AN184

Digital Counting Scale

Relevant Devices

This application note applies to the following devices:
C8051F350, C8051F351, C8051F352, and C8051F353.

1. Introduction

The C8051F350 provides a low-cost system-on-chip solution for digital counting/measurement scales such as postal scales, deli scales, and analytical scales. Using the F350, the entire measurement system can be implemented using a few passive components, an LCD driver, and an LCD.

This application note provides an example of a digital counting scale design, including design considerations, using the C8051F350.

This reference design includes the following:

- Background and theory of operation
- Hardware and software description
- Typical performance examples
- Complete firmware (developed in C)

2. Background

Bridge transducers are the most common sensor type in digital counting scales. They convert a force into a voltage that is proportional to the force applied to the bridge. Bridge transducers are used because they are extremely linear and have repeatable characteristics for large applied forces. Their only real drawback is low sensitivity, which typically ranges between 1 mV/V to 10 mV/V.

Transducer manufacturers exhaustively characterize their sensors to ensure they have linear transfer functions for the region where they are specified for use. Bridge transducers are manufactured with various technologies ranging from diffused silicon to bonded foil materials. Each technology has its advantages and disadvantages concerning its sensitivity, linearity, and thermal stability. Consult the various manufacturer's data sheets to select the transducer that best meets the application's requirements.

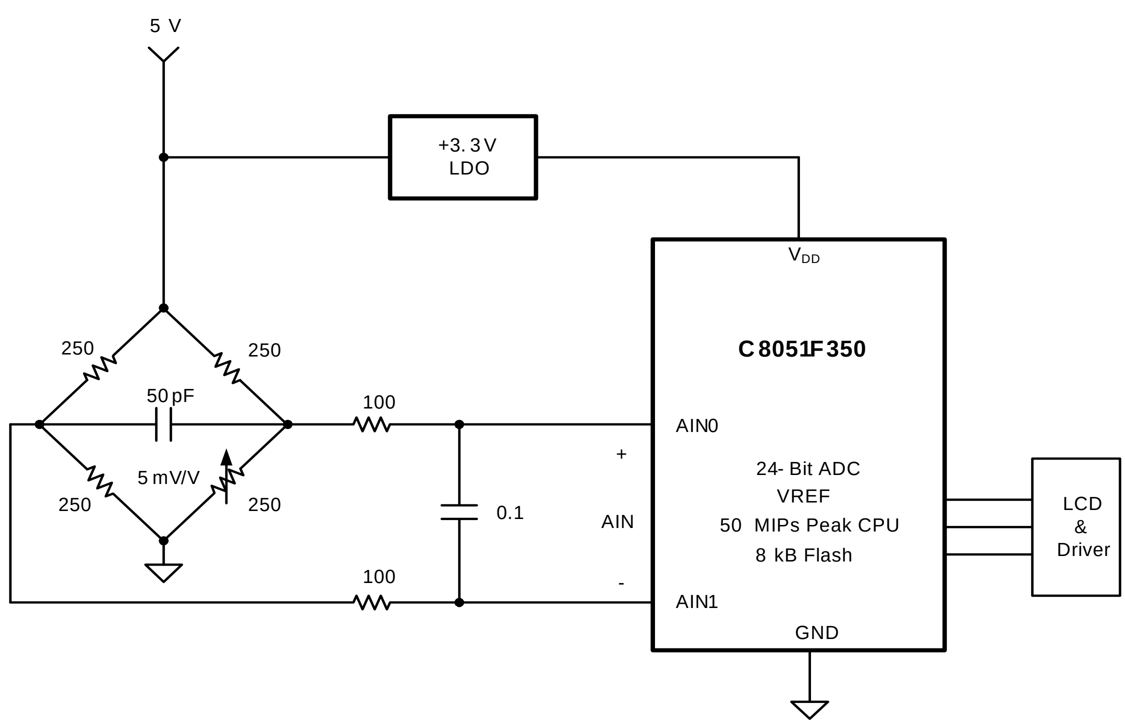


Figure 1. C8051F350 Bridge Transducer Circuit

3. Theory of Operation

Bridge transducers are ratiometric sensors.Therefore, their full scale output voltage depends on the voltage used to excite the sensor (i.e., its excitation voltage). 5 V is a common excitation voltage. For example, a 5 lb. bridge transducer with a 10 mV/V sensitivity excited with a 5 V supply will ideally produce a 50 mV output voltage when a 5 lb. weight is placed on the transducer.

$$\text{FullScaleout} = \frac{5\,\text{lb}}{5\,\text{lb}} \times 5\,\text{V} \times \frac{10\,\text{mV}}{\text{V}} = 50\text{mV}$$

Equation 1. Digital Output Code

Relatively speaking, 50 mV is a small signal. To compound the issue, high precision scales will divide this 50 mV full scale output signal into 10,000 divisions or 5 µV. This 5 µV equates to one unit of measure on the scales output display (perhaps 1 oz.). Now, 5 µV is a very small signal to resolve. The modern way to resolve such an analog signal is to use an analog-to-digital converter (ADC), preferably a 24-bit ADC, with an least significant bit (LSB) that is smaller than the signal to be resolved. A common full scale range for most ADCs today is 2.5 V, which is set by the voltage reference of the system. Equation 2 illustrates that one 24-bit LSB (least significant bit or 1 ADC count), with a 2.5 V reference, is 149 nV.

$$\text{Dout} = \frac{(\text{Ain})2^n}{\text{Vref}} = 149\,\text{nV}$$

Equation 2. Digital Output Code

In this example, the 24-bit ADC's 149 nV LSB easily resolves the 5 µV minimum output level of the bridge and provides significant overhead to account for other system issues such as temperature drift and offset error.

A 16-bit ADC can also be used to resolve the 5 µV minimum transducer voltage. However, an amplifier is required in this system as the LSB of a 16-bit ADC is only 38 µV (2.5 V V_{REF}). By amplifying the bridge transducer's output voltage by eight and increasing the transducer's minimum output voltage to 40 µV, the combo amplifier plus 16-bit ADC provides enough resolution to resolves 10,000 counts. Note that in a 16-bit system, a 8x amplifier is marginally adequate to resolve 5 µV. Most designers increase this gain and design the amplifier's gain stage to match the full scale input range of the ADC. In this 16-bit example, a 50x gain would be used as 50 mV times 50 is 2.5 V, the typical full scale input of an ADC.

Here is one final comment when designing a high accuracy counting scale, system designers need to consider other parasitic board level issues like amplifier noise and drift and parasitic thermocouple effects on the system board. Thermocouple effects from common tin/lead solder can be as high as 3 µV/°C. On a 10,000 count scale exposed to a temperature gradient of 10 degrees, the scales could drift several counts.

4. Architectural Description

The software developed in this note for digital counting scales is based around the C8051F350. The F350 is a fully integrated mixed-signal system-on-a-chip MCU. The flexibility of the F350 allows for quick customizing for a particular application's counting scale. Highlighted features of the F350 are listed here. For more information, refer to the device data sheet.

- High-Speed 8051-compatible micro controller core (up to 50 MIPs peak)
- In-system, full-speed, non-intrusive debug interface
- True 24-Bit, ADC, with 17-Channel analog multiplexer
- Two 8-Bit, 2 mA IDACs
- 8 kB of on-chip Flash memory
- 768 bytes of on-chip RAM
- SMBus/I2C and Enhanced UART serial interfaces implemented in hardware
- Four general-purpose 16-bit timers
- Programmable counter/timer array (PCA) with three capture/compare modules and watchdog timer function
- On-chip power-on reset, and V_{DD} monitor
- On-chip temperature sensor
- On-chip voltage comparator
- Two byte-wide I/O port (5 V tolerant)

Figure 1 illustrates a C8051F350-based digital counting scale. The circuit consists of a bridge transducer, a low pass filter, an LDO, an LCD driver and an LCD. The C8051F350DK evaluation board was used to develop and test the code. A 100 mV precision dc signal source from an Agilent E3630A was used to simulate the bridge transducer. The Silicon Laboratories C2 interface on the evaluation board aided in developing/debugging the software.

5. Software

The software works as follows. First, the software function *main*() calls *Config_F350*() where it configures the C8051F350's Cross Bar, oscillator, ADC, and timers. After the device is configured, *main*() calls *CalibrateADCforMeasurement*() where a two point calibration routine, self-offset calibration followed by self-gain calibration, is performed. After errors from the sampling system are removed, the MCU calculates the size of one unit of measure via the function *Calculate_One_Count().* Finally, main() enters an infinite while(1) loop. In this loop, the MCU monitors the system temperature and the output voltage from the bridge transducer. After the temperature is taken and the current count on the scales is computed, the MCU updates the output display. The system code developer has a choice of output displays: the UART (at 9600, N, 8, 1) or an LCD driver. The UART is the default output display. Software flow charts are detailed in Figure 2 though Figure 6.

5.1. Device Calibration

On first run, the Silicon Labs digital counting scale software goes through a calibration sequence to remove measurement errors, such as offset, from the sampling system. *CalibrateADCforMeasurement*() performs this task.

To ensure that the on-chip ADC contributes minimal error to the measurement and that an accurate count is acquired, these algorithms use a two-point self-calibration scheme. In this scheme, a self-offset calibration is performed followed by a self-gain calibration. Offset calibration eliminates any ADC offset. Gain calibration eliminates any slope error in the ADC transfer function. For better accuracy, the user can quickly modify the code to perform system calibration to eliminate errors from the on-chip gain amplifier or an external amplifier. Refer to the F350 data sheet for details. If system calibration is desired, the user is expected to apply two known voltages, preferably one near ground and one near full-scale.

Once self calibration is complete, the software stores these coefficients in Flash eliminating the need to calibrate the ADC in the future.

Calibration is most effective at slow output word rates. If more than one gain setting is used and system calibration is implemented, system calibration should be performed at each gain setting to eliminate errors between the different gain settings.

5.1.1. Caliculate_One_Count()

Calculate_One_Count() effectively performs a system calibration for one unit count. The unit to be counted can be just about anything: a penny, a pencil, an integrated circuit package. The only requirement is that each unit be uniform and fairly consistent in weight.

To calculate the equivalent number of ADC counts that is equal to 1 unit, the user first places the tare weight, the "unloaded weight", on the bridge. In this example, it is 0 V (in real world applications, the tare weight might be the bucket at the grocery store that holds the produce being weighed). This value is then stored in RAM. Then the user applies 100 mV (in the real world it might be 100 pennies). The software then takes another conversion (actually five conversions are taken and averaged to minimize noise and provide a better calibration point) and stores it in RAM. Then, the software computes the number of ADC codes equivalent to 1 Unit (in our case 1 mV; in real word applications it might be 1 gram). These coefficients are then stored in Flash for future calculations.

5.2. ADC Sampling

Once the number of ADC counts required for one unit is computed and errors from the system have been eliminated via self-calibration or system-calibration, the software uses further data conversions to compute the current count (total number of units) on the bridge transducer.

Monitor_Weigh_Scale () is the subroutine that monitors the transducer. It monitors the bridge via the on-chip 24-bit ADC0 at a sample rate of approximately 23 Hz.

Temperature—monitored main() while loop

Using the on-chip temperature sensor, main()'s *while* loop also monitors the temperature every cycle. Note that the on-chip temperature sensor is left uncalibrated. For more accurate temperature measurements, a one or two point temperature calibration may be used. Further note that an external temperature sensor can be used.

Update Display—updated in main() while loop

The *Update_Display_Via_UART()* subroutine is called once every cycle of the infinite while loop. The code can be modified to choose between updating via the PC over UART or updating via LCD.

This function provides starter algorithms for the HT1620 (Holtek) LCD driver. It provides algorithms to write to the HT1620 and read from the HT1620. Algorithms to activate and deactivate the elements on the LCD that correspond to specific letters and/or numerals will need to be added, based on the implementation-specific LCD chosen.

6. Conclusion

The C8051F350's high level of integration and small form-factor makes it ideal for digital counting scale applications. This note discussed how to use the C8051F350 family and its on-chip 24-bit analog-to-digital converter in digital scale applications that use a bridge transducer for the sensor. Example code is provided.

Appendix A—Software Flow Diagrams

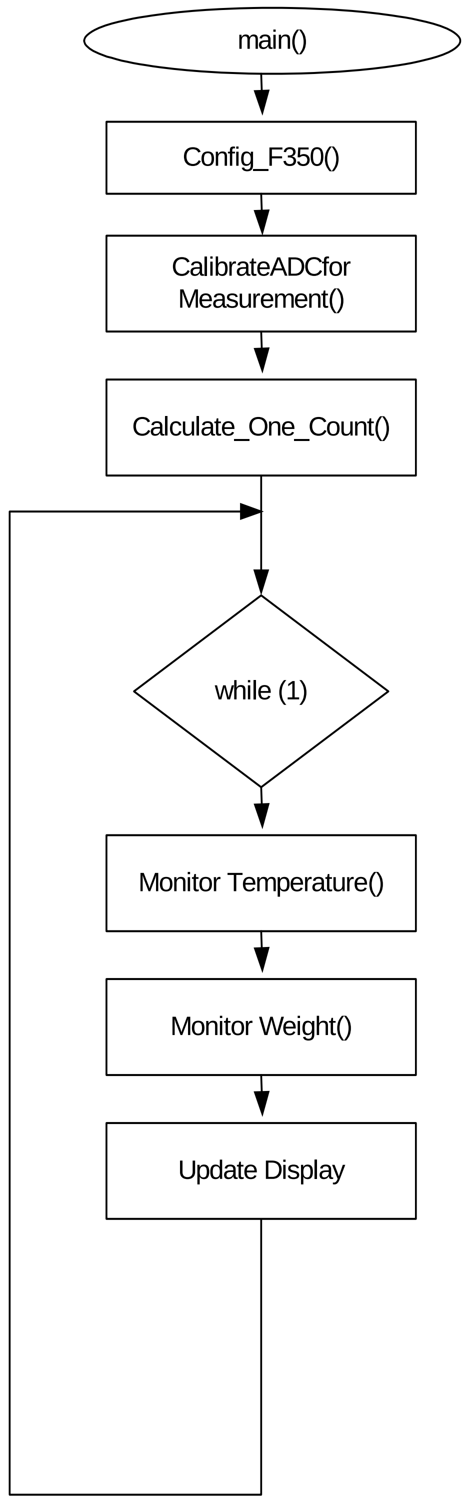


Figure 2. ***Main*****() Flow Chart.**

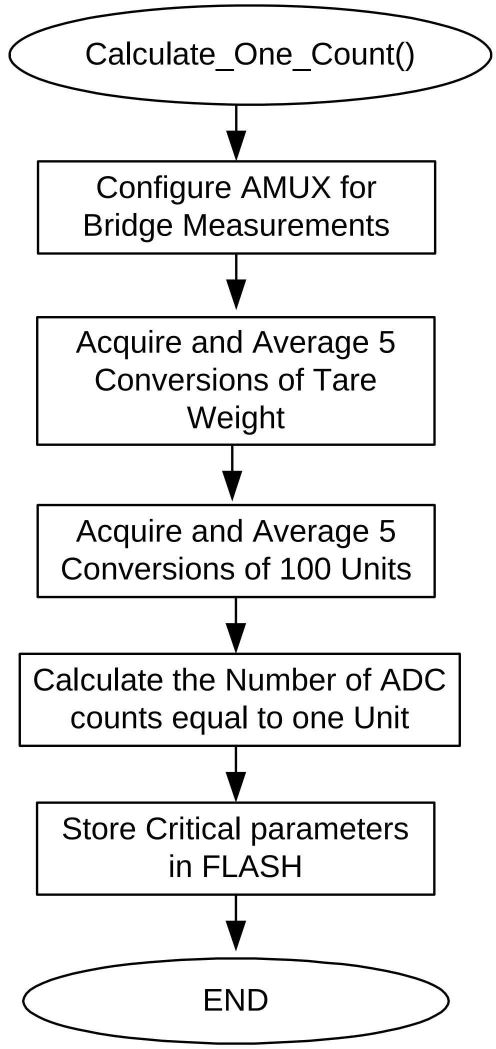


Figure 3. *Calculate_One_Count*() Flow Chart.

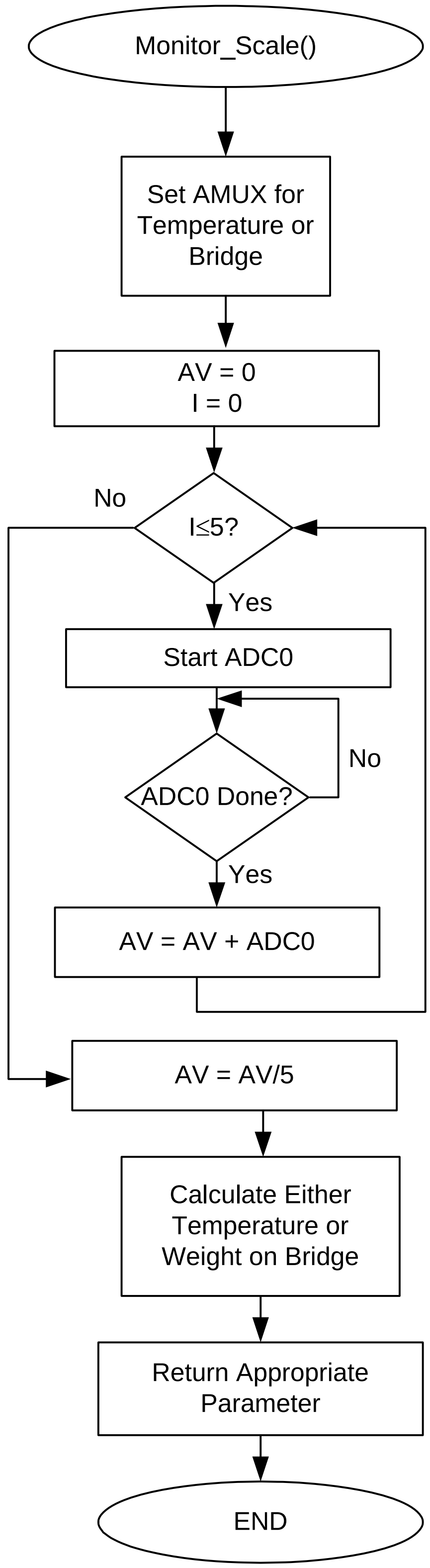


Figure 4. *Monitor_Weigh_Scale*() Flow Chart.

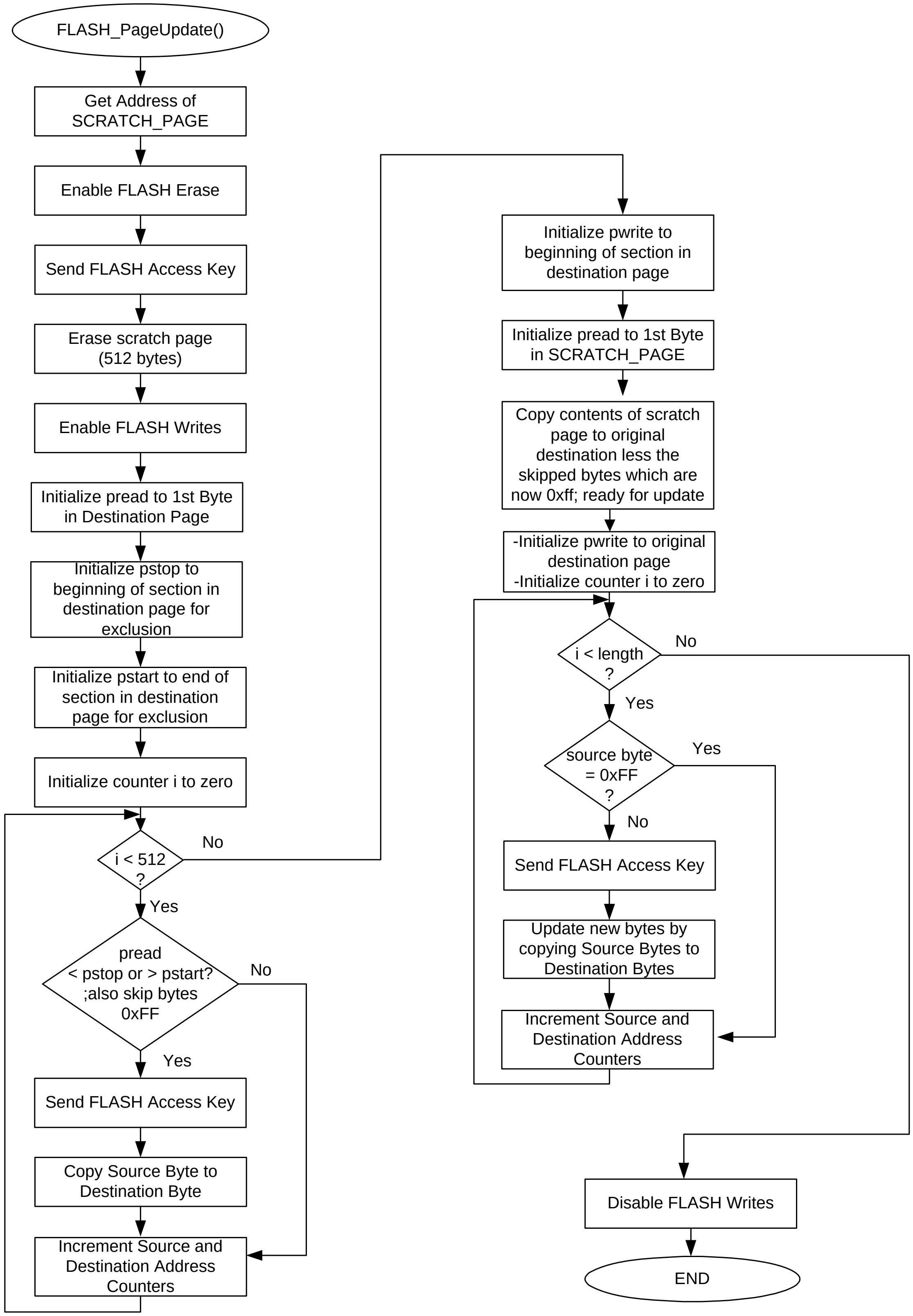


Figure 5. *FLASH_PageUpdate*() Flow Chart.

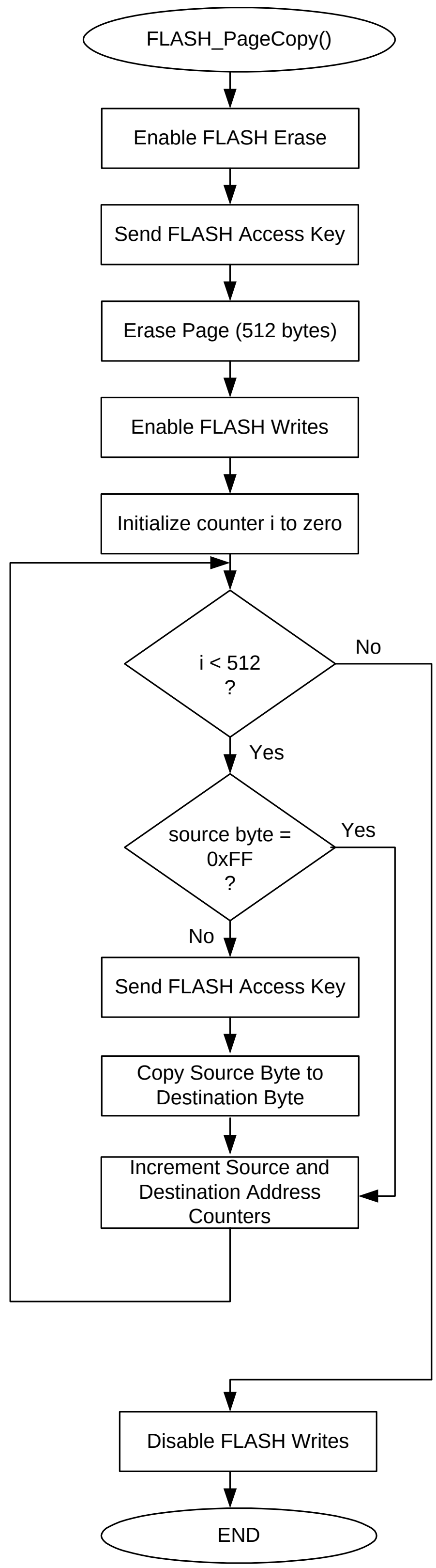


Figure 6. *Flash_PageCopy*() Flow Chart.

APPENDIX B—SOFTWARE

F350_Weigh_Scale.h

```
//-----------------------------------------------------------------------------
//
// Copyright 2004 Silicon Laboratories
//
// Filename:      F350_Weigh_Scale.h
// Target Device: C8051F350
// Created:       15 JAN 2004
// Created By:    DKC
// Tool chain:    KEIL Eval C51
//
// This is header file that is used to define all preprossor directives,
//      global variables, and prototypes.
// The user must modify this header file for their particular Bridge Transducer
//      in use before proceeding.
//
//-----------------------------------------------------------------------------
// Function Prototypes
//-----------------------------------------------------------------------------
void Config_F350(void);
void FLASH_PageCopy (unsigned char code *src, unsigned char xdata *dest)
reentrant;
void FLASH_PageUpdate (unsigned char *src, unsigned char *dest, int length)
reentrant;
void CalibrateADCforMeasurement(void);
void Calculate_One_Count(void);
int  Monitor_Weigh_Scale(unsigned char);
void Update_Display_Via_UART(void);
void Update_Display_Via_LCD(void);

//-----------------------------------------------------------------------------
// UNIONs, STRUCTUREs, and ENUMs
//-----------------------------------------------------------------------------
typedef union LONG {                              // byte-addressable LONG
  long l;
  unsigned char b[4];
} LONG;

typedef union INT {                               // byte-addressable INT
  int i;
  unsigned char b[2];
} INT;

LONG code DATA_PAGE[128]    _at_ 0x1A00;          // Reserved Space
char code SCRATCH_PAGE[512] _at_ 0x1800;          // Reserved Space

//-----------------------------------------------------------------------------
// Globale Variable Definitions
//-----------------------------------------------------------------------------
                                                  // Units;
unsigned int   Temperature           = 0;         // 0.1K
unsigned int   Weight                = 0;         // lbs
unsigned long  Tare_Count            = 0;         // Digital Output,empty scale
unsigned long  Full_Counts           = 0;         // Digital Output,full scale
unsigned long  One_Count             = 0;         // Digital Output,one unit
```

```
unsigned int   Data_Word             = 0;          // Used to Communicate
                                                   //   to HT1620 LCD Driver

LONG temp_LONG_1,temp_LONG_2;                      // Temporary Storage Var.
INT  temp_INT_1,temp_INT_2;                        // Temporary Storage Var.


//-----------------------------------------------------------------------------
// 8051F350 PARAMETERS
//-----------------------------------------------------------------------------
#define SYSCLK             3062500                 // System clock frequency
#define BAUDRATE           9600                    // Baud rate of UART in bps
#define TEMP_SENSOR_OFFSET -438                    // Temp Sensor OFFSET(uV/degC)
#define TEMP_SENSOR_GAIN   -1730                   // Temp Sensor Gain(uV/degC)
#define VREF               3300                    // ADC Volt Ref, (mV)
#define SCRATCH_PAGE       0x1800                  // FLASH page, temp storage

//-----------------------------------------------------------------------------
// Bit maskable PORT Definitions
//-----------------------------------------------------------------------------
sbit FREE1                         = P0^0;         // P0.0 : FREE
sbit FREE2                         = P0^1;         // P0.1 : FREE
sbit FREE3                         = P0^2;         // P0.2 : FREE
sbit CS                            = P1^0;         // P1.0 : Chip Select
sbit WR                            = P1^1;         // P1.1 : Write
sbit RD                            = P1^2;         // P1.2 : Read
sbit DATA                          = P1^3;         // P1.3 : DATA

// AMUX Selections; Analog Inputs
#define VSCALE                     0x08            // P0.0(+)  : Unipolar, VIN
#define TSCALE                     0xF8            // Internal : Unipolar TEMP

//-----------------------------------------------------------------------------
// Calibration/Calculation PARAMETERS
//-----------------------------------------------------------------------------
                                                   // An estimate of the
                                                   // Temperature<SLOPE>
                                                   //   in [tenth codes / K]
#define TEMP_SLOPE ((long) 16777216 / 100 * TEMP_SENSOR_GAIN / VREF)

//-----------------------------------------------------------------------------
// Directives for Weigh Scale Monitor Function
//-----------------------------------------------------------------------------
                                                   // Units
#define TEMPERATURE                0x01            // 0.1K
#define WEIGHT                     0x02            // Grams

//-----------------------------------------------------------------------------
// Directives for Weigh Scale DATA_PAGE Elements
//-----------------------------------------------------------------------------
#define Check_Byte_1               0x00            // 0x0A0A Default value
#define scale_slope                0x01            // Measurement Slope Register
#define scale_offset               0x02            // Voltage Offset Register
#define tare_count                 0x03            // Voltage Offset Register
#define full_counts                0x04            // Voltage Offset Register
#define one_count                  0x05            // Voltage Offset Register
```

F350_Weigh_Scale.c

```
//-----------------------------------------------------------------------------
//
// Copyright 2004 Silicon Laboratories
//
// Filename:      F350_Weigh_Scale.h
// Target Device: 8051F350
// Created:       15 JAN 2004
// Created By:    DKC
// Tool chain:    KEIL Eval C51
//
// This is a stand alone weith scale design. It output units in lbs via an LCD
// or a PC via the UART.
//
//-----------------------------------------------------------------------------
// Includes
//-----------------------------------------------------------------------------
#include <c8051f350.h>
#include "F350_Weigh_Scale.h"          //Weigh Scale Hearder File
#include <stdio.h>

//-----------------------------------------------------------------------------
// Support Subroutines
//-----------------------------------------------------------------------------
void Config_F350(void)
{  RSTSRC   = 0x06;                    // Enable VDD Monitor and missing clock
//-----------------------------------------------------------------------------
// PCA Configuration
//-----------------------------------------------------------------------------
   PCA0MD &= ~0x40;                    // WDTE = 0 (Disable watchdog timer)

//-----------------------------------------------------------------------------
// Port Configuration
//-----------------------------------------------------------------------------
   XBR0     = 0x01;                    // Enable UART to Pins P0.4, P0.5
   XBR1     = 0x40;                    // Enable Crossbar

   P0SKIP   = 0x00;                    // Skip No Port Pins
   P0MDOUT |= 0x10;                    // Enable UTX as push-pull output
   P0MDIN   = 0xFF;                    // Configure No Pins as Analog Inputs

//-----------------------------------------------------------------------------
// Oscilator Configuration
//-----------------------------------------------------------------------------
   OSCICN |= 0xC0;                     // Configure internal oscillator for
                                       // its default frequency

//-----------------------------------------------------------------------------
// UART0_Init
//-----------------------------------------------------------------------------
//
// Configure the UART0 using Timer1, for <BAUDRATE> and 8-N-1.
//
   SCON0 = 0x10;                       // SCON0: 8-bit variable bit rate
                                       //        level of STOP bit is ignored
                                       //        RX enabled
                                       //        ninth bits are zeros
```

```
                                    //        clear RI0 and TI0 bits
   if (SYSCLK/BAUDRATE/2/256 < 1) {
      TH1 = -(SYSCLK/BAUDRATE/2);
      CKCON |=  0x08;               // T1M = 1; SCA1:0 = xx
   } else if (SYSCLK/BAUDRATE/2/256 < 4) {
      TH1 = -(SYSCLK/BAUDRATE/2/4);
      CKCON &= ~0x0B;               // T1M = 0; SCA1:0 = 01
      CKCON |=  0x01;
   } else if (SYSCLK/BAUDRATE/2/256 < 12) {
      TH1 = -(SYSCLK/BAUDRATE/2/12);
      CKCON &= ~0x0B;               // T1M = 0; SCA1:0 = 00
   } else {
      TH1 = -(SYSCLK/BAUDRATE/2/48);
      CKCON &= ~0x0B;               // T1M = 0; SCA1:0 = 10
      CKCON |=  0x02;
   }

   TL1 = TH1;                       // init Timer1
   TMOD &= ~0xf0;                   // TMOD: timer 1 in 8-bit autoreload
   TMOD |=  0x20;
   TR1 = 1;                         // START Timer1
   TI0 = 1;                         // Indicate TX0 ready
}

//-----------------------------------------------------------------------------
// FLASH_PageUpdate
//-----------------------------------------------------------------------------
//
// This routine updates <length> characters at <dest> with those pointed to
// by <src> using a page-based read-modify-write process.
//
// It first erases <SCRATCH_PAGE> and copies the page containing the <dest>
// data to <SCRATCH_PAGE>, excluding <length> bytes starting at <dest>.
// It then copies <SCRATCH_PAGE> back to the page containing <dest>.  Finally,
// it copies <length> bytes from <src> to <dest>, completing the update
// process.
// Note: this algorithm does not take into account memory page boundaries...
//    It assumes each update is within one page.
void FLASH_PageUpdate (signed char *src, signed char *dest, int length) reentrant
{
   int i;                           // byte counter
   unsigned char xdata *pwrite;     // FLASH write pointer
   unsigned char code *pread;       // FLASH read pointer
   unsigned char code *pstop;
   unsigned char code *pstart;

   pwrite = (unsigned char xdata *) SCRATCH_PAGE;

   EA = 0;                          // disable interrupts (precautionary)
   PSCTL = 0x03;                    // MOVX writes target FLASH memory;
                                    // FLASH Erase operations enabled
   FLKEY = 0xA5;                    // FLASH key sequence #1
   FLKEY = 0xF1;                    // FLASH key sequence #2

   *pwrite = 0x00;                  // initiate erase operation

   PSCTL = 0x00;                    // Disable FLASH Writes

   // initialize <pread> to beginning of FLASH page containing <dest>
```

```
   pread = (unsigned char code *) ((unsigned int) dest & 0xFE00);

   // <pstop> points to <dest>, which is the beginning of the area to exclude
   // from the copy process.
   pstop = (unsigned char code *) dest;

   // <pstart> points to the byte right after the area to exclude from the
   // copy process.
   pstart = (unsigned char code *) (pstop + length);

   // Now we copy the page containing <dest> to SCRATCH_PAGE, excluding
   // the bytes that are to be changed.
   for (i = 0; i < 512; i++) {
      if ((pread < pstop) || (pread >= pstart)) {
         if (*pread != 0xFF) {         // exclude copying 0xff's for efficiency

            PSCTL = 0x01;              // disable FLASH erase operations;
                                       // MOVX writes target FLASH
            FLKEY = 0xA5;              // FLASH key sequence #1
            FLKEY = 0xF1;              // FLASH key sequence #2
            *pwrite = *pread;          // copy bytes
         }  PSCTL = 0x00;              // Disable FLASH Writes
      }
      pwrite++;                        // advance pointers
      pread++;
   }

   // At this point, <SCRATCH_PAGE> has a copy of the entire page containing
   // <dest>, with the exclusion of the bytes to be replaced.  We now copy
   // <SCRATCH_PAGE> back to the page containing <dest>.

   pwrite = (unsigned char xdata *) ((unsigned int) dest & 0xFE00);
   pread = (unsigned char code *) SCRATCH_PAGE;
   FLASH_PageCopy (pread, pwrite);

   // At this point, the page containing <dest> has been restored; the bytes
   // to be changed are now 0xFF's; we can proceed with copying the bytes
   // from <src> into <dest> to complete the update.

   pwrite = (unsigned char xdata *) dest;

   EA = 0;                             // disable interrupts (precautionary)

   for (i = 0; i < length; i++) {
      if (*src != 0xFF) {              // exclude writing 0xff's for
         PSCTL = 0x01;                 // MOVX writes target FLASH memory
                                       // efficiency
         FLKEY = 0xA5;                 // FLASH key sequence #1
         FLKEY = 0xF1;                 // FLASH key sequence #2
         *pwrite++ = *src++;           // copy bytes
      }  PSCTL = 0x00;                 // Disable FLASH Writes
   }
}

//-----------------------------------------------------------------------------
// FLASH_PageCopy
//-----------------------------------------------------------------------------
//
// This routine copies the FLASH page starting at <src> to <dest>.  It erases
```

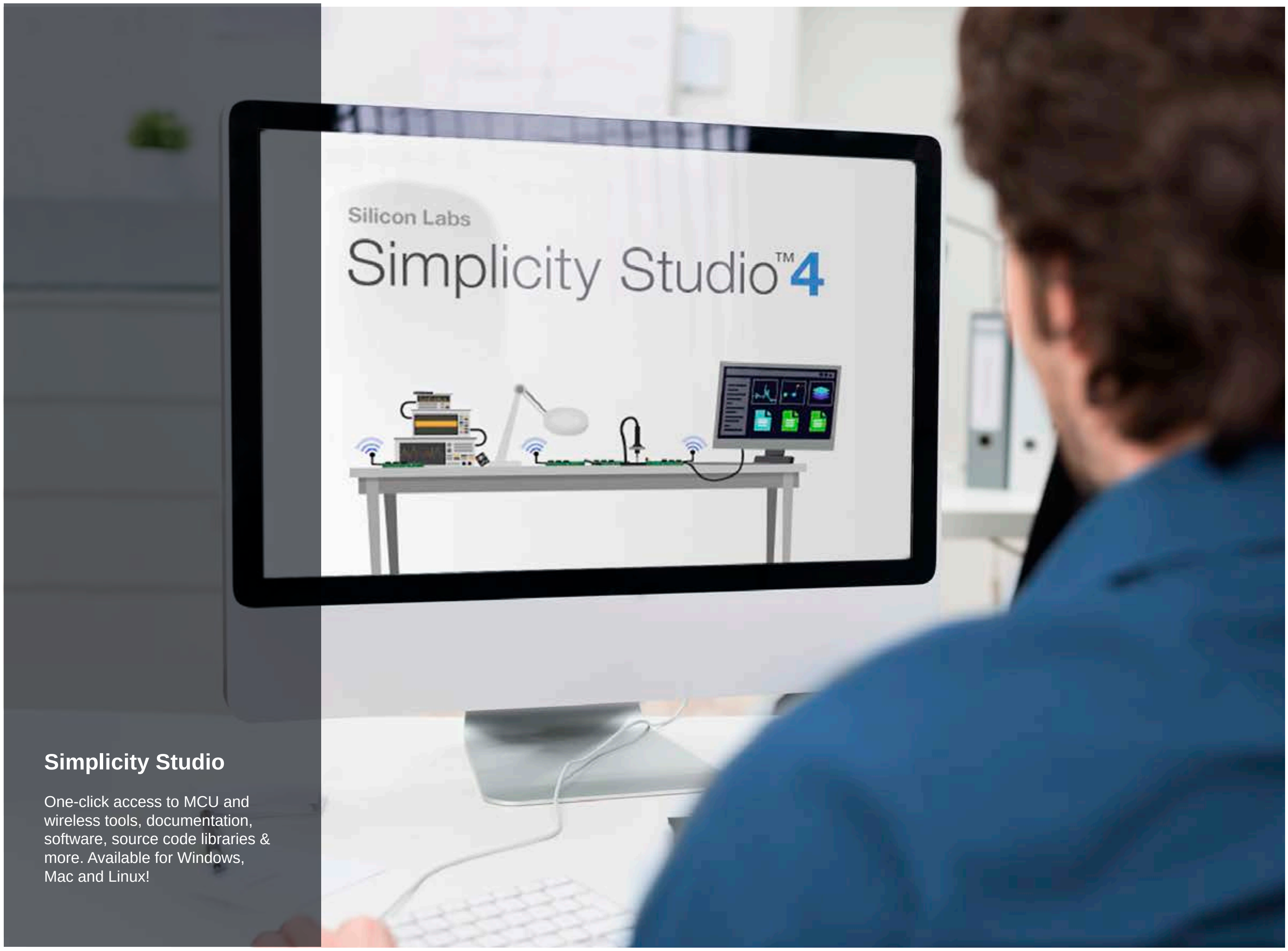
```
// the <dest> page before the copy process begins.
//
void FLASH_PageCopy (unsigned char code *src, unsigned char xdata *dest) reentrant
{
   int i;                             // byte counter

   EA = 0;                            // disable interrupts (precautionary)
   PSCTL = 0x03;                      // MOVX writes target FLASH memory;
                                      // FLASH erase operations enabled

   FLKEY = 0xA5;                      // FLASH key sequence #1
   FLKEY = 0xF1;                      // FLASH key sequence #2
   *dest = 0;                         // initiate erasure of <dest> FLASH
                                      // page
   PSCTL = 0x00;                      // Disable FLASH Writes

   for (i = 0; i < 512; i++) {
      if (*src != 0xFF) {             // exclude writing 0xff's for efficiency

         PSCTL = 0x01;                // disable FLASH erase operations;
                                      // MOVX writes target FLASH memory
         FLKEY = 0xA5;                // FLASH key sequence #1
         FLKEY = 0xF1;                // FLASH key sequence #2
         *dest = *src;                // copy bytes
      }  PSCTL = 0x00;                // Disable FLASH Writes
      dest++;                         // advance pointers
      src++;
   }
}

//-----------------------------------------------------------------------------
// CalibrateADCforMeasurement
//-----------------------------------------------------------------------------
// This routine assumes memory block 0x1A00 is erased
// Then this function calibrates the voltage channel and stores the calibration
// coefficients in the parameters xxx_slope and xxx_offset.
// This calibration routine uses the F350's internal calibration functions.
//
void CalibrateADCforMeasurement(void)
{  unsigned char xdata *idata pwrite;  // FLASH write pointer

   EA = 0;                            // Disable All Interrupts

                                      // Perform Self Calibration
   ADC0MD   = 0x84;                   // Enable ADC; Internal Offset Cal.
   while(!AD0CALC);                   // Wait for calibration to complete

   ADC0MD   = 0x85;                   // Enable ADC; Internal Gain Cal.
   while(!AD0CALC);                   // Wait for Calibration to complete

                                      // Memory's been erased at 0x1A00
                                      // Store Gain Coefficients to FLASH
   temp_LONG_1.l = 1234;              // Prepare storage parameter
   pwrite = (char xdata *)&(DATA_PAGE[scale_slope].l);
   FLASH_PageUpdate ((unsigned char *)&temp_LONG_1.b[0], pwrite, 4);

                                      // Memory's been erased at 0x1A00
                                      // Store the Offset Coeffs to FLASH
   temp_LONG_1.l = 1234;              // Prepare storage parameter
```

```
   pwrite = (char xdata *)&(DATA_PAGE[scale_offset].l);
   FLASH_PageUpdate ((signed char *)&temp_LONG_1.b[0], pwrite, 4);
}

//-----------------------------------------------------------------------------
// Calculate_One_Count
//-----------------------------------------------------------------------------
// This routine calculates the tare weight of the scale in digital counts
// Then this function assumes the user places 100 equal units to be weighed
// on scale.  From these two measurements the count of one unit is computed.
// From this, the number of units on the scales can be determined in future
// measurements
void Calculate_One_Count(void)
{  unsigned char xdata *idata pwrite;// FLASH write pointer
   char i;

   ADC0CN   = 0x00;                     // Unipolar; Gain = 1
   ADC0DECH = 0x04;                     // Set Output word rate at for ~23 Hz
   ADC0DECL = 0x00;                     // Set Output word rate at for ~23 Hz
   ADC0MUX  = VSCALE;                   // Select appropriate input for AMUX

   Tare_Count = 0;                      // Initialize to zero
   for(i=5;i;--i)                       // Average next 5 conversions
   {
      temp_LONG_1.l = 0;
      AD0INT = 0;                       // Clear end-of-conversion indicator
      ADC0MD   = 0x82;                  // Enable ADC; Single conversions
      while(!AD0INT);                   // Wait for conversion to complete
      temp_LONG_1.l = ADC0H;
      temp_LONG_1.l = temp_LONG_1.l <<16;
     temp_LONG_1.l += ADC0M <<8;
      temp_LONG_1.l += ADC0L ;
      Tare_Count += temp_LONG_1.l;
   }
   Tare_Count = Tare_Count/5;           // Store the Tare Digital Count Value

   Full_Counts = 0;                     // Initialize to zero
   for(i=5;i;--i)                       // Average next 5 conversions
   {
      temp_LONG_1.l = 0;
      AD0INT = 0;                       // Clear end-of-conversion indicator
      ADC0MD   = 0x82;                  // Enable ADC; Single conversions
      while(!AD0INT);                   // Wait for conversion to complete
      temp_LONG_1.l = ADC0H;
      temp_LONG_1.l = temp_LONG_1.l <<16;
     temp_LONG_1.l += ADC0M <<8;
      temp_LONG_1.l += ADC0L ;
      Full_Counts += temp_LONG_1.l;
   }

   Full_Counts = Full_Counts/ 5;        // Store the Calibration Count Value;
                                        //     unually 100 units

   One_Count = Full_Counts - Tare_Count;
   One_Count = One_Count/100;           // Divide by 100 assumes there are
                                        //     100 cal units on scale

   ADC0MD   = 0x00;                     // Turn off ADC Module
```

```
                                  // Memory's already been erased at 0x1A00
                                  // Store Gain Coefficients to FLASH
   temp_LONG_1.l = 1234;          // Prepare for FLASH Write
   pwrite = (char xdata *)&(DATA_PAGE[tare_count].l);
   FLASH_PageUpdate ((signed char *)&temp_LONG_1.b[0], pwrite, 4);

   temp_LONG_1.l = 1234;          // Prepare for FLASH Write
   pwrite = (char xdata *)&(DATA_PAGE[full_counts].l);
   FLASH_PageUpdate ((signed char *)&temp_LONG_1.b[0], pwrite, 4);

   temp_LONG_1.l = 1234;          // Prepare for FLASH Write
   pwrite = (char xdata *)&(DATA_PAGE[one_count].l);
   FLASH_PageUpdate ((signed char *)&temp_LONG_1.b[0], pwrite, 4);
}


//-----------------------------------------------------------------------------
// Monitor Weigh_Scale
//-----------------------------------------------------------------------------
// This routine configures the ADC's AMUX, acquires conversions and returns
// appropriate parameter (weight or temperature) via variable result.
// Weight is returned in Unit counts. Example 100 pennies is 100.
// Temperature is returned in degree Kelvin.  Ex 273 Kelvin returns a value 273
//
int Monitor_Weigh_Scale(unsigned char value)
{  char i;
   unsigned long av =0,delay_count=0;
   long signed result;

   ADC0DECH = 0x04;               // Set Output word rate at for 23 Hz
   ADC0DECL = 0x00;               // Set Output word rate at for 23 Hz

   switch (value)
   {
      case TEMPERATURE:
         ADC0CN  = 0x00;          // Unipolar; Gain = 1
         ADC0MUX = TSCALE;        // Select appropriate input for AMUX
         break;

      case WEIGHT:
         ADC0CN   = 0x00;         // Unipolar; Gain = 1
         ADC0MUX  = VSCALE;       // Select appropriate input for AMUX
         break;
   }
   //Compute average of next 5 A/D conversions
   av = 0;                        // Initialize to Zero
   for(i=5;i;--i)                 // Average next 5 conversions
   {
      temp_LONG_1.l = 0;
      AD0INT = 0;                 // Clear end-of-conversion indicator
      ADC0MD   = 0x82;            // Enable ADC; Single conversions
      while(!AD0INT);             // Wait for conversion to complete
      temp_LONG_1.l = ADC0H;
      temp_LONG_1.l = temp_LONG_1.l <<16;
     temp_LONG_1.l += ADC0M <<8;
      temp_LONG_1.l += ADC0L ;
      av += temp_LONG_1.l;
   }
```

```
   ADC0MD   = 0x00;                     // Turn off ADC Module

   av = av/5;                           // Compute the average

   switch (value)
   {
      case TEMPERATURE:
         result =  (long) av /TEMP_SLOPE*1000; // Account for Temp. Slope
         result -= 100*TEMP_SENSOR_OFFSET;     // Account for Temp. Offset
         result = result + 27315;       // Convert to Degrees Kelvin
         break;

      case WEIGHT:
        result = av - Tare_Count;
        result = result/One_Count;
        break;
   }
   return (signed int) result;
}

void Update_Display_Via_UART(void)
{
   // Send Information to Hyper Terminal at 9600 Baud,8,n,1
   //printf ("Temperature = %d hundredths degrees K\n", Temperature);
   printf ("Counts = %d  units\n", (int)Weight);
}
//-----------------------------------------------------------------------------
// Update_Display_Via_LCD
//-----------------------------------------------------------------------------
// This function provides starter algorithms for the HT1620,(Holtek) LCD driver.
// It provides algorithms to write to te HT1620 and Read from the HT1620.
// Specific algorithms to activate and deactivate the elements on the LCD that
// correspond to specific letters and numerals will need to be develop.
//
void Update_Display_Via_LCD(void)
{  unsigned char BIT_count; // counter for SPI transaction

   // Here is an example of a write command to the HT1620
   Data_Word = 0x1400;                  // Prepare information for writing to LCD
   Data_Word = Data_Word << 3;          // Prepare information for writing to LCD
   CS   = 0;                            // Select Serial Port
   for (BIT_count = 13; BIT_count > 0;BIT_count--)// 13 bits
   {
      DATA = Data_Word & 0x8000;        // put current outgoing bit on DATA
      Data_Word = Data_Word <<1;        // shift next bit into MSB

      WR = 0x01;                        //set sck high

      WR = 0x00;                        // set sck low
   }

   CS   = 1;                            // Deselect LCD

   // Here is an example of a read command to the HT1620

   Data_Word = 0x1800;                  // Prepare information for writing to LCD
   Data_Word = Data_Word << 3;          // Prepare information for writing to LCD
   CS   = 0;                            // Select LCD
   for (BIT_count = 9; BIT_count > 0;BIT_count--)// 9 bits
```

```
   {
      DATA = Data_Word & 0x8000;          // put current outgoing bit on DATA
      Data_Word = Data_Word <<1;          // shift next bit into MSB

      WR = 0x01;                          // set sck high

      Data_Word |= DATA;                  // capture current bit on DATA

      WR = 0x00;                          // set sck low
   }
   for (BIT_count = 4; BIT_count > 0;BIT_count--)// 4 bits
   {
      DATA = Data_Word & 0x8000;          // put current outgoing bit on DATA
      Data_Word = Data_Word <<1;          // shift next bit into MSB

      RD = 0x01;                          // set sck high

      Data_Word |= DATA;                  // capture current bit on DATA

      RD = 0x00;                          // set sck low
   }

   CS   = 1;                              // Deselect LCD
}

//-----------------------------------------------------------------------------
// Main Function
//-----------------------------------------------------------------------------
// - Main calls all the functions necessary to configure the C8051F350.
//     It also calls routines to calibrate the electronic scale.  Once setup
//      and calibration are complete, Main() determines the Unit count on
//      the scale and outputs via the UART.
//
//      System Calibration is only performed the first time through the cycle.
//
void main(void)
{
   Config_F350();                         // Config F350

   CalibrateADCforMeasurement();          // Calibrate ADC

   Calculate_One_Count();                 // Calculate the size of one count

   // Update Temperature  and Bridge Monitoring Algorithms
   while(1)
   {                                      // Once pressed get temperature and weight
     Temperature = Monitor_Weigh_Scale (TEMPERATURE);// Get Latest Temperature
                                          // Acquire Pack's Voltages
      Weight = Monitor_Weigh_Scale (WEIGHT);

                                          // Update PC Via UART
      Update_Display_Via_UART();
                                          // Update PC Via LCD
      //Update_Display_Via_LCD();
   }
}
// END of File
```



IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

http://www.silabs.com