INTRODUCTION

A **device driver** is a set of subroutines used to control an I/O device.

- Device driver subroutines **initialize** devices.
- Device driver subroutines **configure** device services.
- Device driver subroutines **query (poll)** device status.
- Device driver interrupt service subroutines **handle** asynchronous device events.
- Device drivers are written by the engineer and provided to users.

An **application programming interface (API)** is a set of function prototypes for any library of functions.

- Device drivers expose an API for application writers to access devices.
- Some operating systems require standard APIs for specific devices such as printers. This allows any application to access any printer because the OS application always uses the same functions for any selected printer.
- **Embedded systems engineers** often build embedded systems without sophisticated operating systems. Thus, the engineers usually write specific device driver APIs.

Project management tools in integrated development environments support multiple files in a project.

- AVR Studio projects can contain multiple files in the project directory.
- The main control loop is usually placed in the file called **main.c**.
- Device driver filenames generally reflect the device under control. For example, **adc.h** would contain the function prototypes for an analog-to-digital converter device driver while **adc.c** would contain the subroutine implementations
- The #include preprocessor directives is used to add the device driver function prototypes to the main C file. (#include “adc.h”)
- Sophisticated build tools manage compiling all project files.

C header files declare the function prototypes, global variables, and interrupt service routines provided as part of the device driver. These **.h** files are often the only file distributed to end users because the subroutine implementations are compiled into library form for distribution. An example is provided on page 2.

C implementation files contain the C code that implements the function prototypes and interrupt service routines provided as part of the device driver. These **.c** files are often precompiled and distributed to end in library form. An example is provided on the page 3.

```
/*************************************************************
 * FILENAME:    adc.h
 * AUTHOR:      meier@msoe.edu <Dr. Meier>
 * COPYRIGHT:   This example is intellectual property of MSOE
 *              Professor Dr. Russ Meier.
 * LICENSE:     Permission is granted for use if the copyright
 *              notice is retained. No person may publish this
 *              code for public use.
 * PROVIDES:    MSOE CE2810 embedded systems example .h file
 *              - declares the function prototypes for an ADC
 *                device driver set
 * MODIFIED:    Created 10 Dec 2008
 *              - 10 Dec 2008 basic structure
 *              - 11 Dec 2008 comments
 *              - 11 Dec 2008 moved to C99 types
 * USAGE:       Atmega32 microcontroller
 *              - include this header file in any project that
 *                uses the adc
 *************************************************************/

#ifndef DRIVER_ADC
#define DRIVER_ADC

#include <inttypes.h>

// function:    adc_sample
// provides:    one-function control of ADC
// modifies:    ADMUX, ADCSRA -- all bits
// parameters:  clk_div = 2,4,8,16,32,64,128 division
//              pin = pin to convert
// returns:     left-adjusted ADCH result

uint8_t adc_sample(uint8_t clk_div, uint8_t pin);

// function:    adc_convert
// provides:    one ADC conversion result
// modifies:    ADCSRA ADEN and ADSC bits
// parameters:  none
// returns:     ADCH, use adc_set_lar before

uint8_t adc_convert(void);

// subroutine:  adc_set_clkdiv
// provides:    control of ADC system clock divider
// modifies:    ADMUX clock divider bits
// parameters:  clk_div = 2,4,8,16,32,64,128 division

void adc_set_clkdiv(uint8_t clk_div);

// subroutine:  adc_set_pin
// provides:    control of ADC pin to convert
// modifies:    ADMUX pin selection bits
// parameters:  pin to convert as a byte

void adc_set_pin(uint8_t pin);

// subroutine:  adc_set_lar
// provides:    turns on ADC left-adjusted result
// modifies:    ADMUX LAR bit only
// parameters:  none
```

etcetera…

```
#endif
```

```
/************************************************************
 * FILENAME:    adc.c
 * AUTHOR:      meier@msoe.edu <Dr. Meier>
 * COPYRIGHT:   This example is intellectual property of MSOE
 *              Professor Dr. Russ Meier.
 * LICENSE:     Permission is granted for use if the copyright
 *              notice is retained. No person may publish this
 *              code for public use.
 * PROVIDES:    MSOE CE2810 embedded systems example .c file
 *              - implements device driver subroutines defined
 *                in adc.h file
 * MODIFIED:    Created 10 Dec 2008
 *              - 10 Dec 2008 basic structure
 *              - 11 Dec 2008 comments
 * USAGE:       Atmega32 microcontroller
 ************************************************************/

// preprocessor directives
#include <avr/io.h>
#include <inttypes.h>
#include "adc.h"

// global variables : none

// function:    adc_sample
// provides:    one-function control of ADC
// modifies:    ADMUX, ADCSRA -- all bits
// parameters:  clk_div = 2,4,8,16,32,64,128 division
//              pin = pin to convert
// returns:     left-adjusted ADCH result

uint8_t adc_sample(uint8_t clk_div, uint8_t pin)
{
  // ADMUX
  // select VCC power pin
  // select left-adjusted result
  // or in user pin choice after clearing upper bits
  ADMUX = 0b01100000 | (pin & 0b00011111);
  // ADCSRA
  // power-on, start conversion, no interrupts based on clk_div
  // reference:  page 214 and 215 of Atmega32 datasheet
  switch(clk_div)
  {
    case 4:   ADCSRA = 0b11000010;
              break;
    case 8:   ADCSRA = 0b11000011;
              break;
    case 16:  ADCSRA = 0b11000100;
              break;
    case 32:  ADCSRA = 0b11000101;
              break;
    case 64:  ADCSRA = 0b11000110;
              break;
    case 128: ADCSRA = 0b11000111;
              break;
    // handle 2,2 and all other incorrect choices
    default:  ADCSRA = 0b11000000;
              break;
  }
  // wait for ADC to complete (ADCSRA bit 6 clears when done)
  // wait while the ADCSC bit is still set
  // and thus the AND is non-zero
  while ( etcetera… )
```

A complete solution will not be included in this tech note. Readers are encouraged to complete the implementation.