Features

- Supported by all Atmel® AVR® devices with the required peripherals (TWI, SPI, RTC)
- Reusable modules for other applications
- SD card FAT file system access specifically for devices with limited available memory
- TWI Master driver
- AT30TSE75x temperature sensor driver
- Using the RTC and predefined GCC macros for keeping track of real time

Introduction

This application note describes the implementation of a temperature logger, demonstrating use of several peripherals of the ATtiny817 AVR device. The TWI peripheral is used to communicate with an Atmel AT30TSE digital temperature sensor. The asynchronous RTC module is used to generate a timestamp for each temperature measurement, and the data is written to an SD card via an SPI interface using the ELM-Chan Petit FAT file system library. The example code can be used with either an ATtiny817 Xplained Pro or an ATtiny817 Xplained Mini (with some modifications, see the Required Hardware chapter for more information), and an I/O1 Xplained Pro Extension Kit. It is possible to reuse parts of the code for other applications requiring SD card file system access, using the TWI module in master mode, interfacing with an AT30TSE temperature sensor, or using the RTC to keep track of real time.
Table of Contents

Features........................................................................................................................................... 1

Introduction......................................................................................................................................1

1. TWI for Temperature Sensor Interfacing....................................................................................3
   1.1. Floating Point Values with vfprintf for AVR............................................................................ 3
   1.2. Buffer Construction...................................................................................................................3

2. Using the RTC for Real Time Tracking.......................................................................................4
   2.1. Predefined Macros for Time Initialization............................................................................... 6
   2.2. RTC for Time Stamping of Temperature Measurements............................................................7

3. Petit FAT File System...............................................................................................................8
   3.1. Petit FatFs Limitations..............................................................................................................8
   3.2. SPI for SD Card Interfacing.....................................................................................................8
   3.3. Preparing an SD Card for use with Petit FatFs........................................................................9
   3.4. SD Card Disk I/O Layer for AVR............................................................................................9

4. Required Hardware...................................................................................................................10
   4.1. Changing Operating Voltage on the ATtiny817 Xplained Mini...............................................10
   4.2. Mounting the I/O1 Xplained Pro Extension Kit onto the ATtiny817 Xplained Mini..................12

5. Get Source Code from Atmel START......................................................................................15
   5.1. Configuration of Accompanying Code....................................................................................15
   5.2. Example Output......................................................................................................................15

6. Revision History.......................................................................................................................17
1. **TWI for Temperature Sensor Interfacing**

The accompanying code to this application note provides a driver for use of an Atmel AT30TSE temperature sensor, which depends on the TWI master driver, which is also provided. Using these drivers, the configuration of the TWI and temperature sensor is very simple, as follows:

- The pin location of the TWI module should be checked and if necessary, the alternate pin location selected in the PORTMUX peripheral
- Initialize the temperature sensor (requires an already defined TWI master variable)

**Note:** An ISR should be defined for the TWI Master interrupt vector (TWI0_TWIM_vect) which calls the TWI Master Interrupt handler function. This is required to complete a TWI transaction.

**Note:** If using a device with more than one TWI module and TWI0 is not the desired module to use for the temperature sensor, certain symbols should be defined, refer to 'at30tse75x.c'.

1.1. **Floating Point Values with vfprintf for AVR**

The default implementation of vfprintf and its variants for AVR devices does not include floating point conversions, for the purpose of decreasing code compile size. One option is to use linker options to include support for floating point values as seen here*, but it should be kept in mind that this will require a significant amount of extra memory, which may not be available in devices with small memories. Another option, which is used in this application note, is to perform calculations on the floating point value to give the integer and fraction parts to the desired number of decimal places, and treat both parts as integers with a decimal point in between in the character string.

**Note:** * http://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html.

1.2. **Buffer Construction**

The function to read temperature uses a double precision floating point argument for loading the temperature into, but for the purposes of buffer construction using sprintf, the integer and fraction parts are both treated like integers, as mentioned above. This buffer also includes a time stamp, by using the RTC.
2. **Using the RTC for Real Time Tracking**

The RTC peripheral can be used to keep track of real time. The period is configurable, so it is not necessary to generate an interrupt every second; this enables a decrease in power consumption. An overflow need only be generated when the current time is required e.g. for time stamp generation. The period can be changed from anywhere between one second to more than 18 hours.

**Note:** Ideally, this method requires that an accurate oscillator is connected to the TOSC pins and the necessary oscillator design considerations have been adhered to. Significant error is introduced when the oscillator used as the RTC clock source has low accuracy. For the purposes of demonstration, the available oscillators on the Xplained Pro and Mini kits are used. However, in practice these may not be accurate enough.

The RTC should be configured as follows:

**Note:** Because the RTC and the CPU run asynchronously, the STATUS register of the RTC should be used to verify synchronization of register settings before moving on.

- The required oscillator should be enabled in the CLKCTRL module, and selected as the RTC clock source (ideally a highly accurate 32.768kHz oscillator)
- The RTC should be allowed to run in standby
- The prescaler should be set to an appropriate value, to give a count increment once per second (e.g. for a 32.768kHz oscillator, choose DIV32768)
- The period should be set to the desired overflow interval
- The RTC interrupt flags should be cleared and the overflow interrupt enabled
- Finally, the RTC should be enabled to start counting

Additionally, an ISR should be defined for the RTC overflow interrupt vector (RTC_CNT_vect), to complete the time update and any other required periodic tasks such as a sensor measurement and time stamp string generation. Two subroutines are used to update the time. They use a time struct as defined below.

```c
typedef struct{
    uint8_t second;
    uint8_t minute;
    uint8_t hour;
    uint8_t date;
    uint8_t month;
    uint16_t year;
}time;
```

The 'update time' subroutine increments the 'second' by the defined number of seconds in the period, and checks for the necessity to increment the higher value members, i.e. 'minute' is incremented if 'second' is larger than or equal to 60, and so on. The complete functionality of the subroutine is depicted in the figure below.
Figure 2-1. Update Time Subroutine

- **Update time**
  - **Second += Period**
    - **Second >= 60?**
      - **Y**
        - **Minute++**
        - **Second -= 60**
      - **N**
        - **Minute = 60?**
          - **Y**
            - **Hour++**
            - **Minute = 0**
          - **N**
            - **Hour = 24?**
              - **Y**
                - **Date++**
                - **Hour = 0**
              - **N**
                - **N**
                  - **End**
- **Date = 32?**
  - **Y**
    - **N**
      - **Date = 31? & Month = 4,6,9,11?**
        - **Y**
          - **N**
            - **Date = 30? & Month = 2?**
              - **Y**
                - **Date = 29? & Month = 2? & not_a_leap_year?**
                  - **Y**
                    - **Month++**
                    - **Date = 1**
                  - **N**
                    - **Month = 13?**
                      - **Y**
                        - **Year++**
                        - **Month = 1**
                      - **N**
                        - **Month = 1**
      - **N**
        - **Month = 13?**
          - **Y**
            - **Year++**
            - **Month = 1**
          - **N**
            - **End**
The 'not a leap year?' subroutine is used by the 'update time' subroutine, when checking whether 'month' needs to be incremented during February. It returns true if it is not a leap year, i.e. February has 28 days, and false if it is a leap year, i.e. February has 29 days. Its calculations can be seen in the figure below.

**Figure 2-2. Leap Year Check Subroutine**

2.1. **Predefined Macros for Time Initialization**

When using the RTC to generate a time stamp, there is a necessity for a correct initial time. Instead of manually entering the initial value of the time struct, it is possible to use predefined GCC macros for application compilation date and time. It should be noted that a fresh compilation must be completed immediately before flashing the device. If there is a delay between compilation and flashing, the initial time will be incorrect, and therefore also the generated time stamp. As can be seen here*, there are strings defined as follows, which can be processed accordingly into the predefined time struct members:

- **__DATE__** string, of the format "Feb 12 1996" for date, month, and year
2.2. **RTC for Time Stamping of Temperature Measurements**

In this application note, the RTC is used to keep track of real time in order to take a temperature measurement at defined intervals and generate a corresponding time stamp. In this case, a flag "tick" is raised to indicate that the subroutine "tock" should be called from the main while loop. This is done instead of calling "tock" directly from the ISR to ensure that the procedure does not have interrupt priority. The TWI module used to interface with the temperature sensor requires an interrupt to complete operation; this means a deadlock may occur from an interrupt being generated inside another interrupt if their respective priorities are not correctly configured. In the given example, the "tock" subroutine is called from the main loop for simplicity, and therefore configuration of interrupt priorities is not necessary. The functionality of the "tock" subroutine includes taking a temperature measurement, updating the time, and sending the updated buffer to the SD card.
3. **Petit FAT File System**

An SD card is a very convenient way to store large amounts of non-volatile data. A file system is generally used to allow the data to be portable and easily accessible from a PC. An industry standard, widely used file system is FAT. Many AVR applications require optimized usage of flash memory and SRAM; when only limited interfacing is needed or insufficient resources are available, a normal FAT file system module may be unsuitable, hence the need for the subset Petit FAT file system module (Petit FatFs), available from ELM-ChaN*. It is specifically designed for 8-bit microcontrollers such as AVR 8-bit microcontrollers, and enable basic interfacing with an FAT file system formatted MMC or SD card while requiring limited resources. It is platform independent and so requires a lower disk I/O layer to be functional, but sample drivers are available.

The functionality provided by the Petit FatFs library includes:

- Mounting a volume with a FAT file system
- Opening a file
- Read a file
- Write to a file (with some restrictions)
- Move R/W pointer
- Open a directory
- Read a directory item

Configuration regarding inclusion or exclusion of any function is defined in the configuration header file (pffconf.h), in order to minimize compile size.

**Note:** *http://elm-chan.org/fsw/ff/00index_p.html.

3.1. **Petit FatFs Limitations**

Petitfs specifically uses as little flash and stack as possible. This, however, comes at the expense of some functionality. Files are not able be created or increased in size, and only one file can be accessed at a time. The mode of sector access is limited in terms of efficiency, and therefore there are significant periods where the SDC/MMC is busy. This busy period occurs for every partial sector access operation, so in applications where an entire sector cannot be read or written at once, the maximum access frequency further decreases.

3.2. **SPI for SD Card Interfacing**

Control of multimedia and SD cards without a native host interface is possible by using the card’s SPI mode. The communication protocol is relatively simple, as described in detail here (http://elm-chan.org/docs/mmc/mmc_e.html) by ELM-Chan. It is possible to use an AVR SPI peripheral for this with ease. The necessary configuration for SD card initialization is as follows:

- The desired pin locations should be checked, and the alternate location selected if necessary in the PORTMUX peripheral
- The MOSI, CS, SCK pin directions should be set as output
- A pull up should be enabled on the MISO pin (the SD card DO pin - this is very important specifically for SD cards; without it they will fail to initialize)
- Master SPI mode should be enabled
- An SPI prescaler should be selected to result in a frequency between 100 and 400kHz (this is also very important for SD card initialization). This will vary depending on the peripheral clock frequency. More information can be found in the device datasheet CLKCTRL chapter.
• Buffer mode should be enabled to make flags available for transmit and receive complete
• Wait for Receive should be enabled to allow data to go directly to the shift register
• The slave select line should be disabled (this should be controlled in software)
• SPI Mode 0 (default setting, recommended) or SPI Mode 3 (will work in most cases) should be selected
• Finally, the SPI should be enabled

After initialization, the SPI clock prescaler and 2X setting can be changed to result in a faster SPI clock.

3.3. Preparing an SD Card for use with Petit FatFs
The write function of Petit FatFs is unable to create files or expand the size of files that already exist. Therefore, before starting to use an SD card with Petit FatFs, it is necessary to first insert it into a PC, format it to FAT file system, and create a file of the desired size, with a known name.

3.4. SD Card Disk I/O Layer for AVR
The disk I/O layer code for AVR used in this application note (diskio_avr.c) is a modified version of a sample file available from ELM-Chan*. The SD card settings described above are included in the initialization routine. The hash defines for SPI pins and the functions for SPI initialization, transmit, and receive should be reviewed for use with a device other than the ATtiny817 and modified if necessary. Use of the Petitfs library is demonstrated in the 'init_sd_card()' and 'send_buffer()' functions in the main file.

Note: * http://elm-chan.org/fsw/ff/pfsample.zip

Note: File read is not used in this application, so its functionality has been disabled in the 'pffconf.h' file to save memory.
4. Required Hardware

In order to use the example code for this application note on an ATtiny Xplained Pro, simply connect the I/O1 Xplained Pro Extension kit to extension header 1 (EXT 1).

In order to use an ATtiny817 Xplained Mini instead of an Xplained Pro a few modifications are necessary, described below.

4.1. Changing Operating Voltage on the ATtiny817 Xplained Mini

The ATtiny817 Xplained Mini has a default $V_{CC}$ of 5V. Many extension kits require 3.3V; in order to enable the boards to communicate effectively, they need to operate at the same voltage. It is possible to change the operating voltage on the Xplained Mini with only a few steps. Figure ATtiny817 Xplained Mini Power Supply Circuitry Schematic shows the schematic for the power supply circuitry on the Xplained Mini. By removing the 0Ω resistor R100, and making the 3.3V connection outlined in figure ATtiny817 Xplained Mini Supply Voltage Choices, the operating voltage will be changed from 5V to 3.3V.

Figure 4-1. ATtiny817 Xplained Mini Power Supply Circuitry Schematic
Figure 4-2. ATtiny817 Xplained Mini Supply Voltage Choices

- Make this connection for 3.3V VCC
- Make this connection for 5V VCC

Remove this Resistor

R100

Atmel AVR42780: Temperature Logger with ATtiny817 and SD Card [APPLICATION NOTE]
4.2. Mounting the I/O1 Xplained Pro Extension Kit onto the ATtiny817 Xplained Mini

The figure below shows the finished hardware modifications with the I/O1 Xplained Pro Extension Kit mounted. In order to mount it like this, it is necessary to solder a 2x10 100mil pin header, pull up resistors on the I²C lines, and strap wires to connect the relevant pins to the pin header.

Figure 4-3. Finished Hardware Modifications with I/O1 Xplained Pro Extension Kit

The required connections for this project include SPI (for interfacing with the SD card), I²C (for TWI communication with the temperature sensor), LED (because the one on the Xplained Mini uses the same
pin as SPI_SCK), power and ground. These connections can be seen in figure *Strap Wire Connections Finished*, and are indicated more clearly in the figure *Strap Wire Connections to Mount I/O1 Xplained Pro Extension Kit*. The boxes indicate the color of the wire in the photo, and where there are multiple wires of the same color, a number indicates which two boxes match. The yellow boxes with an asterisk (e.g. 1*) indicate the connection should be made with a resistor (the ones used in the photo are 2.7kΩ).

Figure 4-4. Strap Wire Connections Finished
Figure 4-5. Strap Wire Connections to Mount I/O1 Xplained Pro Extension Kit

- SPI
- GND
- I2C
- VCC
- GPIO(LED)
- Resistor
5. **Get Source Code from Atmel START**

The example code is available through Atmel START, which is a web-based tool that enables configuration of application code through a graphical user interface. The code can be downloaded for both Atmel Studio 7.0 and IAR™ IDE via the **Examples**-link below, or the **BROWSE EXAMPLES** button on the Atmel START front page.

**Web page:** [http://start.atmel.com/](http://start.atmel.com/)

**Documentation:** [http://start.atmel.com/static/help/index.html](http://start.atmel.com/static/help/index.html)

**Examples:** [http://start.atmel.com/#examples](http://start.atmel.com/#examples)

In the Examples-browser, search for: AVR42780 Temperature Logger (press **User Guide** in Atmel START for detailed requirements for the example project).

Double-click the downloaded .atzip file and the project will be imported to Atmel Studio 7.0.

For information on how to import the project in IAR, press the **Documentation**-link above, select ‘Atmel Start Output in External Tools’ and 'IAR Embedded Workbench™'.

5.1. **Configuration of Accompanying Code**

The accompanying code to this application note enables easy reconfiguration of the sample period. In order to change it from the default value of 30 seconds, all that needs to be done is change the hash define "SAMPLE_PERIOD" at the top of the main file. This value can be changed to any value between 1 and 65535 seconds (more than 18 hours).

If an ATtiny817 Xplained Mini is being used, change the hash define "MINI_BOARD" from "false" to "true". This affects the LED output pin and the clock source for the RTC.

5.2. **Example Output**

The output format of the example project can be seen in the figure below. This example is with the setting SAMPLE_PERIOD set to 2 (seconds).
Figure 5-1. Example Output of the Temperature Logger Demo
# Revision History

<table>
<thead>
<tr>
<th>Doc. Rev.</th>
<th>Date</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>42780A</td>
<td>10/2016</td>
<td>Initial document release</td>
</tr>
</tbody>
</table>