AVR 8-bit Microcontrollers

AVR944: LED Driver Library for AVR 8-bit Microcontrollers

APPLICATION NOTE

Introduction

Atmel® MSL series is a compact, high power set of LED drivers. This application note explains how to use Atmel AVR® 8-bit microcontrollers to interface with these LED drivers. The MSL series of drivers feature either TWI or SPI interface to read and write to their internal registers. The Atmel LED driver library provides easy to use wrapper functions to read/write to those registers using any AVR 8-bit microcontroller as master. This library supports both TWI and SPI interfaces. The table below shows the interfaces supported by this library for each AVR.

Table -1. Interfaces Supported by the Atmel LED Driver Library

Microcontroller	Interface
megaAVR	TWI
XMEGA	TWI
tinyAVR	SPI, USI (SPI)

Features

- Library to control the Atmel LED Driver MSL series of Atmel LED drivers
- Includes TWI library for Atmel XMEGA® and megaAVR®
  - Compatible with Philips' I^2C protocol
- Includes SPI library for the Atmel tinyAVR®
  - Also includes USI library to be used as SPI for tinyAVR
- MCU and application-independent application programming interface
- Easy to configure and use

Table of Contents

1. Prerequisites

The LED Driver Library used in this document requires basic familiarity with the following:

- Compiling C projects with Atmel Studio 7, as the library is written using this IDE
- General familiarity with SPI and TWI interfaces, and electrical connection requirements
- Knowledge in the basics of the MSL series register set and their use
- A programming and debugging device to debug and test the compiled application, or to download the application hex files into the targeted device, such as the AVR JTAGICE mkII or Atmel-ICE

2. Limitations

- The LED Driver Library was compiled and tested with the Atmel Studio 7 with GCC C compiler. This library was not compiled with IAR™ or any other compiler
- The LED Driver Library only supports TWI with Atmel megaAVR and XMEGA devices and only SPI and USI (SPI) with Atmel tinyAVR devices
- ‘ATXMEGA’, ‘MEGAAVR’, ‘TINYAVR’, ‘SPI’, ‘TWI’ and ‘SPI_USI’ are important keywords; do not use them elsewhere in the code as they are used as defines in the configuration file

3. Creating a Project

The table below shows the list of files contained in this library along with a short description.

To include this library in a new project, “atmel_led_device_config.h” must be configured (as explained in the following chapter) and added along with the required driver files and the compiler file (avr_compiler.h). For example, to create a project for ATxmega TWI interface, the following files should be included:

- atxmega_twi_driver.c
- atxmega_twi_driver.h
- avr_compiler.h*
- atmel_led_device_config.h*

The configuration and compiler files (marked with *) must be included irrespective of the AVR and interface as the project will not compile without these.

In the user application source file, include “atmel_led_device_config.h” by adding the following statement on the top:

```
#include "atmel_led_device_config.h"
```

This file will automatically include all the required files. The file “atmel_led_drvr_demo.c” provides a very good example of how to include and use this library.

Table 3-1. List of Files in the Atmel LED Driver Library

Source file	Description
atmega_twi_driver.c	Driver for megaAVR with TWI interface
atmega_twi_driver.h	Header file for atmega_twi_driver.c
atxmega_twi_driver.c	Driver for XMEGA AVR with TWI interface
atxmega_twi_driver.h	Header file for atxmega_twi_driver.h
tiny_avr_spi_via_usi_driver.c	Driver for tinyAVR with USI interface
tiny_avr_spi_via_usi_driver.h	Header file for tiny_avr_spi_via_usi_driver.c
tiny_avr_spi_driver.c	Driver for tinyAVR with SPI interface
tiny_avr_spi_driver.h	Header for tiny_avr_spi_driver.c
atmel_led_device_config.h	Atmel LED driver library configuration file
atmel_led_drvr_demo.c	Atmel LED driver library demo application
avr_compiler.h	AVR compiler file
documentation.h	Used only by Doxygen

4. Configuring the Library

Configure the library with only one file, “atmel_led_device_config.h”. This file contains pre-compilation directives to compile the source required only for the selected AVR and interface. The default configuration file provided with the library is set up for the Atmel ATtiny40 with SPI interface. The configuration file settings for different AVR microcontrollers are explained in this section.

4.1. Selecting the Device and Interface

There are four definitions related to device and interface. Select the target AVR and interface with the first two #defines.

```
//*************************************************************/
//User can define the device here: ATXMEGA, MEGAAVR or TINYAVR
#define ATXMEGA
//*************************************************************/
//Define the interface here SPI, TWI or SPI_USI
#define TWI
```

This example shows the target as ATxmega with TWI interface.

‘ATXMEGA’, ‘MEGAAVR’, ‘TINYAVR’, ‘SPI’, ‘TWI’ and ‘SPI_USI’ are essential keywords; do not be use them anywhere else in the code. They selectively compile only the required source files.

The next two defines are shown below.

```
//CPU Clock on which AVR is running at. It is used to calculate
//baud rate settings and delays.
#define F_CPU 20000000
//*************************************************************
//Define slave addresses here.
#define SLAVE_ADDRESS 0xA0
```

- F_CPU is the frequency in Hertz (Hz) at which the AVR is running. It calculates delays and baud rate settings.
- SLAVE_ADDRESS is the address of the slave MSL. It can be TWI slave or SPI slave. This is a seven bit address and the bit 0 is ignored. Bit 0 is used to indicate a read or a write operation.

4.2. Interface Settings

There are four sections in the “atmel_led_device_config.h” file, each used for different kind of interface. One of the four sections is compiled depending on the type of AVR and interface used.

4.2.1. Atmel XMEGA with TWI Interface

The following code section contains definitions related to this target.

```
//*********ATXMEGA TWI PARAMETERS*****************************/
#if defined(ATXMEGA) && defined(TWI)
    #include "atxmega_twi_driver.h"
    /*! \brief Largest message size that will be sent/received
    excluding address byte and register address. When using array
    write/read functions, the count parameter can not exceed
    NUM_BYTES*/
    #define NUM_BYTES 16
    /*! TWI port used, */
    #define TWI_PORT TWIC
    /*! TWI master Interrupt */
    #define TWI_INT_VECTOR TWIC_TWIM_vect
    /*BAUDRATE 100Khz*/
```

```
    #define BAUDRATE      100000
//************************************************************
```

- NUM_BYTES defines the maximum number of data bytes that can be sent in one transfer. It is used when using the array read/write commands. The count parameter can not exceed NUM_BYTES.
- TWI_PORT defines which TWI module is used
- TWI_INIT_VECTOR defines the interrupt associated with that TWI module
- BAUDRATE defines the target SCL clock frequency in hertz. For most TWI applications, it is 100kHz.

4.2.2. Atmel megaAVR with TWI Interface

The following code section contains definitions related to this target.

```
//*****************MEGAAVR TWI PARAMETERS**********************/
#elif defined(MEGAAVR) && defined(TWI)
    #include "atmega_twi_driver.h"
    /*! \brief Largest message size that will be sent/received
    excluding address byte and register address. When using array
    write/read functions, the count parameter can not exceed
    NUM_BYTES*/
    #define NUM_BYTES 16
    // Bit rate Register setting for 8MHz CPU clock, 100 KHz SCL
    #define TWI_TWBR 0x20
    #define TWI_TWPS 0x00
//************************************************************
```

- NUM_BYTES defines the maximum number of data bytes that can be sent in one transfer. It is used when using the array read/write commands. The count parameter can not exceed NUM_BYTES.
- TWI_BUFFER_SIZE defines the size of TX/RX buffer. In simple read/write transactions with MSL, three bytes are transferred one way.
- TWI_TWBR and TWI_TWPS define the baud rate register settings according to the equation below. Carefully enter these to set the SCL clock frequency.

$$SCL = \frac{CPU\_CLOCK\_FREQ}{16 + 2 * TWBR * \left(4^{TWPS}\right)}$$

4.2.3. Atmel tinyAVR with USI Interface

Most of the tinyAVR microcontrollers do not have SPI module. Instead they have a universal serial interface which can be implemented as either SPI or TWI. This library contains drivers for the SPI implementation of the tinyAVR USI. Drive the SPI interface of the MSL with the AVR USI ports to control the LED driver using this library. Since USI is a three pin interface, use an additional I/O pin as SS by the software.

The following code section contains the definitions related to this target.

```
#elif defined(TINYAVR) && defined(SPI_USI)
        #include "tiny_avr_spi_via_usi_driver.h"

        /* USI port and pin definitions for ATTINY25*/
        #define USI_OUT_REG   PORTB //!< USI port output register.
        #define USI_IN_REG    PINB  //!< USI port input register.
        #define USI_DIR_REG   DDRB  //!< USI port direction register.
        #define USI_CLOCK_PIN PB2   //!< USI clock I/O pin.
        #define USI_DATAIN_PIN     PB0    //!< USI data input pin.
        #define USI_DATAOUT_PIN    PB1    //!< USI data output pin.
        #define CSB_PIN            PB4    //!< Atmel LED driver CSB
        #define TC0_PRESCALER_VALUE 1 //!< Must be 1, 8, 64, 256, 1024
        #define TC0_COMPARE_VALUE  31 //!< Must be 0 to 255. Minimum 31
                                      //with prescaler CLK/1.
```

- USI_OUT_REG defines the USI port used
- USI_IN_REG defines the USI port input register
- USI_DIR_REG defines the USI port direction register
- USI_CLOCK_PIN defines the USCK pin (SCK)
- USI_DATAIN_PIN defines the USI data in (MISO) pin
- USI_DATA_OUT defines the USI data out (MOSI) pin
- CSB_PIN is the software controlled I/O pin (SS). Since USI is a three pin interface, use this pin, which is software controlled, to drive the MSLxxxxCSB input.
- TCO_PRESCALAR_VALUE and TCO_COMPARE_VALUE set the SCK bits-per-second of the interface according to the equation below

Use prescaler value of {1, 8, 64, 256, or 1024} and compare value 0 to 255. Use at least 31 when using prescaler value of 1. Hence maximum SCK bits-per-second is F_CPU/64.

$$bps = \frac{F\_CPU}{PRESCALER * (COMPARE\_VALUE + 1) * 2}$$

4.2.4. Atmel tinyAVR with SPI Interface

For the tinyAVR microcontrollers with built in SPI module, the LED Driver Library contains tinyAVR SPI drivers to interface with Atmel LED driver chips like the Atmel MSL2160. The following code section contains the built-in SPI definitions.

```
#elif defined(TINYAVR) && defined(SPI)
        #include "tiny_avr_spi_driver.h"
        //This is the SCK divider setting as defined by SPR[1:0]
        #define SPI_CLK_DIVIDER 4 //Must be 4,16,64,128
        //This is the SPI2X bit setting which, if set, doubles the SCK.
        #define SPI_CLK_DOUBLE 1 //Must be 1 or 0
        #define SPI_DIR_REG  DDRC   //!< SPI port direction register.
        #define SPI_PORT     PORTC  //!< SPI port output register.
        #define SPI_SCK      PC1    //!< SPI clock I/O pin.
        #define SPI_MISO     PC2    //!< SPI data input pin.
        #define SPI_MOSI     PC4    //!< SPI data output pin.
        #define SPI_SS       PC0    //!< Atmel LED driver CSB pin
```

- SPI_CLOCK_DIVIDER and SPI_CLK_DOUBLE determine the SCK frequency according the equation below
- Use clock divider value of {4, 16, 64, or 128} and clock double value 0 or 1
- SPI_DIR_REG defines the SPI port direction register
- SPI_PORT defines the SPI port output register
- SPI_SCK defines the SCK pin
- SPI_MISO defines the MISO pin
- SPI_MOSI defines the MOSI pin
- SPI_SS defines the Slave select pin which is connected to MSL CSB pin

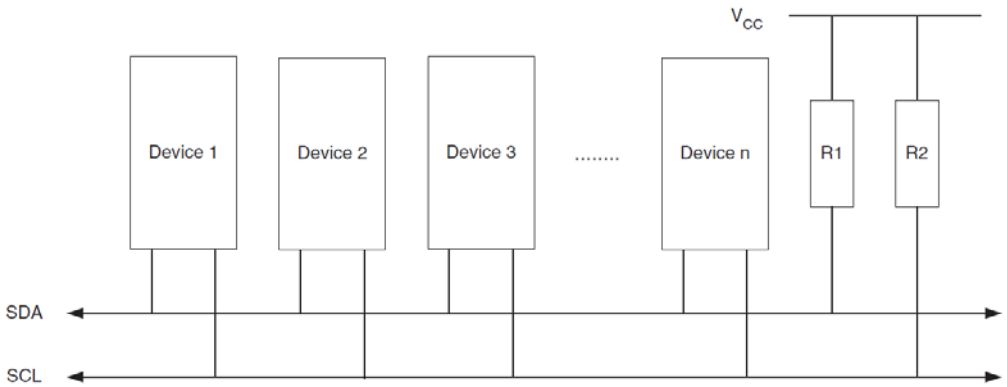
$$bps = \frac{F\_CPU}{SPI\_CLK\_DIVIDER} * 2^{SPI\_CLK\_DOUBLE}$$

5. Electrical Connections

TWI

To connect a MSL as a TWI slave to AVR master (megaAVR or XMEGA), pull up SDA and SCL lines with a 10kΩ resistor as shown in the figure below. In this figure, R1 and R2 are 10kΩ resistors. Devices 1 to n are multiple MSL TWI slaves.

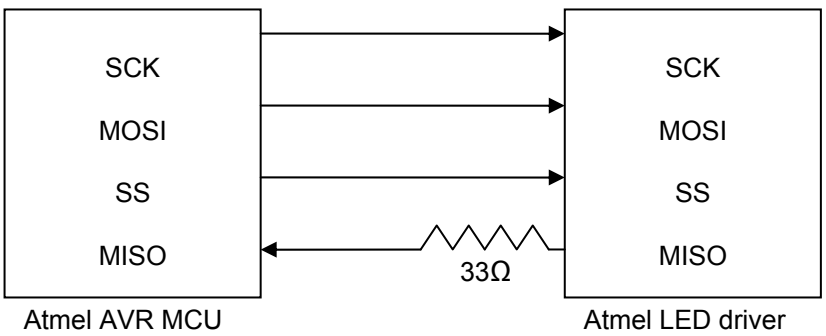
Figure 5-1. TWI Connections



SPI

To connect an MSL as an SPI slave to tinyAVR SPI (or USI) master, do not pull up MISO, MOSI, SCK, or CSB, but add a 33Ω series resistor on MISO between the AVR and MSL as shown in the figure below.

Figure 5-2. SPI Connections



6. Using the Library

This section explains the functions available to the user by this library. The prototypes remain the same irrespective of the Atmel AVR and interface used. For this purpose, this section assumes that the file “atmel_led_device_config.h” is correctly configured.

The library provides the following application programmer interfaces (APIs). More information on these can be found in the accompanying Doxygen documentation.

atmel_led_drvr_init()
This function configures the AVR to act as a master. Call this function before further communication with the slave. This function returns void.

atmel_led_drvr_writeregister(slave_address, REG_ADDR. REG_DATA)
This function accesses the MSLxxxx slave at slave_address and writes REG_DATA to its internal register at REG_ADDR. The function returns 1 if successful, else it returns 0.

atmel_led_drvr_readregister(slave_address, REG_ADDR, *receivedData)
This function accesses the slave at slave_address and reads its internal register at REG_ADDR and stores the data in receivedData pointer. The function returns 1 if successful, else it returns 0.

atmel_led_drvr_writearray(slave_address, REG_ADDR, *Data, count)
This function writes a byte array of length ‘count’ pointed by ‘Data’ pointer to a slave at slave_address, starting at register address REG_ADDR. For XMEGA and megaAVR devices, this count cannot exceed the NUM_BYTES defined in the configuration file.

atmel_led_drvr_readarray(slave_address, REG_ADDR, *Data, count)
This function reads a byte array of length ‘count’ from a slave at slave_address, starting at register address REG_ADDR into a buffer pointed by ‘Data’ pointer. For XMEGA and megaAVR devices, this count can not exceed the NUM_BYTES defined in the configuration file.

Writing user defined functions
Using the three primitive functions allows writing more user friendly functions which read to and write from to specific registers of LED drivers. For example, the global intensity register of Atmel MSL2160 has an internal address of 0x1F. This register sets global LED intensity. The values 0 to 0xFF correspond to 0 to 100% brightness respectively. The example code below shows a simple wrapper function to set global intensity of a particular slave:

```
char SetBrighntessLevel(char slave_addr, char intensity)
{
    return atmel_led_drvr_writeregister(slave_addr, 0x1F,
                                        intensity);
}
```

7. Demo program

The file “atmel_led_drvr_demo.c” contains the source for a demo program for Atmel ATtiny40 AVR with SPI interface to connect to Atmel MSL2160 Evaluation board. The program accesses a MSL2160 slave and writes ‘0xAA’ to its register at ‘0x00’ address and reads the register back. This operation is done in an infinite loop.

Hardware Requirements

The demo setup needs the following components:

- ATtiny40 AVR with any development kit (like the Atmel STK®600) which exposes the MCU pins
- MSL2160 Evaluation board with LED load board and power supply
- 33Ω resistor

8. References

AVR151: Setup and Use of the SPI on megaAVR and tinyAVR:

http://www.atmel.com/images/atmel-2585-setup-and-use-of-the-spi_applicationnote_avr151.pdf

AVR311: Using the TWI Module as I^2C Slave:

http://www.atmel.com/images/atmel-2565-using-the-twi-module-as-i2c-slave_applicationnote_avr311.pdf

AVR315: Using the TWI Module as I^2C Master:

http://www.atmel.com/images/atmel-2564-using-the-twi-module-as-i2c-master_applicationnote_avr315.pdf

AVR319: Using the USI module for SPI communication:

http://www.atmel.com/images/doc2582.pdf

AVR1308: Using the XMEGA TWI:

http://www.atmel.com/images/doc8054.pdf

AVR1309: Using the XMEGA SPI:

http://www.atmel.com/images/doc8057.pdf

MSL2100 datasheet:

http://www.atmel.com/dyn/resources/prod_documents/F1_MSL2100_DB.pdf

MSL2160/61 datasheet:

http://www.atmel.com/dyn/resources/prod_documents/F1_MSL2160_DB.pdf

Atmel Studio 7:

http://www.atmel.com/studio

9. Revision History

Doc Rev.	Date	Comments
8464B	08/2016	New template and some minor updates
8464A	11/2011	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**