# C Example Listing

```
#include </mosaic/allqed.h>
#include "library.c"

//------------------------------------------------------------
//                             Example 1
//------------------------------------------------------------

// This first sample routine demonstrates how to use Init_AD24,
// Use_Onboard_Ref, Start_Conversion, and AD24_Sample.  This routine
// takes 1 differential 24-bit bipolar sample at 10 Hz with a gain
// of 1 and the burnout options turned off and prints it out.  If an
// invalid option is specified or if a timeout occurs, an error flag
// is returned.  Error flags are:
//  INVALID_GAIN = 1, INVALID_FREQ = 2, INVALID_CAL = 3,
//  INVALID_CHANNEL = 4, INVALID_FSYNC = 5, INVALID_BO = 6,
//  INVALID_SIZE = 7, INVALID_POLARITY = 8, TIMEOUT_ERROR = 9
int Sample_Routine ( void )
{
  int flag = 0;
  ulong sample;
  float result;

  if( Init_AD24( MODULE0) )          // 24/7 Data Acquisition Module
  {                                  // is first module on the stack
    Use_Onboard_Ref();               // Use on-board reference

    // Start_Conversion must be called before getting a sample!
    flag = Start_Conversion(  SELF_CAL,
                              1920,       // 10 Hz -> 19200/10=1920
                              GAIN_1,
                              BIPOLAR,
                              WORD_24BIT, // 24 bit resolution
                              BO_OFF,
                              CH_0_1 );
    if( flag == -1 )
    {
      sample = AD24_Sample();              // Get sample
      if( sample != TIMEOUT_ERROR )        // If no timeout occurred
      {
        // Divide by 256 to remove timeout flag & convert to 24 bits
        // Subtract 8388608 (2^23) to remove the bipolar offset
        // Multiply by (5.00+/-0.01)/(2^24) to convert to volts
        // 5.00+/-0.01 is obtained by multiplying the reference
        // voltage by 2; i.e. (2.500+/-0.005)*2
        // Divide result by the gain for gains greater than 1
        // for example: for a gain of 8 divide result by 8
        // DO NOT DIVIDE BY GAIN_8! GAIN_8 != 8!
        result=(float)((sample/256)-8388608)*(0.0000002980);
        printf("\nResult = %2.4f \n",result);
      }
    }
  }
  return(flag);
}
```

```
//------------------------------------------------------------------
//                             Example 2
//------------------------------------------------------------------

// This second sample routine demonstrates how to use AD24_Multiple.
// This routine takes 10 samples from a single channel at 10 hz and
// stores the samples to the pad area.  Returns -1 if successful, 0
// if an invalid module number was passed to Init_AD24, returns 1-9
// if an invalid calibration coefficient was passed to
// Start_Conversion.
int Sample_Routine2 ( void )
{
  int flag = 0;
  EXTENDED_ADDR pad_buffer;        // 88 byte buffer in common RAM

  pad_buffer.sixteen_bit.page16 = 0x00;
  pad_buffer.sixteen_bit.addr16 = 0x8b24;


  if( Init_AD24( MODULE0 ) )       // 24/7 Data Acquisition Module
                                   // is first module on the stack
  {
    Use_Onboard_Ref();             // Use on-board reference

    // Start_Conversion must be called before getting a sample!
    flag = Start_Conversion(  SELF_CAL,
                              1920,       // 10 Hz -> 19200/10=1920
                              GAIN_1,
                              BIPOLAR,
                              WORD_24BIT, // 24 bit resolution
                              BO_OFF,
                              CH_0_1 );
    if( flag == -1 )          // Start_Conversion was successful
    {
      // Get 10 samples, store to RAM
      // The values raw conversion values can be shown by typing:
      // pad 40 dump
      // The values can be read from memory by using: FetchLong()
      flag = AD24_Multiple( 10, pad_buffer.addr32 );
    }
  }
  return(flag);
}

//------------------------------------------------------------------
//                             Example 3
//------------------------------------------------------------------

// The final example shows how to read 4 different channels at a
// fixed sample rate without performing a calibration before each
// sample and without using interrupts.  This example performs a
// Self Calibration on each channel, reads the calibration
// coefficients using Read_FS_Cal() and Read_Zero_Cal(), stores the
// calibration coefficients into a structure, loads the 24-Bit A/D
// with the stored calibration coefficients using
// Start_Conv_With_Values(), periodically samples each channel using
// AD24_Sample_NP(), and finally stores the samples into an array.

#define CH0       0               //  Constants for channels 0 - 3
#define CH1       1
#define CH2       2
#define CH3       3
#define SAMPLE_FREQ 320           // Constant corresponding to 60 hz
                                  // 19200 / 60 = 320 [See Table 5]
#define NUM_SAMPLES 40            // Total number of samples:
                                  // 10 samples for 4 channels
#define NUM_CHANNELS 4            // Num channels we are sampling

// Declare an array for samples & allocate memory for the samples
// from 4 sensors; each sample is 4 bytes.
ulong ad24_data[NUM_SAMPLES/NUM_CHANNELS][NUM_CHANNELS];

typedef struct                    // Config options for each channel
{
  ulong ad_zero_cal;              // 24-bit zero sacle cal val
  ulong ad_fs_cal;                // 24-bit full scale cal val
  int  ad_freq_int;               // Frequency Integer 19 - 4000.
  char ad_gain;                   // Gain 1 to 128.
  char ad_polarity;               // Bipolar or Unipolar mode.
  char ad_res;                    // Resolution: 16-bit or 24-bit.
  char ad_bo;                     // Burn out current on/off
  char ad_fsync;                  // Sync on/off.
  char ad_ch;                     // Channel.
} AD_CHANNEL;

typedef struct                    // Global structure.
{
  AD_CHANNEL ch0;
  AD_CHANNEL ch1;
  AD_CHANNEL ch2;
  AD_CHANNEL ch3;
  char   current_channel;         // Current channel being used.
  int    index;                   // Index into data array
} AD_INFO;

AD_INFO ad24_struct;              // Declare a global instance of
the
                                  // structure.

// Perform a Full Self Calibration on channel 0-1 for bipolar, unity
// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 0 of my_struct with calibration coefficients
// and settings.
int Init_CH0 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_0_1 );

  if( flag == -1 )
  {
    ad24_struct.ch0.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch0.ad_gain     = GAIN_1;
    ad24_struct.ch0.ad_polarity = BIPOLAR;
    ad24_struct.ch0.ad_res      = WORD_24BIT;
    ad24_struct.ch0.ad_bo       = BO_OFF;
    ad24_struct.ch0.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch0.ad_ch       = CH_0_1;
    ad24_struct.ch0.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch0.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 2-3 for bipolar, unity
// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 1 of my_struct with calibration coefficients
// and settings.
int Init_CH1 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_2_3 );

  if( flag == -1 )
  {
    ad24_struct.ch1.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch1.ad_gain     = GAIN_1;
    ad24_struct.ch1.ad_polarity = BIPOLAR;
    ad24_struct.ch1.ad_res      = WORD_24BIT;
    ad24_struct.ch1.ad_bo       = BO_OFF;
    ad24_struct.ch1.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch1.ad_ch       = CH_2_3;
    ad24_struct.ch1.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch1.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 4-5 for bipolar, unity
// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 2 of my_struct with calibration coefficients
// and settings.
int Init_CH2 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_4_5 );

  if( flag == -1 )
  {
    ad24_struct.ch2.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch2.ad_gain     = GAIN_1;
    ad24_struct.ch2.ad_polarity = BIPOLAR;
    ad24_struct.ch2.ad_res      = WORD_24BIT;
    ad24_struct.ch2.ad_bo       = BO_OFF;
    ad24_struct.ch2.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch2.ad_ch       = CH_4_5;
    ad24_struct.ch2.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch2.ad_fs_cal   = Read_FS_Cal();
  }
  return(flag);
}

// Perform a Full Self Calibration on channel 6-7 for bipolar, unity
// gain, 60 Hz operation and get calibration coefficients.
// Initialize channel 3 of my_struct with calibration coefficients
// and settings.
int Init_CH3 ( void )
{
  int flag;

  flag = Start_Conversion( SELF_CAL, SAMPLE_FREQ, GAIN_1, BIPOLAR,
                           WORD_24BIT, BO_OFF, CH_6_7 );

  if( flag == -1 )
  {
    ad24_struct.ch3.ad_freq_int = SAMPLE_FREQ;
    ad24_struct.ch3.ad_gain     = GAIN_1;
    ad24_struct.ch3.ad_polarity = BIPOLAR;
    ad24_struct.ch3.ad_res      = WORD_24BIT;
    ad24_struct.ch3.ad_bo       = BO_OFF;
    ad24_struct.ch3.ad_fsync    = FSYNC_OFF;
    ad24_struct.ch3.ad_ch       = CH_6_7;
    ad24_struct.ch3.ad_zero_cal = Read_Zero_Cal();
    ad24_struct.ch3.ad_fs_cal   = Read_FS_Cal();
  }
    return(flag);
}

// This routine loads the 24-Bit A/D with the next set of
// calibration coefficients without performing a calibration.
void Load_Coefficients( int current_ch )
{

  switch( current_ch )
  {
    case CH0: Start_Conv_With_Values( ad24_struct.ch0.ad_fs_cal,
              ad24_struct.ch0.ad_zero_cal,
              ad24_struct.ch0.ad_freq_int,
              ad24_struct.ch0.ad_gain, ad24_struct.ch0.ad_polarity,
              ad24_struct.ch0.ad_res, ad24_struct.ch0.ad_bo,
              ad24_struct.ch0.ad_fsync, ad24_struct.ch0.ad_ch );
              break;
    case CH1: Start_Conv_With_Values( ad24_struct.ch1.ad_fs_cal,
              ad24_struct.ch1.ad_zero_cal,
              ad24_struct.ch1.ad_freq_int,
              ad24_struct.ch1.ad_gain, ad24_struct.ch1.ad_polarity,
              ad24_struct.ch1.ad_res, ad24_struct.ch1.ad_bo,
              ad24_struct.ch1.ad_fsync, ad24_struct.ch1.ad_ch );
              break;
    case CH2: Start_Conv_With_Values( ad24_struct.ch2.ad_fs_cal,
              ad24_struct.ch2.ad_zero_cal,
               ad24_struct.ch2.ad_freq_int,
               ad24_struct.ch2.ad_gain, ad24_struct.ch2.ad_polarity,
               ad24_struct.ch2.ad_res, ad24_struct.ch2.ad_bo,
               ad24_struct.ch2.ad_fsync, ad24_struct.ch2.ad_ch );
               break;
    case CH3: Start_Conv_With_Values( ad24_struct.ch3.ad_fs_cal,
               ad24_struct.ch3.ad_zero_cal,
               ad24_struct.ch3.ad_freq_int,
               ad24_struct.ch3.ad_gain, ad24_struct.ch3.ad_polarity,
               ad24_struct.ch3.ad_res, ad24_struct.ch3.ad_bo,
               ad24_struct.ch3.ad_fsync, ad24_struct.ch3.ad_ch );
               break;
  }
}

// This routine takes one sample, stores it to an array, then starts
// a conversion for the next channel.
void Get_Sample ( void )
{
// Get sample from the 24-Bit A/D, store to array
ad24_data[ad24_struct.index][ad24_struct.current_channel]=AD24_Sample_NP();

  // Increment channel number, did we sample all the channels yet?
  if( ad24_struct.current_channel++ >= NUM_CHANNELS - 1 )
  {
    // Init channel number to 0, we finished sampling all channels.
    ad24_struct.current_channel = 0;
    // Set varible to store next group of data
    ad24_struct.index++;
  }

  // Load coefficients for the next channel
  Load_Coefficients ( ad24_struct.current_channel );
}

// This routine repeatedly calls Get_Sample() every timeslice_ticks
// * 5ms.  Be sure other tasks do not take longer than
// timeslice_ticks * 5ms.
void Execute_So_Often ( uint num_times, ulong timeslice_ticks )
{
  ulong target_time;
  int i;

  for(i=0;i<num_times;i++)
  {
    target_time = TIMESLICE_COUNT + timeslice_ticks;
    Get_Sample();
    do
    {
      Pause();
    } while (TIMESLICE_COUNT<target_time);
  }
}

// This routine takes 10 samples from 4 sensors at 60 Hz.  All of
// the settings for each channel are stored in a global structure.
// All channels must have the same sampling rate!
```

```
int Sample_Routine3 ( void )
{
  int flag;

  flag = Init_AD24( MODULE0 );// 24/7 Data Acquisition Module is
                             // the first module on the stack

  Use_Onboard_Ref();          // Use on-board reference
  ad24_struct.current_channel = CH0;// Set ch0 as current channel
  ad24_struct.index = 0;      // Init array index number

  // Init structure;  Be sure to call Init_CH0 last since it is the
  // first channel sampled.
  Init_CH3(); Init_CH2(); Init_CH1(); Init_CH0();

  // Get 1 sample every 60 ms.  60 ms is the fastest we can call
  // Get_Sample() because the sample rate is 60Hz and the 24-Bit A/D
  // takes 3 clock cycles to obtain a sample when using
  // Start_Conv_With_Values().  This alone is 3/60 or 50 ms.  If a
  // full Self-Calibration was performed before each conversion, the
  // fastest rate you could sample one channel would be 10/60 or 167
  // ms.  This would amount to 666 ms for 4 channels or 1.5 Hz per
  // channel.
  Execute_So_Often( NUM_SAMPLES, (ulong)12 ); // 12*5ms = 60ms
  return(flag);

}
```