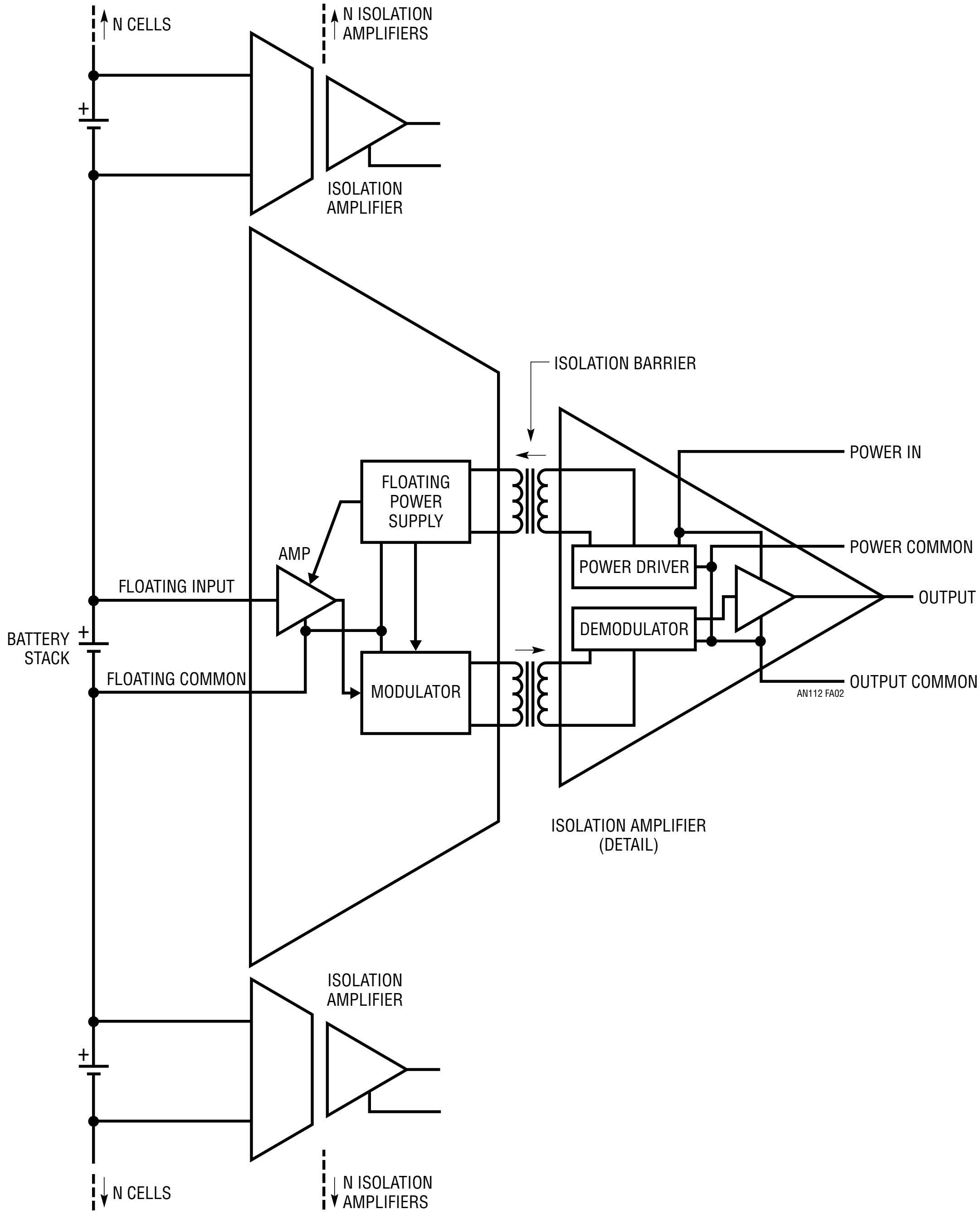


Figure A2. Isolation Amplifier's Galvanically Floating Input Eliminates Common Mode Voltage Effects. Approach Works, but is Complex and Expensive Requiring Isolation Amplifier per Cell

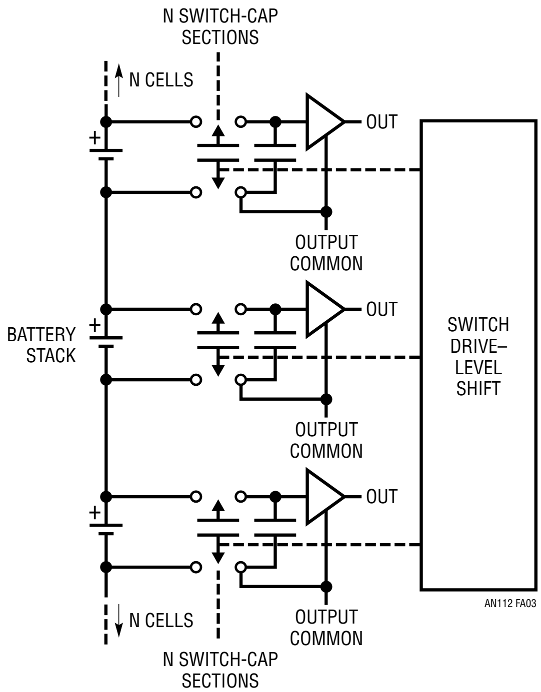


Figure A3. Switched Capacitor Scheme Rejects Common Mode Voltage but Requires High Voltage Switches, Nonoverlapping Drive and Level Shift. Switch Leakage Degrades Accuracy. Optically Driven Switches Can Simplify Level Shift but Breakdown and Leakage Issues Remain

rises. Optically driven switches, particularly those available as conveniently packaged LED driven MOSFETS, can simplify the level shift but expense, voltage breakdown and leakage concerns remain[6].

Switch related disadvantages are eliminated by Figure A4's approach. Each cell's potential is digitized by a dedicated A/D converter. A/D output is transmitted across an isolation barrier via a data isolator (optical, transformer). In its most elementary form, each A/D is powered by a separate, isolated power supply. This isolated supply population is reducible, but cannot be eliminated. Constraints include cell voltage and the A/D's maximum permissible supply and input common mode voltages. Within these limitations, several A/D channels are serviceable by one isolated supply. Further refinement is possible through employment of multiplexed input A/Ds. Even with these improvements, numerous isolated supplies are still mandated by large battery stacks. Although this scheme is technologically sound, it is complex and expensive.

[6]An optically coupled variant of this approach is given in Reference 6.

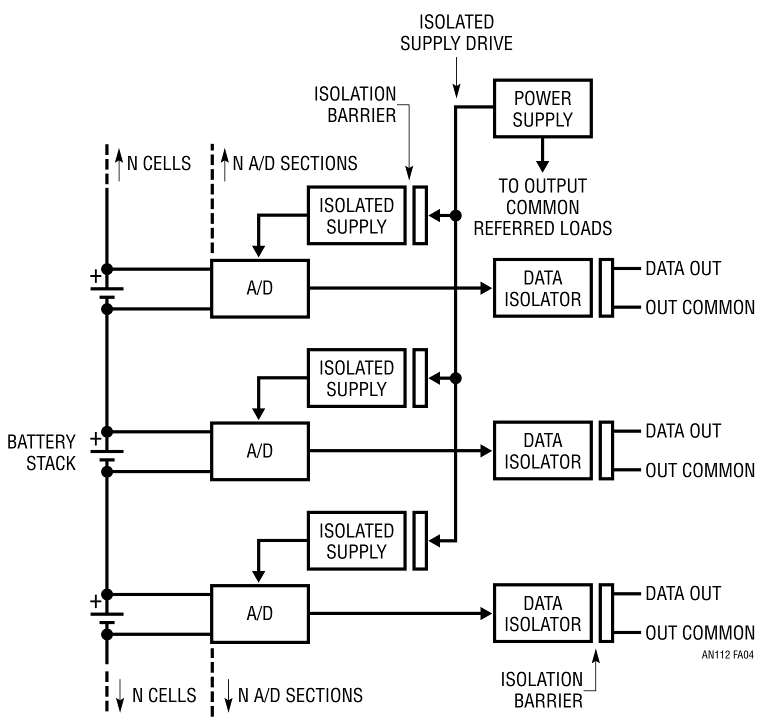


Figure A4. A/D per Cell Requires Isolated Supplies and Data Isolators. Multiplexed Input A/Ds can Minimize A/D Usage. Isolated Supply Population is Reducible, but Cannot be Eliminated

APPENDIX B

A Floating Output, Variable Potential Battery Simulator

Battery stack voltage monitor development is aided by a floating, variable potential battery simulator. This capability permits accuracy verification over a wide range of battery voltage. The floating battery simulator is substituted for a cell in the stack and any desired voltage directly dialed out. Figure B1's circuit is simply a battery-powered follower (A1) with current boosted (A2) output. The LT1021 reference and high resolution potentiometric divider specified permits accurate output settling within 1mV. The composite amplifier unloads the divider and drives a 680µF capacitor to approximate a battery. Diodes preclude reverse biasing the output capacitor during supply sequencing and the 1µF-150k combination provides stable loop compensation. Figure B2 depicts loop response to an input step; no overshoot or untoward dynamics occur despite A2's huge capacitive load. Figure B3 shows battery simulator response (trace B) to trace A's transformer clamp pulse. Closed-loop control and the 680µF capacitor limit simulator output excursion within 30µV. This error is so small that noise averaging techniques and a high gain oscilloscope preamplifier are required to resolve it.

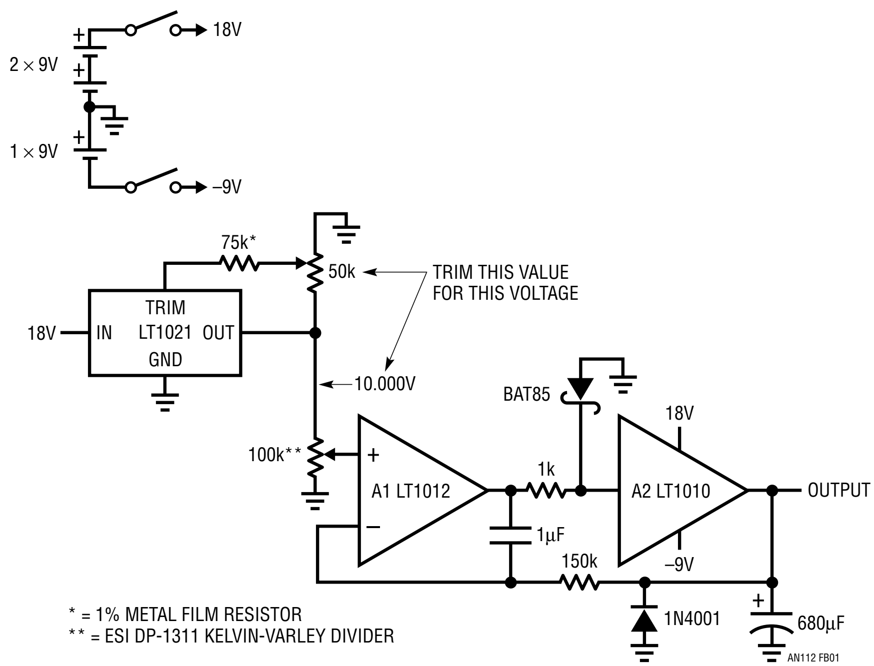


Figure B1. Battery Simulator Has Floating Output Settable Within 1mV. A1 Unloads Kelvin-Varley Divider; A2 Buffers Capacitive Load

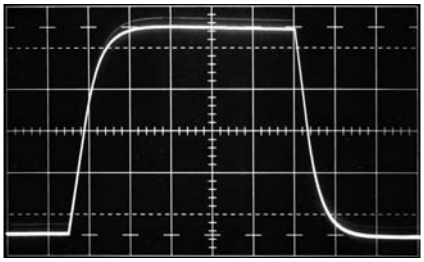


Figure 2B. 150k-1µF Compensation Network Provides Clean Response Despite 680µF Output Capacitor

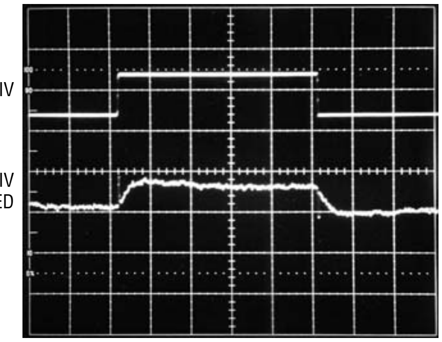


Figure 3B. Battery Simulator Output (Trace B) Responds to Trace A's Transformer Clamp Pulse. Closed-Loop Control and 680µF Capacitor Maintain Simulator Output Within 30µV. Noise Averaged, 50µV/Division Sensitivity is Required to Resolve Response

APPENDIX C

Microcontroller Code Listing

The microcontroller code consists of three files:

Battery_monitor.c contains the main program loop, including calibration and temperature correction, and support functions.

Interrupts.c is the code for the timer2 interrupt that drives the transformer excitation and controls the LTC1867 ADC.

Battery monitor.h contains various defines, global variable declarations and function prototypes.

```
/*****************************************************************************
battery_monitor.c

Six Channel Battery Monitor with continuous gain and offset
correction. Includes a "factory calibration" feature. On first power up,
apply zero volts to all inputs, allow data to settle, and type 'o'.

Next apply 1.25V to all inputs, allow data to settle, and type 'p'.

This calibrates the circuit, and it is ready to run.

Offset correction technique:
Present offset correction = init_offset[7] - voltage[7]
Hotter = less counts on voltage[7] so correction goes POSITIVE,
so ADD this to voltage[i]

voltage[i] = voltage[i] - init_offset[i] + present_offset

Slope correction Technique:
Initial slope = init_fs[6] - init_offset[7] counts per 1.25V
Present slope = voltage[6] - voltage[7] counts per 1.25V

Keyboard command summary:
'a': increment conversion period (default is 1ms)
'z': decrement conversion period
's': increment by 10
'x': decrement by 10
'd': increment pulse-convert delay (default is 2us)
'c': decrement pulse-convert delay
'f': increment pulse-convert delay by 10
'v': decrement pulse-convert delay by 10
'n': Calculate voltages for display
'm': Display raw ADC values
't': Echo text to terminal so you can insert comments into
     data that is being captured. Terminate with '!'
'k': Disable digital filter
'l': Enable digital filter
'o': Store offsets to nonvolatile memory
'p': Store full-scale readings to nonvolatile memory

Written for CCS Compiler Version 3.242
Mark Thoren
Linear Technology Corporation
January 15, 2007
*****************************************************************************/

#include "battery_monitor.h"
#include "interrupts.c"

void main(void)
   {
   int8 i;
   unsigned int16 adccode;
   float temp=0.0, offset_correction, slope, slope_correction;

   initialize();              // Initialize hardware
   rx_usb();                  // Wait until any character is received
   print_cal_constants();     // display calibration constants before starting.

   while(1)
      {
      if(usb_hit()) parse();  // get keyboard command if necessary
      for(i=0; i<=7; ++i)     // Read raw data first
         {
         readflag[i] = 1;     // Tell interrupt that we're reading!!
```

```
        adccode = data[i];
        readflag[i] = 0;
        temp = (float) adccode; // convert to floating point
        if(filter)            // Simple exponential IIR filter
           {
           voltage[i] = 0.9 * voltage[i];
           voltage[i] += 0.1* temp;
           }
        else
           {
           voltage[i] = temp;
           }
        }

     if(calculate) // Display temperature corrected voltages
        {
        // Calculate Corrections.
        // offset correction is stored CH7 reading minus the present reading
        offset_correction = read_offset_cal(7) - voltage[7];
        // Slope correction is the stored slope based on initial CH6 and CH7
        // readings divided by the present slope. Units are (dimensionless)
        slope_correction = (float) read_fs_cal(6) -
                           (float) read_offset_cal(7); // Initial counts/1.25V
        slope_correction = slope_correction / (voltage[6] - voltage[7]);

        for(i=0; i<=5; ++i)      // Print Measurement Channels
           {
           // Units on slope are "volts per ADC count"
           slope = 1.25000 / ((float) read_fs_cal(i) -     // Inefficient but
                             (float) read_offset_cal(i)); // we are RAM limited
           // Correct for initial offset and temperature dependent offset.
           // units on temp are "ADC counts"
           temp = voltage[i] - (float) read_offset_cal(i) + offset_correction;
           // Correct for initial slope
           temp = temp * slope;
           // Units on temp is now "volts"
           // Correct for temperature dependent slope
           temp = temp * slope_correction;
           busbusy = 1;
           printf(tx_usb, "%1.5f, ", temp);
           busbusy = 0;
           }
        busbusy = 1;      // Print to terminal
        printf(tx_usb, "%1.6f, %1.1f, ", slope_correction, offset_correction);
        busbusy = 0;
        }

     else // Display raw ADC counts
        {
        for(i=0; i<=7; ++i)
           {
           busbusy = 1;     // Print to terminal
           printf(tx_usb, "%1.0f, ", voltage[i]);
           busbusy = 0;
           }
        }

     busbusy = 1;
     printf(tx_usb, "D:%d, P:%d\r\n", delay, period); // print period and delay
     busbusy = 0;
     // Delay and blink light
     delay_ms(100); output_high(PIN_C0); delay_ms(100); output_low(PIN_C0);
     } //end of loop
  } //end of main
```

```
/*******************************************************************************
Parse keyboard commands
arguments: none
returns: void
*******************************************************************************/

void parse(void)
   {
   char ch;
   switch(rx_usb())
      {
      case 'a': period += 1; break;    // increment period
      case 'z': period -= 1; break;    // decrement period
      case 's': period += 10; break;   // increment by 10
      case 'x': period -= 10; break;   // decrement by 10
      case 'd': delay += 1; break;     // increment pulse-convert delay
      case 'c': delay -= 1; break;     // decrement pulse-convert delay
      case 'f': delay += 10; break;    //    "   by 10
      case 'v': delay -= 10; break;    //    "   by 10
      case 'n': calculate = 1; break;  // Calculate voltages
      case 'm': calculate = 0; break;  // Display raw values
      case 't':   // Echoes text to terminal so you can insert comments into
         {        // data that is being captured. Terminate with '!'
         busbusy = 1;
         printf(tx_usb, "enter comment\r\n");
         while((ch=rx_usb())!='!') tx_usb(ch);
         tx_usb('\r');
         tx_usb('\n');
         busbusy = 0;
         } break;
      case 'k': filter = 0; break;     // Disable filter
      case 'l': filter = 1; break;     // Enable Filter
      case 'o': write_offset_cal(); break;   // Store offset to nonvolatile mem.
      case 'p': write_fs_cal(); break;       // Store FS to nonvolatile mem.
      }
   setup_timer_2(T2_DIV_BY_16,period,8);     // Update period if necessary
   }

/*******************************************************************************
write offset and full-scale calibration constants to non-volatile memory
arguments: none
returns: void
*******************************************************************************/
void write_offset_cal(void)
   {
   int i;
   unsigned int16 intvoltage;
   for(i=0; i<=7; ++i)
      {
      intvoltage = (unsigned int16) voltage[i];    // Cast as unsigned int16
      write_eeprom (init_offset_base+(2*i), intvoltage >> 8); // Write high byte
      delay_ms(20);
      write_eeprom (init_offset_base+(2*i)+1, intvoltage);  // Write low byte
      delay_ms(20);
      }
   }

void write_fs_cal(void)
   {
   int i;
   unsigned int16 intvoltage;
   for(i=0; i<=7; ++i)
      {
```

```
      intvoltage = (unsigned int16) voltage[i];    // Cast as unsigned int16
      write_eeprom (init_fs_base+(2*i), intvoltage >> 8);  // Write high byte
      delay_ms(20);
      write_eeprom (init_fs_base+(2*i)+1, intvoltage);  // Write low byte
      delay_ms(20);
      }
   }

/*******************************************************************************
read offset and full-scale calibration constants from non-volatile memory
arguments: none
returns: void
*******************************************************************************/
unsigned int16 read_offset_cal(int channel)
   {
   return make16(read_eeprom(init_offset_base+(2*channel)),
                 read_eeprom(init_offset_base+(2*channel)+1));
   }

unsigned int16 read_fs_cal(int channel)
   {
   return make16(read_eeprom(init_fs_base+(2*channel)),
                 read_eeprom(init_fs_base+(2*channel)+1));
   }


/*******************************************************************************
Print calibration constants (raw ADC counts)
arguments: none
returns: void
*******************************************************************************/
void print_cal_constants(void)
   {
   int i;
   for(i=0; i<=7; ++i)
      {
      printf(tx_usb, "ch%d offset: %05Lu, fs: %05Lu\r\n"
      , i, read_offset_cal(i),read_fs_cal(i));
      }
   }

/*******************************************************************************
Interface to the FT24BM USB controller

usb_hit()   arguments: none  returns: 1 if data is ready to read, zero otherwise
rx_usb() arguments: none returns: character from USB controller
tx_usb() argments: data to send to PC, returns: void
*******************************************************************************/
char usb_hit(void)
   {
   return !input(RXF_);
   }

char rx_usb(void)
   {
   char buf;
   while(input(RXF_)) {} // Low when data is available, wait around
   output_low(RD_);
   delay_cycles(1);
   buf=input_d();
   output_high(RD_);
   return(buf);
   }
```

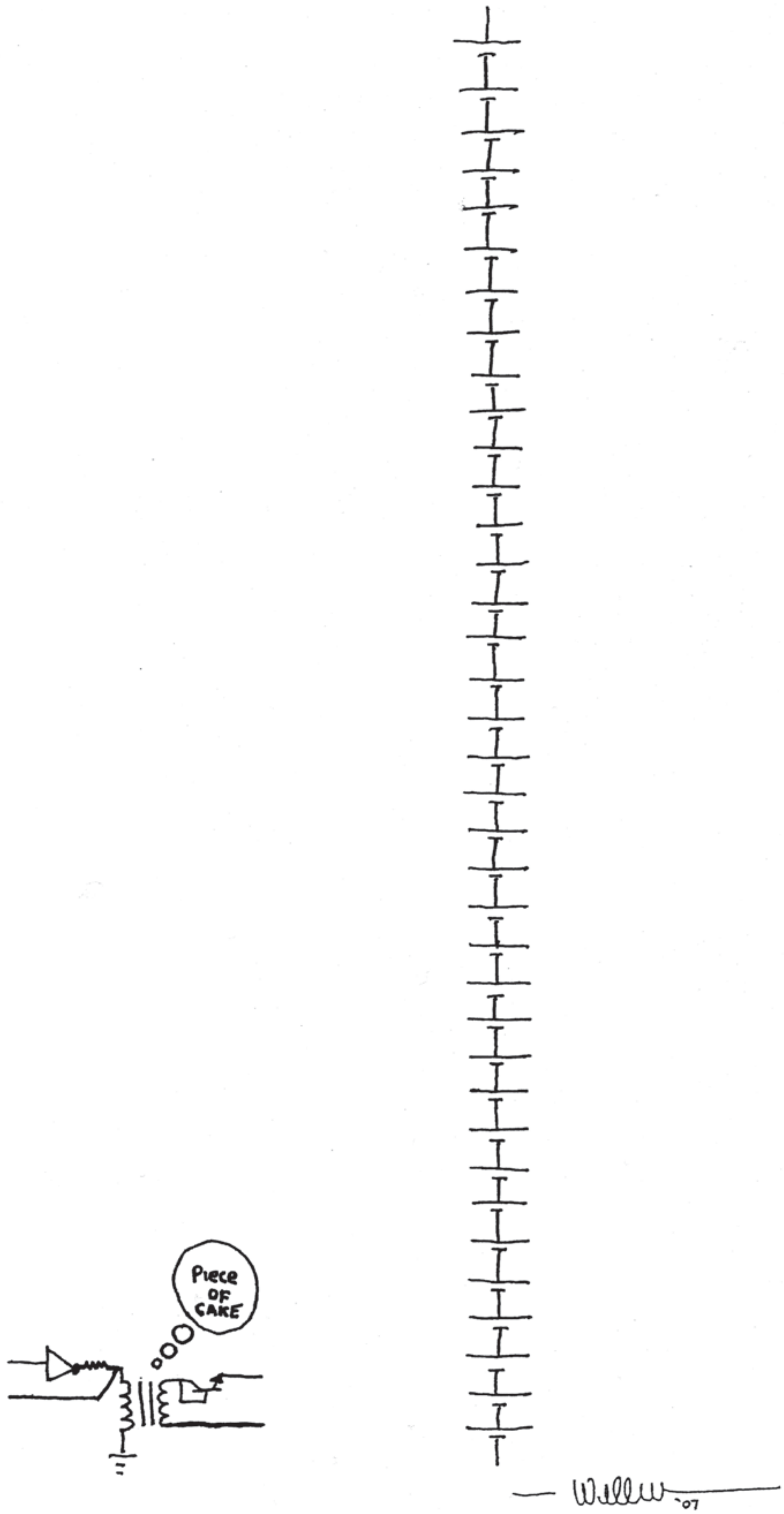
```
void tx_usb(int8 value)
   {
   while(input(TXE_))    //Low when FULL, wait around
      {
      }
   output_d(value);
   output_high(WR);
   delay_cycles(1);
   output_low(WR);
   input_d();
   }

/********************************************************************************
Hardware initialization
arguments: none
returns: void
********************************************************************************/
void initialize(void)
   {
   output_high(ISO_PWR_SD_);  //turn on power
   setup_adc_ports(NO_ANALOGS);
   setup_adc(ADC_OFF);
   setup_psp(PSP_DISABLED);
   setup_spi(SPI_CONFIG);
   CKP = 0; // Set up clock edges - clock idles low, data changes on
   CKE = 1; // falling edges, valid on rising edges.
   output_low(I2C_SPI_);
   output_low(AUX_MAIN_); // SPI is only MAIN
   setup_counters(RTCC_INTERNAL,RTCC_DIV_1);
   setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
   setup_timer_1(T1_DISABLED);
   setup_timer_2(T2_DIV_BY_16,period,8);
   setup_comparator(NC_NC_NC_NC);
   setup_vref(FALSE);

   output_low(PIN_C0);
   delay_ms(100);
   output_high(PIN_C0); // Turn off LEDs
   output_high(PIN_C1);
   output_high(PIN_C2);

// I/O Initialization
   input(RXF_);
   input(TXE_);
   output_high(RD_);
   output_low(WR);
   delay_ms(100);
   output_low(CS);
   delay_us(5);
   output_high(CS);
// Turn on interrupts (only one)
   enable_interrupts(INT_TIMER2);
   enable_interrupts(GLOBAL);
   }
```

```
/***************************************************************************
Timer 2 Interrupt
This interrupt service routine does all of the work of controlling transformer
excitation and controlling the LTC1867.
***************************************************************************/
#int_TIMER2       // Tell compiler that this is the Timer 2 ISR
TIMER2_isr()
   {
   static int8 ledstatus;
   int8 j, highbyte, lowbyte;
   if(++ledstatus == 0x80) output_low(LED);     // Blink light every 256 calls
   if(ledstatus == 0x00) output_high(LED);

   if(!busbusy) // If main() is using the bus, do nothing.
      {
      for(j=0; j<=7; ++j)
         {
         output_d(LATCHWORD[j]);                // Place excitation data on the bus
         output_high(LATCH);                    // Latch in data
         output_low(LATCH);
         output_low(CS);                        // Enable LTC1867 serial interface
         highbyte = spi_read(LTC1867CONFIG[j]); // Read out high byte.
                                                // Acquisition begins on 6th falling clock edge
         output_high(EXCITATION);               // Apply transformer excitation
         delay_us(delay);                       // Wait for analog signal to settle
         lowbyte = spi_read(0);                 // Finish reading data. Input is also settling
                                                // During this time.
         output_high(CS);                       // Start conversion!!!
         output_low(EXCITATION);                // Remove excitation. One instruction cycle is plenty
                                                // of "analog hold time"
         if(!readflag[j]) data[j] = make16(highbyte, lowbyte); // Don't write if main() is reading!!
                                                // This is a simple anti-collision technique. The worst
                                                // case latency is a single reading, or 1ms.
         }//end of for loop
      }//end of if(!busbusy)
   }// end of ISR
```

```
/******************************************************************************
battery_monitor.h
defines, global variables, function prototypes
******************************************************************************/

#include <16F877A.h>    // Standard header
#device adc=8
#use delay(clock=20000000) // Clock frequency is 20MHz
#use rs232(baud=9600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=9)
#define SPI_CONFIG SPI_MASTER|SPI_L_TO_H|SPI_CLK_DIV_4   // 5MHz SPI clk when
                                                          // master clk = 20MHz

//#fuses NOWDT,RC, NOPUT, NOPROTECT, NODEBUG, BROWNOUT, LVP, NOCPD, NOWRT
// This is less confusing - set up configuration word with #rom statement
//   Bit     13 12   11    10    9   8   7    6   5 4    3      2     1     0
// Function  CP -- DEBUG WRT1 WRT0 CPD LVP BOREN - - PWRTEN# WDTEN FOSC1 FOSC0
//
#rom 0x2007 = {0x3F3A}

/////////////////////////////////////
// Battery Monitor Project Defines //
/////////////////////////////////////

// Global variables
int16 data[8];              // Raw data from the LTC1867
int8 readflag[8];          // Tells ISR that main is reading data, do not write
int1 busbusy = 0;           // Tells ISR that main is talking on the bus
int1 calculate = 1;         // Send calculated voltages to terminal when asserted
int1 filter = 1;           // Enables digital filter when asserted
unsigned int8 period = 40; // Period between reads
unsigned int8 delay = 2;    // Additional settling time after applying excitation
float voltage[8];          // Holds floating point calculated voltages

// Non-volatile memory base addresses for calibration constants
#define init_offset_base 0
#define init_fs_base 16

// First, define the SDI words to be sent to the LTC1867
// All are Single ended, unipolar, 4.096V range.
#define LTC1867CH0   0x84
#define LTC1867CH1   0xC4
#define LTC1867CH2   0x94
#define LTC1867CH3   0xD4
#define LTC1867CH4   0xA4
#define LTC1867CH5   0xE4
#define LTC1867CH6   0xB4
#define LTC1867CH7   0xF4

// Excitation enable lines. Write this to the '574 register
// before enabling excitation pulse.
#define EXC0   0x01
#define EXC1   0x02
#define EXC2   0x04
#define EXC3   0x08
#define EXC4   0x10
#define EXC5   0x20
#define EXC6   0x40
#define EXC7   0x80
```

```
// Now define two lookup tables such that the excitation signal lines up with
// the selected LTC1867 input.
byte CONST LTC1867CONFIG [8] = {LTC1867CH1, LTC1867CH2, LTC1867CH3, LTC1867CH4,
                                LTC1867CH5, LTC1867CH6, LTC1867CH7, LTC1867CH0};
byte CONST LATCHWORD [8] = {EXC6, EXC5, EXC4, EXC3, EXC2, EXC1, EXC0, EXC7};

//Pin Definitions
#define EXCITATION   PIN_B0   // Enables excitation to the selected channel
#define LATCH        PIN_B1   // 74HC573 latch pin
#define LED          PIN_C1   // Spare blinky light

#define RD_          PIN_A0
#define RXF_         PIN_A1
#define WR           PIN_A2
#define TXE_         PIN_A3
#define ISO_PWR_SD_  PIN_A4
#define LCD_EN       PIN_A5
#define CS           PIN_B5

#define AUX_MAIN_    PIN_E1
#define I2C_SPI_     PIN_E2

#byte SSPCON   = 0x14
#byte SSPSTAT  = 0x94
#bit CKP             = SSPCON.4
#bit CKE             = SSPSTAT.6

// Function Prototypes
void parse(void);
void write_offset_cal(void);
void write_fs_cal(void);
unsigned int16 read_offset_cal(int channel);
unsigned int16 read_fs_cal(int channel);
void print_cal_constants(void);
char usb_hit(void);
void initialize(void);
void tx_usb(int8 value);
char rx_usb(void);
```


Information furnished by Linear Technology Corporation is believed to be accurate and reliable. However, no responsibility is assumed for its use. Linear Technology Corporation makes no representation that the interconnection of its circuits as described herein will not infringe on existing patent rights.



Linear Technology Corporation
1630 McCarthy Blvd., Milpitas, CA 95035-7417
(408) 432-1900 • FAX: (408) 434-0507 • www.linear.com

LT 0307 • PRINTED IN USA

© LINEAR TECHNOLOGY CORPORATION 2007