Forth Example Listing

```
\ ----------------------------------------------------------------------
\                          Example 1
\ ----------------------------------------------------------------------

hex

FF constant LSB_MASK                \ Constant to isolate 8 least
                                    \ significant bits of an integer
decimal

\ 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: INIT_ERROR = 0,
\           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
: Sample_Routine ( -- flag | flag = success )
false locals{ &flag }

  MODULE0 Init_AD24                 \ 24/7 Data Acquisition Module is
                                    \ the first module on the stack
  if
    Use_Onboard_Ref                 \ Use on-board reference
    SELF_CAL
    1920                            \ 10 Hz -> 19200 / 10 = 1920
    GAIN_1
    BIPOLAR
    WORD_24BIT                      \ 24 bit resolution
    BO_OFF
    CH_2_3
    Start_Conversion                \ This must be called before
                                    \ getting a sample
    to &flag
    &flag -1 =
    if
      AD24_Sample                   \ Get sample
      over LSB_MASK AND             \ Just look at 8 LSB of the sample
      TIMEOUT_ERROR = not           \ Does it equal to timeout flag?
      if                            \ If no timeout occurred,
        -8 DSCALE                   \ Shift sample to get 24 bits
                                    \ Convert to volts
        din 8388608 d- dflot        \ Subtract off 0x800000 because
                                    \ of the bipolar conversion
        0.0000002980 f*             \ Multiply by (5.00+/-0.01)/(2^24)

       \ 5.00+/-0.01 is obtained by multiplying the reference
       \ voltage by 2; i.e. (2.500+/-0.005)*2

        \ 1.0 f/                    \ Divide by the gain if necessary
                                    \ DO NOT DIVIDE BY GAIN_1!
```

```
                                    \ GAIN_1 != 1, GAIN_2 != 2, ...
        f.                          \ Print out result
        true to &flag               \ Return true
      endif
    endif
  endif
  &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 TRUE if successful,
\ FALSE if a timeout occurred in AD24_Multiple, an invalid calibration
\ coefficient was passed to Start_Conversion, or an invalid module
\ number was passed to Init_AD24.

: Sample_Routine2 ( -- flag )

  MODULE0 Init_AD24                 \ 24/7 Data Acquisition Module is
                                    \ the first module on the stack
  if
    Use_Onboard_Ref                 \ Use on-board reference
    SELF_CAL
    1920                            \ 10 Hz -> 19200 / 10 = 1920
    GAIN_1
    BIPOLAR
    WORD_24BIT                      \ 24 bit resolution
    BO_OFF
    CH_0_1
    Start_Conversion                \ This must be called before
                                    \ getting a sample
    if                              \ If a conversion was
                                    \ successfully started,
      10 pad AD24_Multiple          \ Get 10 samples, store to pad
    else
      FALSE                         \ Invalid calibration coefficient
    endif
  else
    FALSE                           \ Invalid module number
  endif
;

\ ------------------------------------------------------------------
\                         Example 3
\ ------------------------------------------------------------------

\ The final sample routine uses the timeslice clock to obtain 10
\ samples from 4 different sensors at 60 Hz without using interrupts.
\ This routine uses a global structure to contain the settings and
\ calibration coefficients of each channel.

0   constant CH0                    \ Constants for channels 0 - 3
```

```
1   constant CH1
2   constant CH2
3   constant CH3
320 constant SAMPLE_FREQ              \ freq int corresponding to 60 hz
                                      \ 19200 / 60 = 320 [See Table 5]
40  constant NUM_SAMPLES              \ Total number of samples:
                                      \ 10 samples for 4 channels
4   constant NUM_CHANNELS             \ Num channels we are sampling

array: my_data                        \ Declare an array for samples

structure.begin: ad_channel           \ Config options for each channel
  double-> +ad_zero_cal               \ 24-bit zero scale cal val
  double-> +ad_fs_cal                 \ 24-bit full scale cal val
  int->  +ad_freq_int                 \ Frequency Integer 19 - 4000.
  byte-> +ad_gain                     \ Gain 1 to 128.
  byte-> +ad_polarity                 \ Bipolar or Unipolar mode.
  byte-> +ad_res                      \ Resolution: 16-bit or 24-bit.
  byte-> +ad_bo                       \ Burn out current on/off
  byte-> +ad_fsync                    \ Sync on/off.
  byte-> +ad_ch                       \ Channel.
structure.end

structure.begin: ad_info              \ Global structure.
  ad_channel struct-> +ch0
  ad_channel struct-> +ch1
  ad_channel struct-> +ch2
  ad_channel struct-> +ch3
  byte->  +current_channel            \ Current channel being used.
  int->   +index                      \ Index into data array
structure.end

ad_info v.instance: my_struct         \ Declare a global instance of
                                      \ the structure in variable area.

: Init_CH0 ( -- flag )
  \ 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.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_0_1
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch0 +ad_freq_int  !
    GAIN_1        my_struct +ch0 +ad_gain     c!
    BIPOLAR       my_struct +ch0 +ad_polarity c!
    WORD_24BIT    my_struct +ch0 +ad_res      c!
    BO_OFF        my_struct +ch0 +ad_bo       c!
    FSYNC_OFF     my_struct +ch0 +ad_fsync    c!
    CH_0_1        my_struct +ch0 +ad_ch       c!
    Read_Zero_Cal my_struct +ch0 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch0 +ad_fs_cal   2!
    true
  else
    false                         \ Invalid calibration coefficients CH0
  endif
```

```
;

: Init_CH1 ( -- flag )
  \ Perform a Full Self Calibration on channel 2-3 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_2_3
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch1 +ad_freq_int  !
    GAIN_1        my_struct +ch1 +ad_gain     c!
    BIPOLAR       my_struct +ch1 +ad_polarity c!
    WORD_24BIT    my_struct +ch1 +ad_res      c!
    BO_OFF        my_struct +ch1 +ad_bo       c!
    FSYNC_OFF     my_struct +ch1 +ad_fsync    c!
    CH_2_3        my_struct +ch1 +ad_ch       c!
    Read_Zero_Cal my_struct +ch1 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch1 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH1
  endif
;

: Init_CH2 ( -- flag )
  \ Perform a Full Self Calibration on channel 4-5 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_4_5
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch2 +ad_freq_int  !
    GAIN_1        my_struct +ch2 +ad_gain     c!
    BIPOLAR       my_struct +ch2 +ad_polarity c!
    WORD_24BIT    my_struct +ch2 +ad_res      c!
    BO_OFF        my_struct +ch2 +ad_bo       c!
    FSYNC_OFF     my_struct +ch2 +ad_fsync    c!
    CH_4_5        my_struct +ch2 +ad_ch       c!
    Read_Zero_Cal my_struct +ch2 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch2 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH2
  endif
;

: Init_CH3 ( -- flag )
  \ Perform a Full Self Calibration on channel 6-7 for bipolar, unity
  \ gain, 60 Hz operation and get calibration coefficients.  Initialize
  \ channel 0 of my_struct with calibration coefficients and settings.
  SELF_CAL 320 GAIN_1 BIPOLAR WORD_24BIT BO_OFF CH_6_7
  Start_Conversion
  -1 =
  if
    SAMPLE_FREQ          my_struct +ch3 +ad_freq_int  !
```

```
    GAIN_1        my_struct +ch3 +ad_gain     c!
    BIPOLAR       my_struct +ch3 +ad_polarity c!
    WORD_24BIT    my_struct +ch3 +ad_res      c!
    BO_OFF        my_struct +ch3 +ad_bo       c!
    FSYNC_OFF     my_struct +ch3 +ad_fsync    c!
    CH_6_7        my_struct +ch3 +ad_ch       c!
    Read_Zero_Cal my_struct +ch3 +ad_zero_cal 2!
    Read_FS_Cal   my_struct +ch3 +ad_fs_cal   2!
    true
  else
    false                        \ Invalid calibration coefficients CH3
  endif
;

: Do_So_Often (word_xcfa \ ud -- | ud is in ticks of timeslice clock)

\ This word calls another routine periodically, with a fixed time
\ interval between calls of ud ticks of the timeslicer clock.  The
\ routine is designated by word.xcfa and it should return only a flag
\ on the stack. If the flag is true it will continue to be repeatedly
\ executed; as soon as it returns with a false flag this routine stops
\ calling it and returns immediately.  The word.xcfa is called at times
\ 0, ud, 2*ud, 3*ud, etc.. measured in units of timeslicer clock ticks.

\ If the execution time of word.xcfa is greater than ud ticks of the
\ timeslicer then it is just repeatedly called as rapidly as possible.
\ With a 5 msec timeslicer period the interval between calls can be up
\ to 248 days with a resolution of 5 msec.  Because we depend on the
\ timeslicer clock that clock should not be stopped or reset while this
\ routine is running.  To prevent unnoticed rollover if this routine is
\ interrupted by another task, the other task should not take longer
\ than 248 days; that is, control must return to this routine at least
\ once every 248 days.  Also word.xcfa should not take longer than 248
\ days to execute either.  That should generally not be a problem.

\ If another task has control when ud ticks are done and it is time to
\ call word.xcfa then the call to word.xcfa will be delayed until this
\ routine regains control.  However, as long as the other routine and
\ the word.xcfa routine together don't take longer than ud then all
\ subsequent timing will still occur at integer multiples of ud; there
\ is no cumulative timing error.

\ There is a PAUSE which may be removed if you don't want any other
\ tasks to have a chance at machine time.

locals{ d&time_interval x&word_xcfa | d&target_time d&start_time
        d&elapsed_time }
  timeslice.count 2@ to d&start_time \ get the start time
  begin
    d&time_interval to d&target_time
    x&word_xcfa execute              \ execute the user's word
  while           \ we stop repetitively calling the user's word
                  \ when it returns with a false flag
                  \ D&Target.Time and D&Elapsed.Time are measured from
    D&Start.Time
    begin
```

```
      pause
      timeslice.count 2@ 2dup
      d&start_time d- to d&elapsed_time to d&start_time
      d&elapsed_time d&target_time
      du<
    while         \ We readjust the start and target times to maximize
                  \ the time available to other tasks before we
                  \ experience a rollover.  This way the rollover
                  \ horizon is always pushed out to the maximum count.
      d&target_time d&elapsed_time d- to d&target_time
    repeat
    d&start_time d&target_time d+ d&elapsed_time d- to d&start_time
  repeat
;

\ This routine takes one sample, stores it to an array, then starts a
\ conversion for the next channel.
: Get_Sample ( -- flag | done? )
my_struct +current_channel c@          \ Get current channel
my_struct +index           @           \ Get current index
locals{ &index &ch | x&struct_base }

  AD24_Sample_NP                       \ Get sample from a/d
  &index &ch my_data 2!                \ Store to array

  &ch 1 + NUM_CHANNELS <
  if
    &ch 1 +                            \ Increment channel number
    dup
    my_struct +current_channel c!      \ Store to structure
    to &ch                             \ Store to local
  else
    0 my_struct +current_channel c!    \ Roll over channel
    0 to &ch                           \ Roll over local
    &index 1 + my_struct +index !      \ Increment index
  endif

  my_struct &ch ad_channel * xn+       \ Get base address of struct
  to x&struct_base                     \ store to local

  x&struct_base +ad_fs_cal   2@        \ Get settings for next channel
  x&struct_base +ad_zero_cal 2@
  x&struct_base +ad_freq_int  @
  x&struct_base +ad_gain     c@
  x&struct_base +ad_polarity c@
  x&struct_base +ad_res      c@
  x&struct_base +ad_bo       c@
  x&struct_base +ad_fsync    c@
  x&struct_base +ad_ch       c@
  Start_Conv_With_Values

  &ch 1 + &index 1 + * NUM_SAMPLES >=  \ Index and channel start at 0
  if
    false                              \ Done sampling
  else
    true                               \ Keep going
```

```
  endif
;

\ 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!
: Sample_Routine3 ( -- flag )

  \ Allocate memory for 10 samples from 4 sensors; each sample is
  \ 4 bytes.
  NUM_SAMPLES NUM_CHANNELS / NUM_CHANNELS 2 4 ' my_data dimensioned

  MODULE0 Init_AD24                  \ 24/7 Data Acquisition Module is
                                     \ the first module on the stack
  if
    Use_Onboard_Ref                  \ Use on-board ref for samples

    CH0 my_struct +current_channel c! \ Set ch0 as the current channel
    0   my_struct +index !            \ Init array index number
    Init_CH1                         \ Init global structure
    Init_CH2 or
    Init_CH3 or
    Init_CH0 or                      \ Init ch 0 last since it will be
                                     \ the first channel to be sampled

    \ 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 166
    \ ms.  This would amount to 666 ms for 4 channels or 1.5 Hz per
    \ channel.
    cfa.for get_sample 12 0 do_so_often \ 12 * 5ms = 60 ms
  endif
;
```