APPLICATION NOTE

AT03821: SAM D20 GPS Tracker Reference Design

ASF PROJECT DOCUMENTATION

Features

- Location requests and other commands can be issued over SMS
- Logging of current GPS location to a SD Card
- RTC time periodically synced to the GPS UTC time
- Low power FreeRTOS™ tickless sleep while idle
- Panic button for remote alerting of current location
- Basic Geofence alerts when the tracker enters/exits an area
- Accelerometer to sense movement for better GPS power management
- Conversion script to convert tracker log data to the open GPX format for importing into a variety of PC applications for analysis

This reference design provides an implementation of a complete GPS Tracker application, useful for real-time and logged positional tracking of people and assets. Example uses include asset tracking for theft prevention, location tracking of children by parents, package tracking for long-distance shipping, and historical logging of hiking, and other recreational activities.

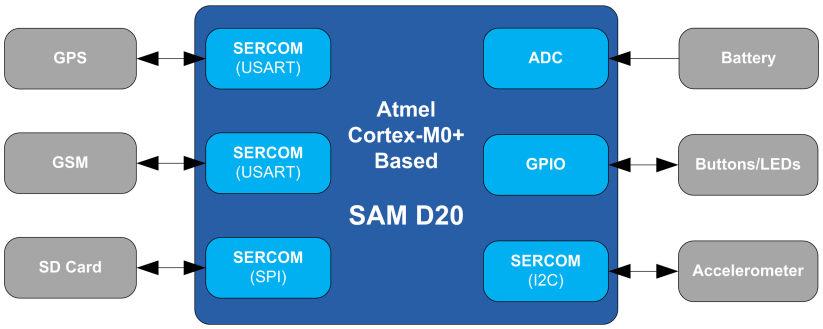


Table of Contents

1. Overview

This document describes the SAM D20 based GPS Tracker reference design.

The design implements a low power GPS tracker application, capable of logging the current device location from a GPS module to a SD Card, and responding to commands sent via SMS over a GSM modem.

The application is designed to run on the Atmel® SAM D20 series of low-power microcontrollers, which is a 32-bit ARM® Cortex®-M0+ based device series with a rich peripheral set, good processing performance, and low unit cost.

The application's reference hardware is designed to be used either in combination with the Atmel *SAM D20 Xplained Pro* board, or as a standalone unit with its own on-board processor. Information of Atmel's new *Xplained Pro* series of development boards can be found at http://www.atmel.com/xplainedpro/.

1.1 Compiling the Source Code

The GPS Tracker project is written for the GNU GCC compiler; other compilers may or may not work. The Atmel ARM GCC toolchain is available either integrated into Atmel Studio, or as a standalone package from the Atmel Website (http://www.atmel.com).

Atmel Studio is a free, modern development environment for Atmel AVR® and SAM devices, and can be downloaded directly from the Atmel Website (http://www.atmel.com/atmelstudio).

The project documentation is written for the Doxygen documentation generation tool (http://www.doxygen.org).

1.2 Application Configuration

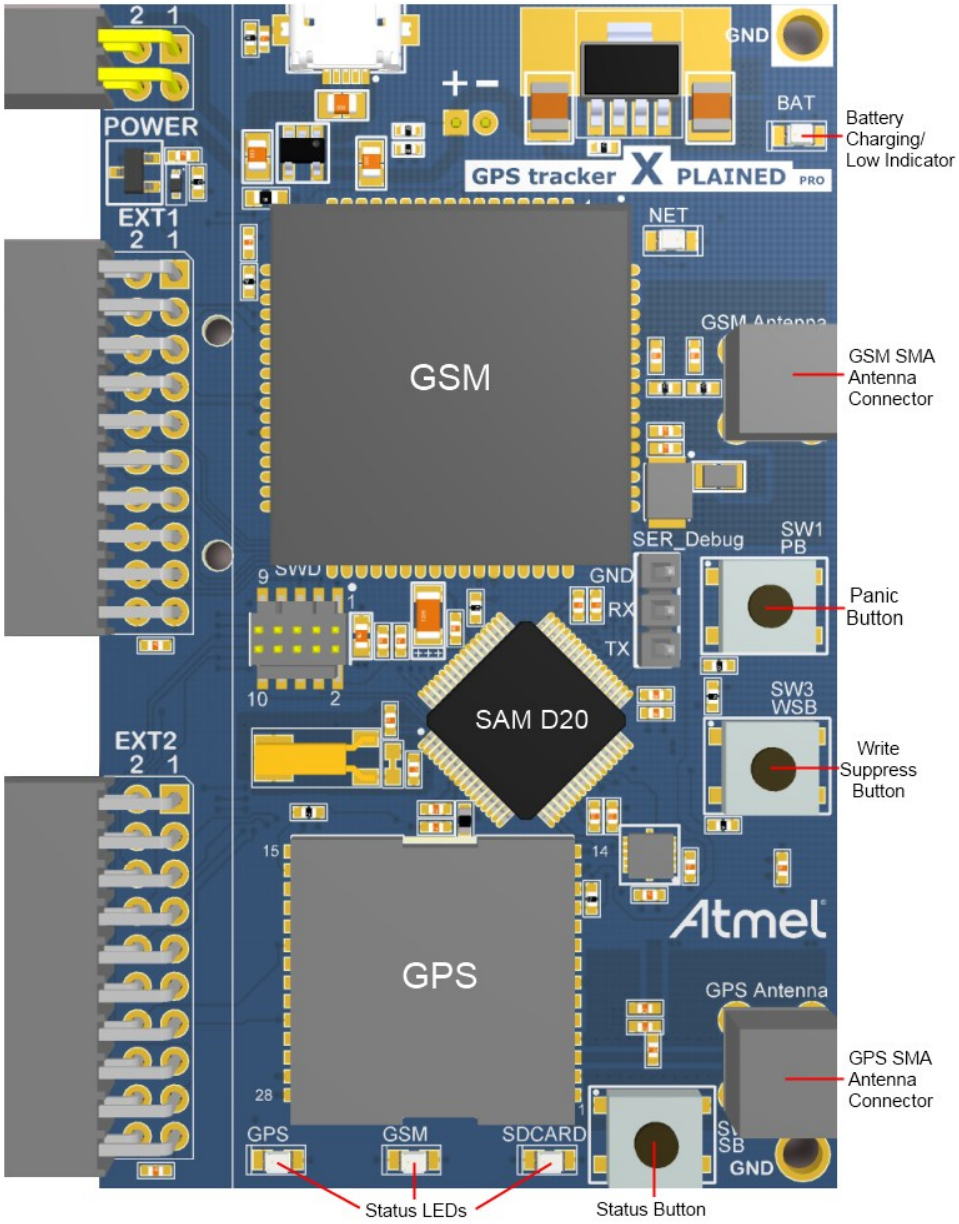
Various application-specific configuration settings to control the system behavior (and to control debug and diagnostics) can be set inside the `conf_application.h` header of the application.

1.3 Source Code License

2. Hardware Design

The board layout is shown below, with the major components labeled. Not all components are mounted on all versions on the *GPS Tracker Xplained* board; Wing and Standalone variants of the board will have various components non-mounted.

Figure 2-1. GPS Tracker Xplained Overview



2.1 Component Selection

Components for the *GPS Tracker Xplained* were carefully selected to ensure their wide availability, and to reduce the overall system cost. Note that in many cases, alternative parts may be used according to customer needs.

2.1.1 Accelerometer

To reduce the power consumption of the system to an acceptable level during periods of no movement activity, an accelerometer was selected to provide asynchronous wake-ups of the main processor when physical movement of the board was detected.

The BMA150 accelerometer from Bosch Sensortec (http://www.bosch-sensortec.com) is a low power 3-axis accelerometer with a high sensitivity and the ability to provide asynchronous, configurable motion interrupts.

Due to its popularity in a variety of consumer products, it is widely available at many distributors, making it ideal for use in the *GPS Tracker Xplained* design.

2.1.2 Battery Charger

Modern devices are expected to contain integrated battery chargers, as the market has moved to small form factor, high peak current, high capacity Lithium Ion batteries for portable consumer goods. With these batteries also comes the requirement for an intelligent integrated battery charging mechanism.

The MAX1555 from Maxim (http://www.maximintegrated.com) is a low-cost single chip solution battery charger, driven from a standard 5V input rail. This gives the *GPS Tracker Xplained* a convenient method of recharging the battery via a standard USB connection through the on-board (power-only) USB charging port.

2.1.3 GPS Module

Many GPS modules are in production from various vendors, with minor variations on price, accuracy, position fix speed, and indoor tracking performance. For the *GPS Tracker Xplained* the Copernicus II GPS receiver module from Trimble (http://www.trimble.com) was selected due to its relatively low cost and modest active/sleep mode current requirements.

As the majority of vendor GPS modules are capable of outputting locational data in the global NMEA 0183 standard GPS format, other modules may be substituted according to the customer's performance requirements and price limitations.

The Trimble Copernicus II runs directly from the same 3.3V regulated supply as the SAM D20, and thus does not require additional voltage level conversion in the design.

2.1.4 GSM Module

For the GSM module, the common SIM900 module from SimCom (http://wm.sim.com) was selected. This all-in-one module provides a standard Hayes "AT" command set for call and message management with multi-band operation for use on global GSM network carriers. As it is a commonly used GSM module, it has wide availability and is suitable for OEM integration into custom designs.

The SIM900 is designed to be powered directly from a 2.7V-4.2V Lithium Ion battery, reducing the system cost by not requiring an additional regulator with high current output (due to the significant burst current requirements of the module during transmissions). However, an unfortunate electrical characteristic of the SIM900 is a 3.1V absolute maximum voltage on its digital interface pins. As the rest of the system runs from a common 3.3V rail regulated from the battery, external level conversion between the module and the SAM D20 processor is required.

2.2 Power Rails

The system uses several voltage rails:

- **VBAT**, the raw battery voltage (2.7V-4.2V)
- **VBUS**, the nominal 5V supply from the USB charger port
- **VGSM**, the filtered VBAT battery voltage for the GSM module
- **VCC_P5V0**, the 5V supply from the *SAM D20 Xplained Pro* (if attached)
- **VCC_3V3**, the output of the on-board 3.3V regulator
- **VCC_2V8**, the output of the GSM module's internal 2.8V regulator

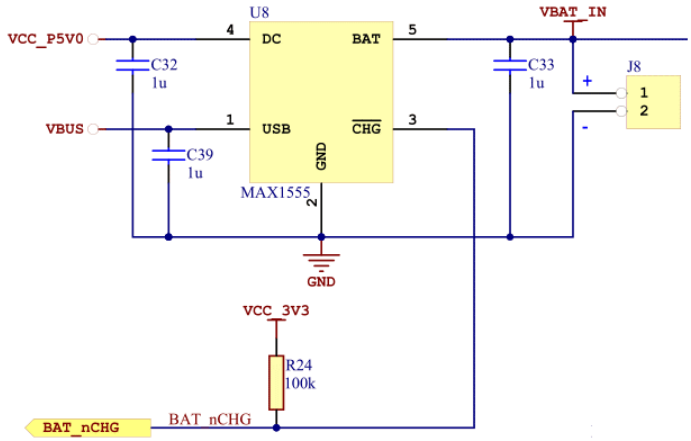
Of the above, only one external regulator is required, a 3.3V Low Drop-Out (LDO) type to reduce the raw battery voltage into a stable, clean 3.3V logic supply for the SAM D20 and other components. The remaining rails are sourced directly from on-board or external sources.

Of note is the VCC_2V8 power rail; this low current rail is supplied by the GSM module, and is used on the GSM-side of the voltage converters between the SAM D20 and the GSM module. However, it is available only when the GSM module is turned on, and thus is not present by default.

2.3 Battery Charger

The battery charger circuit is based on the "Typical Application Circuit" schematic given in the MAX1555 component datasheet. It gives the capability of charging the on-board Lithium Ion from either the on-board USB charging connector, or the 5V rail from the attached *SAM D20 Xplained Pro* (Standalone board configuration only).

Figure 2-2. Battery Charger Schematic

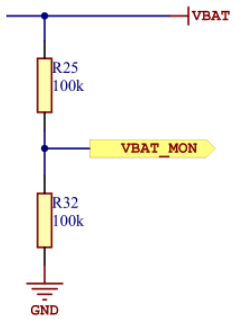


The MAX1555's active-low charge status pin /CHG is routed to the SAM D20 to provide charging state information to the application software.

2.4 Battery Voltage Monitor

Monitoring of the raw battery voltage is performed by a simple voltage divider on the board, halving the input battery voltage so that it is within the ADC input voltage range of the SAM D20. A pair of 100K resistors were chosen to reduce the static current draw of the measurement circuit on the system.

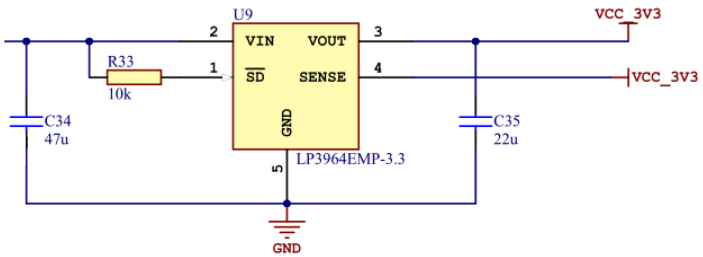
Figure 2-3. Battery Voltage Monitor Schematic



2.5 Main 3.3V Regulator

The primary 3.3V logic rail for the system is derived from the 3.3V LDO regulator, chosen so that it will continue to output a stable voltage even when the battery voltage drops to close to the main regulator's minimum input voltage. The input VIN and output VOUT pins have bulk capacitors attached to reduce noise.

Figure 2-4. 3.3V LDO Regulator Schematic

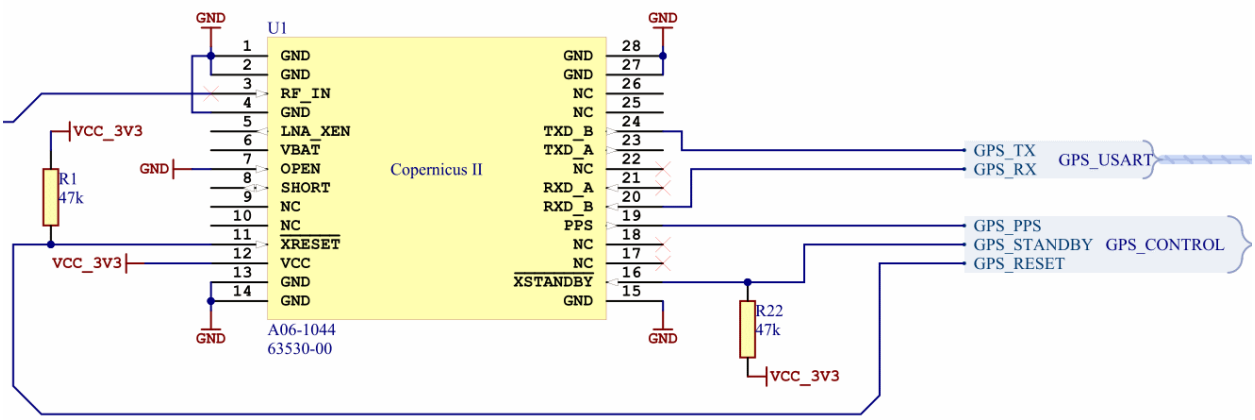


As the 3.3V rail is required for correct system operation at all times, the active-low SD shut-down pin of the regulator is fixed to a high logic level.

2.6 GPS Module

The GPS module is connected so that its TXD_B and RXD_B pins (secondary USART transmit and receive lines, respectively) are connected to the SAM D20. The secondary interface was chosen over the primary as the former outputs the required NMEA0183 format data required by the system by default, while the latter outputs a vendor specific binary protocol.

Figure 2-5. GPS Module Schematic

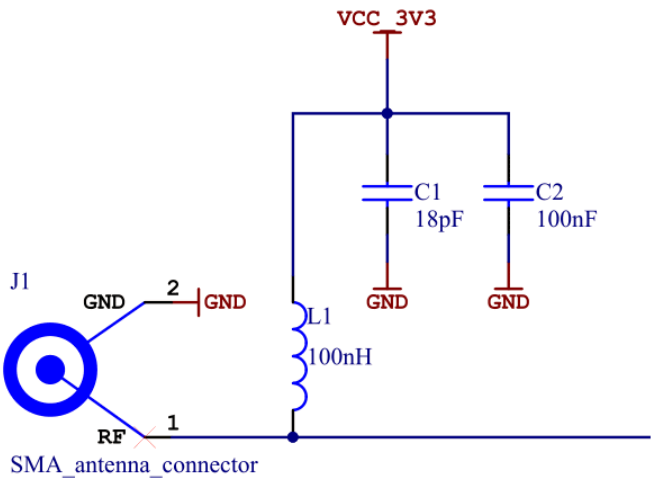


The active-low XSTANDBY pin is connected via an external pull-up to the SAM D20, so that the GPS unit can be placed into a low power mode on demand.

2.6.1 Active Antenna

To support active antennas, 3.3V power rail is connected via a filter circuit to the antenna connector.

Figure 2-6. GPS Active Antenna Schematic



2.7 GSM Module

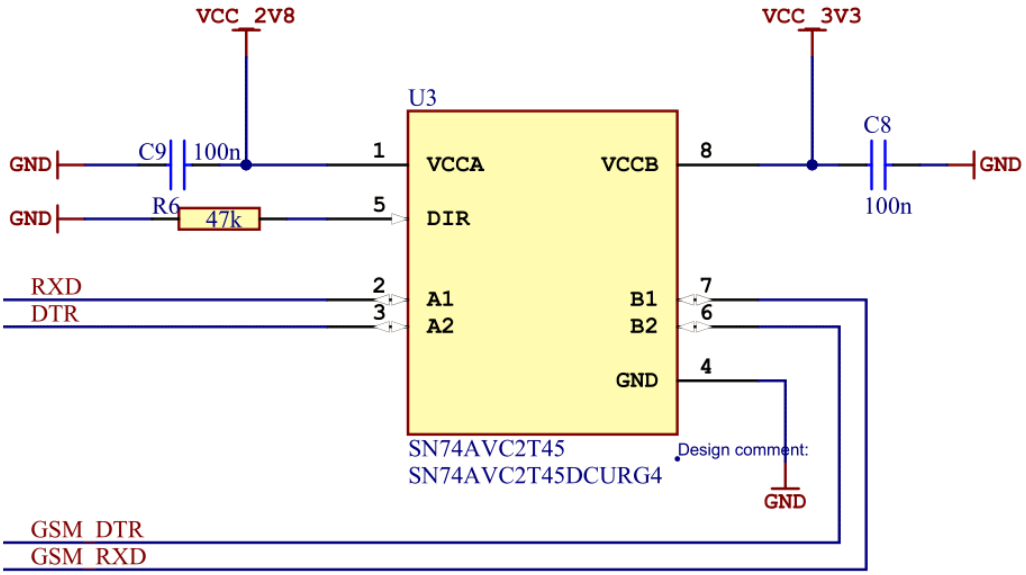
The SIM900 GSM module is connected to the SAM D20 microcontroller via its UART communication lines, with the optional keypad and screen control lines (for standalone operation of the GSM module) left unconnected.

2.7.1 Voltage Level Converters

While the main SAM D20 microcontroller runs from the 3.3V voltage rail, the SIM900 GSM module's digital interface has a maximum voltage input of only 3.1V. To avoid damage to the module, a level converter is used to

convert between the 2.8V rail (sourced from the GSM module's internal regulator) and the 3.3V rail (sourced from the main 3.3V regulator).

Figure 2-7. GSM Voltage Level Converter Schematic

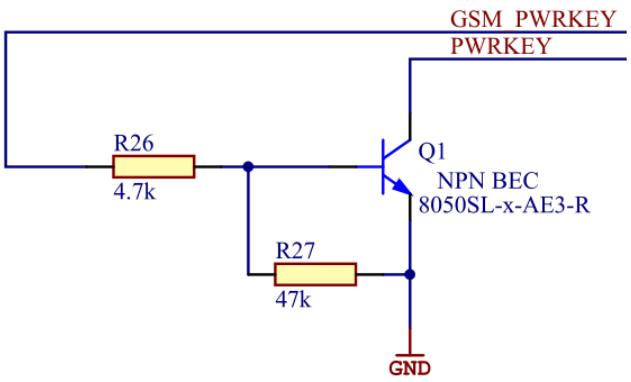


The level converters (SN74AVC2T45) are bi-directional, and convert the signals between the microcontroller and the GSM module in a transparent manner.

2.7.2 Power Key Driver

To start up the SIM900 module from power-off, the PWRKEY is pulled low via an external transistor driven by a GPIO of the SAM D20 microcontroller. This configuration is required instead of using another level converter IC because the 2.8V rail from the GSM module that powers the low side of the level converters is only present when the GSM module is already powered on.

Figure 2-8. GSM PWRKEY Driver Schematic

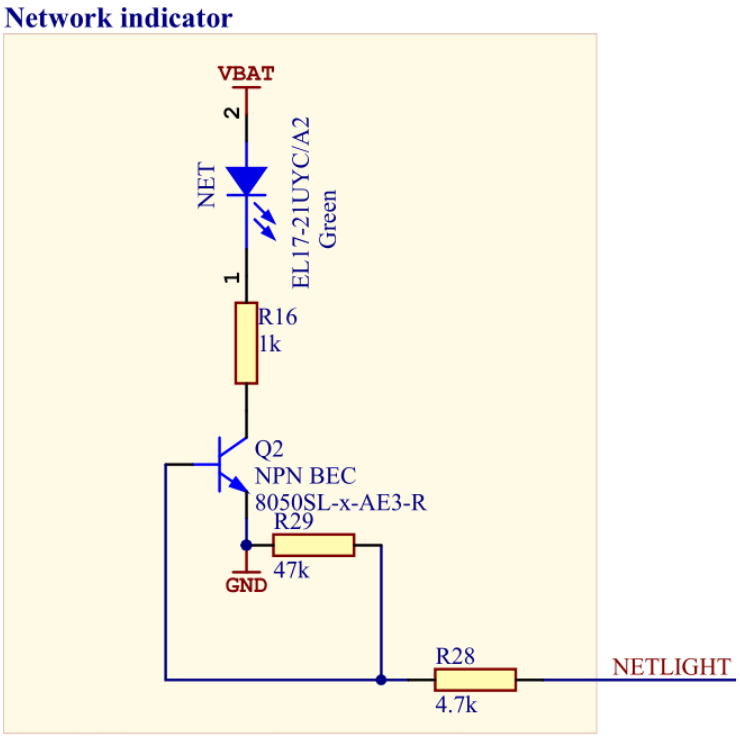


The use of the external transistor inverts the active logic level of the PWRKEY signal to the SAM D20, making it active-high instead of active-low.

2.7.3 Network Status Indicator

For debugging purposes, the NETLIGHT network status indicator pin of the GSM is connected to a dedicated status LED. This gives an easy indicator of when the GSM module is correctly powered up.

Figure 2-9. GSM Network Light Schematic

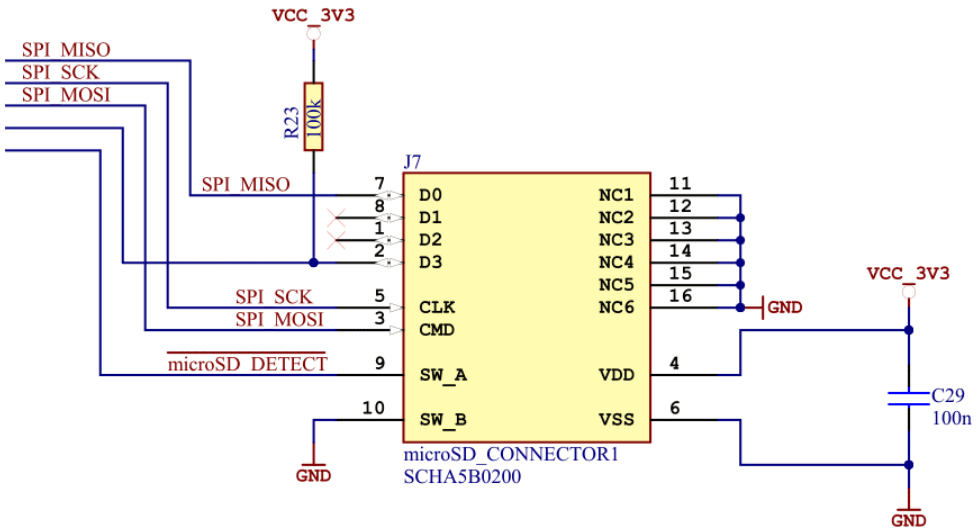


The NETLIGHT status LED is driven via an external transistor, as per the recommended circuit schematic given in Section "4.12 Network Status Indication" of the SIM900 hardware design guide from SimCom.

2.8 SD Card

To simplify the software in light of the low bandwidth requirements of the application, the SPI interface of the SD Card socket was connected to the SAM D20 instead of the more complicated 4-bit parallel bus.

Figure 2-10. SD Card Schematic

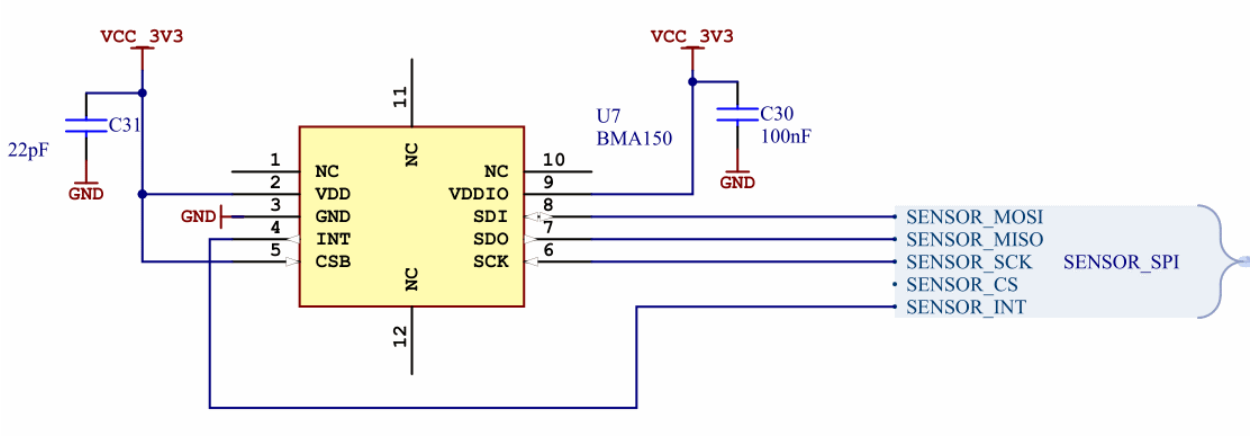


Card insertion and removal detection is performed via the SW_A and SW_B pins of the socket, which are shorted together when a card is present. The SW_A pin was routed to an external interrupt pin of the SAM D20 while the SW_B pin was tied to GND, making the detection signal active low in conjunction with the internal pull-up resistor of the SAM D20 microcontroller.

2.9 Accelerometer

The BMA150 accelerometer is wired in its I^2C interface mode configuration, rather than the alternative 3-wire or 4-wire SPI modes to reduce the number of SAM D20 I/O pins required to communicate with the sensor.

Figure 2-11. Accelerometer Schematic

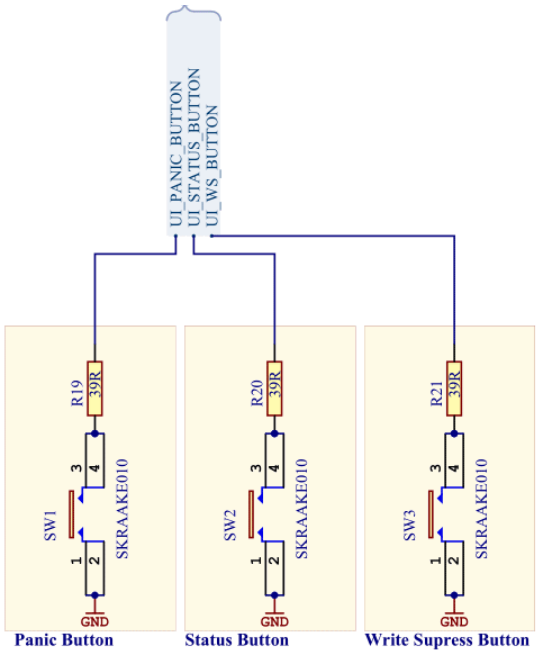


The INT configurable event digital interrupt pin from the sensor is routed to an external interrupt pin of the SAM D20, so that the latter can be woken up from sleep mode asynchronously when motion events are detected.

2.10 User Interface Buttons

As the SAM D20 microcontroller contains configurable internal pull-up resistors, board Status, and Write Suppression buttons are wired to simply ground the GPIO pins of the SAM D20 when pressed.

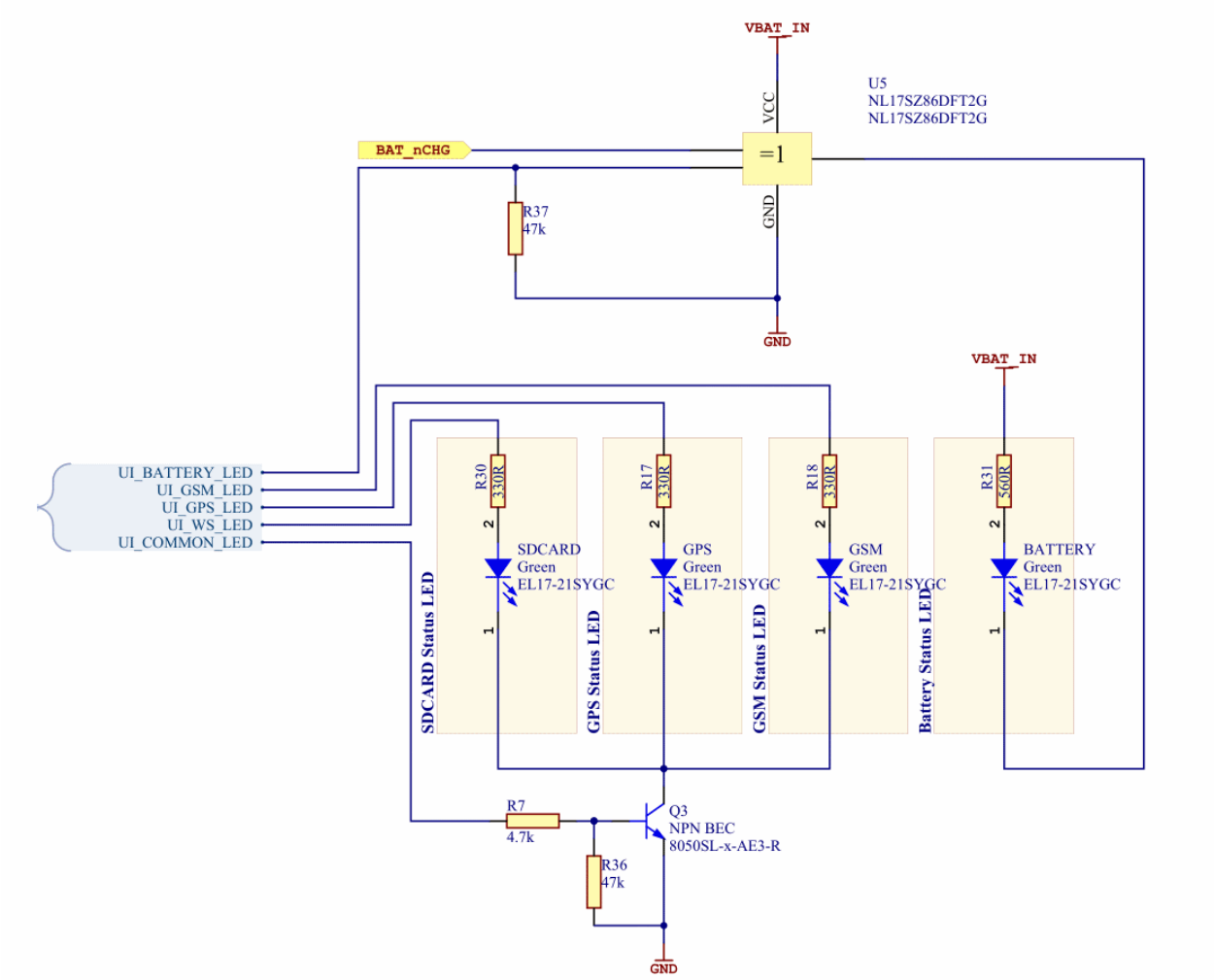
Figure 2-12. UI Buttons Schematic



2.11 Status LEDs

The three module GPS, GSM, and SD Card status LEDs are wired in an active-high configuration, with their cathodes tried to a common transistor driver pin modulated by a SAM D20 GPIO pin. This allows the on/off states of the LEDs to be controlled individually, whilst also allowing the SAM D20 to adjust the status LED brightness as a group using a single PWM output.

Figure 2-13. Status LEDs Schematic



The fourth LED indicates the battery charge state and is wired in an active-low configuration, fed from a hardware XOR logic gate. This gate has its logic inputs sourced from the /CHG charge status pin of the battery charger as well as a GPIO pin of the SAM D20, so that it can indicate both the charger hardware state and the battery voltage measurement state at the same time.

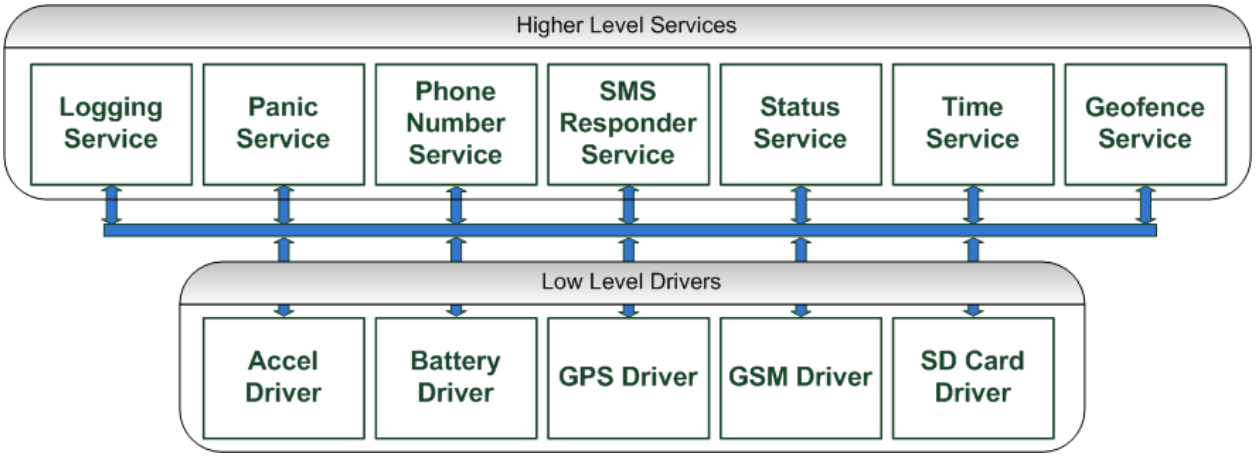
The GPIO pin of the SAM D20 effectively acts as an inverter of the battery LED state when set high, and acts as a pass-through when low. This allows the firmware to toggle the battery LED state when required by feeding a square-wave into the XOR gate input, while allowing the battery charger to turn on the LED when charging even if the SAM D20 is powered off due the the main 3.3V regulator failing from an insufficient input voltage from the battery.

Battery Charger /CHG	SAM D20 GPIO Pin	Battery LED State
0 (Charging)	0 (Pass-through)	0 (On)
1 (Not Charging)	0 (Pass-through)	1 (Off)
0 (Charging)	1 (Invert)	1 (Off)
1 (Not Charging)	1 (Invert)	0 (On)

3. Software Design

The system firmware is divided into a number of lower level hardware module drivers, and higher level services. Both the module drivers and services are FreeRTOS based, and can reference each other as needed in a thread-safe manner.

Figure 3-1. High Level Software Structure

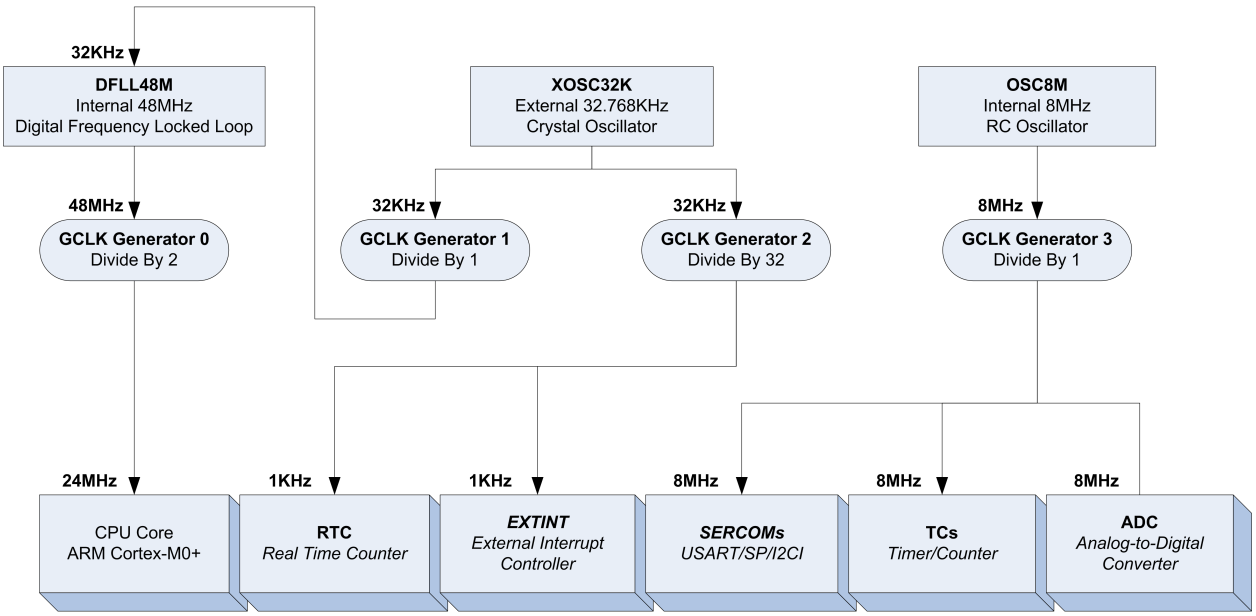


3.1 Clock Tree

The SAM D20 contains a rich internal clock tree, capable of generating a wide variety of asynchronous clocks for various internal modules as-needed to optimize accuracy, power, and other characteristics.

The clock tree configuration for the GPS Tracker application is shown below.

Figure 3-2. GPS Tracker SAM D20 Internal Clock Tree



As the full processing power capabilities of the SAM D20's CPU core are not required for the GPS Tracker application, it is fed from the internal 48MHz Digital Frequency Locked Loop module via a division factor of 2, giving a 24MHz core CPU clock. The DFLL output is a closed-loop multiplication of the external 32kHz watch crystal, to give a faster lock time and better output accuracy.

The External Interrupt Controller (EIC) and Real Time Clock (RTC) modules share a common generic clock generator with a 1kHz output prescaled from the external 32kHz watch crystal (for low power, high accuracy operation).

The remaining SERCOM SPI/USART/I^{2}C communication modules, Analog-to-Digital Converter module and hardware timer modules share a common 8MHz clock, sourced from the internal 8MHz RC oscillator (for high-performance, low latency operation). Within these modules, internal prescalers further divide down the digital interface clock to the speeds required for the set baud rates, conversion rates, and counting frequencies.

3.2 Atmel Software Framework

The Atmel® Software Framework (ASF) is a collection of free embedded software for Atmel microcontroller devices. It simplifies the usage of Atmel products, providing an abstraction to the hardware and high-value middleware. For more information, see http://www.atmel.com/asf.

The GPS Tracker projects makes use of ASF for drivers and services to control the underlying SAM D20 hardware peripherals, to speed up development and reduce the amount of application-specific code required for the reference design.

3.3 Real Time Operating System

Using a Real Time Operating System (RTOS) for the application reduces software complexity, increases code compartmentalization, and lowers the overall current consumption as the scheduler can more efficiently decide when and how long to enter a low power sleep mode. To further reduce power consumption, the FreeRTOS "Tickless Idle" feature is used.

3.3.1 FreeRTOS

FreeRTOS is a popular, portable, open source, royalty free, mini Real Time Kernel. For complete information on FreeRTOS, see http://www.freertos.org.

3.3.2 Tickless Idle

The tickless feature of FreeRTOS allows the device to sleep for periods longer than the predefined OS tick rate. This is useful when no tasks are ready to execute for more than one OS tick. The benefit of this is lower power consumption.

The tickless feature is implemented using a timer, configured with the desired timeout value, to wake the device. The same timer is also used to generate the system tick, so that time is kept in the optimal way, eliminating drift in most cases. If some other source wakes up the device before the sleep period is complete, but after more than one OS tick, there will be a slight drift, as the timers count value must be corrected.

3.4 Accelerometer Driver

The accelerometer driver is implemented via a simple interrupt based system; the Bosch BMA150 (connected over a I^{2}C link to the SAM D20) is first set up to generate a latched interrupt on its dedicated INT pin when movement is detected, and the SAM D20 External Interrupt (EXTINT) module is configured to sense the interrupt line changes.

Inside the movement detection callback the sensor interrupt line is reset, and the registered function for movement detection is called. The function to call when movement is sensed is registered by accel_configure_movement_detect() from the external driver or service.

3.5 Battery Management Driver

The battery management driver uses the ADC inside the SAM D20 to sense the current battery voltage periodically. To reduce the system cost, a 0.5x gain external resistive divider is used to halve the battery voltage into the SAM D20. The internal 0.5x gain mode of the ADC is selected to further reduce this by another factor of 2 for comparison against the internal 1V band-gap reference. The resulting VBAT/4 voltage signal is thus sampled via the ADC each time a FreeRTOS software timer expires.

This scheme allows the SAM D20 to measure the battery voltage from ~0.0V to ~4.0V with a high degree of accuracy and stability (as 16x oversampling is enabled in the ADC driver) to detect when a low battery event

occurs. Battery voltages between 4.0V and 4.2V cannot be precisely measured by the ADC, but will not damage the device. However, the lower battery voltages (the range of interest) are within the conversion range of the ADC.

When a battery level acquisition is complete, the ADC callback function computes the real battery voltage and compares it to the configured low battery voltage level set in the application configuration header. The result is then pushed to the Status Service so that the UI can be updated appropriately.

The current battery status (OK or Low) can be queried by external drivers and services by calling battery_is_voltage_low(). A battery with a voltage of below 3.5V is considered low, and in need of recharging.

To reduce the current draw of the ADC module, it is disabled between each periodic battery sample.

3.6 GPS Driver

The GPS driver manages the reception and parsing of the current GPS time, date, and location data. Internally, it is implemented by wrapping the ASF NMEA parser, a module capable of decoding ASCII encoded NMEA 0183 messages, a global GPS message standard.

Raw GPS data is buffered from the attached module asynchronously, via a custom USART reception callback function. Received characters are placed into a FreeRTOS queue created when the driver is initialized.

Periodically, the GPS deferred processing task attempts to read a line of data from the GPS reception queue, before passing it to the NMEA parser for analysis. Valid decoded NMEA messages are then further processed by the driver to obtain and update the current time, date, and location depending on the message contents. Updates to the internal driver state are protected via semaphores to ensure data integrity.

Position data is returned from the NMEA parser as a number of degrees, minutes, and fractional minutes, with the fractional value scaled by a fixed `NMEA_FRACTIONAL_SCALER` multiplier to avoid the use of floating point values.

The location reported by the NMEA parser output (as a floating point number of degrees) is thus:

$$\begin{aligned} position_{degrees} = & \frac{NMEA_{\text{degrees}}}{1} + \\ & \frac{NMEA_{\text{minutes}}}{60} + \\ & \frac{NMEA_{\text{frac\_minutes}}}{60 \times \text{NMEA\_FRACTIONAL\_SCALER}} \end{aligned} \tag{3.1}$$

The GPS driver converts the parsed NMEA values into a single fixed number of degrees for both longitude and latitude using the above equation, with the result similarly scaled by the constant `GPS_POS_DEGREES_SCALER` to avoid floating point types. The converted output is then offered for consumption by other drivers and services.

The current position is tracked inside the driver, and each time an identical position is reported (i.e. the GPS is stationary) an internal counter is incremented. Once a set number of identical reports are received, the driver enables sleep mode on the GPS to reduce its current consumption. The Accelerometer Driver is then used to wake up the GPS when motion is detected to resume live updates and reset the sleep counter.

3.7 GSM Driver

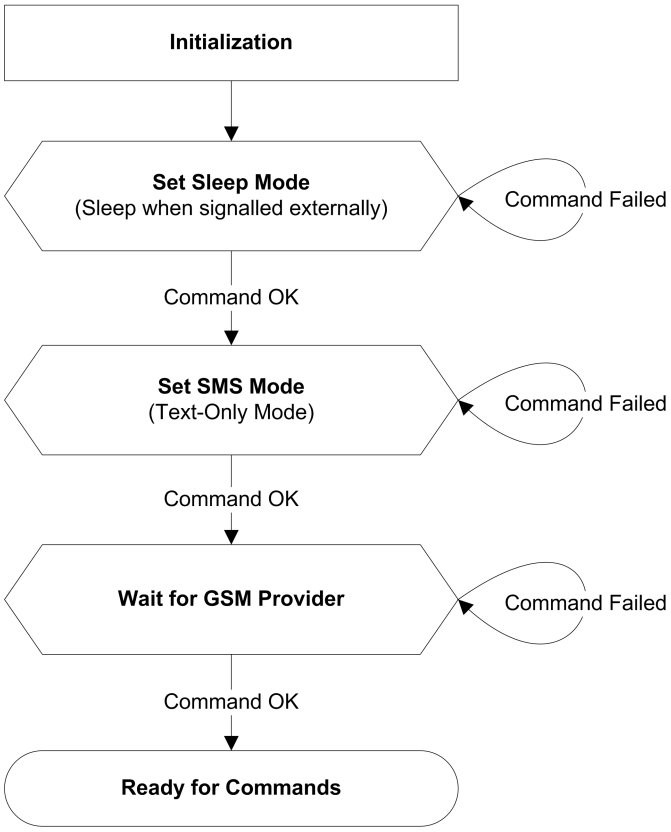
The GSM driver provides an interface to perform a limited set of actions (message receive, message send, message delete, etc.) to the attached GSM module. As GSM actions can take a long time to complete, the module APIs are designed to be asynchronous with callback functions triggered in response to updates as they become available.

When the driver is initialized, a software FreeRTOS timer is created to manage timeouts when communicating with the GSM module. In the event of a timeout while waiting for the module to respond, the driver will reset its internal state, abort the issued command, and attempt to re-initialize the driver.

After initialization, the GSM driver is ready to accept commands from other drivers and services in the system.

Initialization of the GSM module is performed via an internal state machine, as shown below. State transitions are only made when a command completes successfully; each initialization step is repeated until the module processes the command correctly.

Figure 3-3. GSM Initialization Flowchart



Raw GSM data is buffered from the attached module asynchronously, via a custom USART reception callback function. Received characters are placed into a FreeRTOS queue created when the driver is initialized.

Periodically, the GSM deferred processing task will attempt to read a response line of data from the queue, and parse it using the core AT command parser and the currently set internal command-specific parser. The core AT command parser checks for an error or command completion messages, while the command-specific parser examines the GSM responses for messages specific to the latest issued command. The parsed output is then further processed by the external module that requested the command, via the currently registered user callback function.

As a special case, the successful sending of a SMS message starts a timeout managed by an internal FreeRTOS software timer module. Until the timeout period expires, or the timeout is canceled prematurely via a call to gsm_force_sms_unlock(), no further SMS messages can be sent to avoid expensive carrier charges in the event of a software fault.

To conserve power when no commands are being issued, the GSM module is kept in a lower power sleep mode via an external pin controlled by the SAM D20. When in sleep mode, the GSM module is still able to receive incoming call and message requests.

3.8 SD Card Driver

The SD Card driver wraps the low level ASF SD Card driver, to link the raw block read/write functions to the third-party FATFS file system library (also available from ASF) in an OS-safe manner. This service also watches for SD card insertion and removal to automatically mount the FAT file-system of the inserted SD Card.

When initialized, the service creates a pair of semaphore objects; one to be used by the low level FATFS hooks to give exclusive access to the SD Card to a thread, and another to serve as a blocking object for the deferred card mount task. A pair of pins are configured as external interrupts using the SAM D20 EXTINT module to detect changes on the SD Card detection line, and the Write Suppression button.

When the write suppression button state changes, the new state is fed to the Status Service. No other processing of the button state is required, as the low level FATFS library hooks directly query the pin state as needed to determine if suppression is enabled or not.

When a card is inserted or removed, the callback function releases the semaphore to wake up the deferred SD Card mount task. This task then runs at the next scheduling opportunity to mount the file-system stored on the SD Card (if present), or unmount it (if removed) in the FATFS library.

The FATFS library hooks for thread-lock/release, raw disk block read/write, and time-stamp retrieval are hooked inside the SD Card driver in the various `sdcard_fatfs_*` functions, and linked to the appropriate system drivers and services. This allows the FATFS API functions to directly interact with the inserted SD Card's file system in a thread-safe manner.

3.9 Geofencing Service

The Geofencing service is responsible for the periodic checking of the current GPS location, and comparing it against the set bounding box coordinates read from a configuration file stored on the inserted SD Card (refer to Receiving Geofence Alerts).

When started, the service creates a new task to perform the location sampling. Each minute, the Geofence box coordinates are read from the SD Card and parsed to convert the raw ASCII text data of the file into a set of integer coordinates. These coordinates are then compared against the current GPS location to determine if the tracker has moved into or out of the boundary; if so, an alert is given.

In order to issue a Geofence alert, the service first obtains the current time from the Time Service, formats the event text the bounded area) into the appropriate message, and then sends the formatted message to each of the phone numbers listed on a file from the SD Card.

The Geofence service can be started or stopped externally via geofence_set_service_state(). When stopped, the Geofence service processing task is suspended to reduce the CPU load and allow the processor to remain in sleep mode for longer periods.

3.10 Logging Service

The Logging service is responsible for periodic querying of the current GPS time, date, and location information, for writing into a log file stored on the SD Card in Comma Separated Values (CSV) format.

When started, the service creates a new FreeRTOS FIFO queue to buffer the samples, and a periodic FreeRTOS software timer to add new samples to the buffer. This design allows the system to batch together multiple samples for batch writing to the SD Card, which lowers the time spent out of low power sleep mode in both the main processor and the SD Card.

Each time the sample timer expires the current GPS time, date, and location is queried; if the GPS has accurate data the values are stored into the buffer. The deferred Logging task will periodically examine the sample queue and if it contains data it will append the formatted entries to the log file on the SD Card.

The logging service can be started or stopped externally via logging_set_service_state(). When stopped, both the periodic sample timer and the Logging service deferred processing task is suspended to reduce the CPU load and allow the processor to remain in sleep mode for longer periods.

3.11 Panic Service

The Panic service is responsible for monitoring the panic button on the tracker, to detect when the user is experiencing a "panic situation". In response, the service will attempt to obtain the current GPS time and date, then broadcast it as a series of SMS messages to recipients read from the inserted SD Card's **PANICNUM.TXT** file (refer to Controlling the System Via SMS).

When started, the Panic service sets up an external interrupt (EXTINT) pin to detect when the panic button is pressed. Hardware filtering is enabled to reduce system wake-ups in response to transients on the input pin. If the button is pressed, a callback function fires to start the deferred panic processing by releasing a semaphore the Panic task is blocked on. At the next scheduling opportunity, the Panic processing task is then run, which waits to see if the panic button is held for the minimum time before initiating the panic sequence.

Panic sequences consist of reading from the list of panic message recipients stored on the SD card, obtaining the current GPS time and date, formatting of the message contents into the buffer, then the transmission of the panic messages to the recipients in turn. Once the panic sequence is complete, the task is again blocked waiting for a new toggle of the panic button.

3.12 Phone Number Service

The Phone Number service gives other drivers and services a way to easily read phone numbers from files stored on the inserted SD Card. This service provides an abstraction from the raw flat text file on the disk, and gives a simple interface to read the next extracted phone number (with non-number characters stripped) or to check if a given phone number is contained in the file. This reduces code duplication and gives a central point where the phone number read logic can be altered.

When phone numbers are read from a file on disk, all characters other than '+' and the digits 0-9 are removed. The remaining elements are concatenated together into the returned buffer, giving the resulting phone number in a standard format. This allows the user to provide text labels to each entry as a memory aid to the user without impacting the software, as long as the label does not contain any characters that comprise a valid phone number.

For example, the entry:

```
Test User: +00 1234 5678
```

Would be correctly parsed as the number:

```
+0012345678
```

However, the entry:

```
Test User 1: +00 1234 5678
```

Would be incorrectly parsed as the number:

```
1+0012345678
```

As it contains a reserved character as part of the label. All phone numbers should be specified with their full country code included.

3.13 SMS Responder Service

The SMS responder service is uses the lower level GSM driver to read and process commands from remote users sent via SMS.

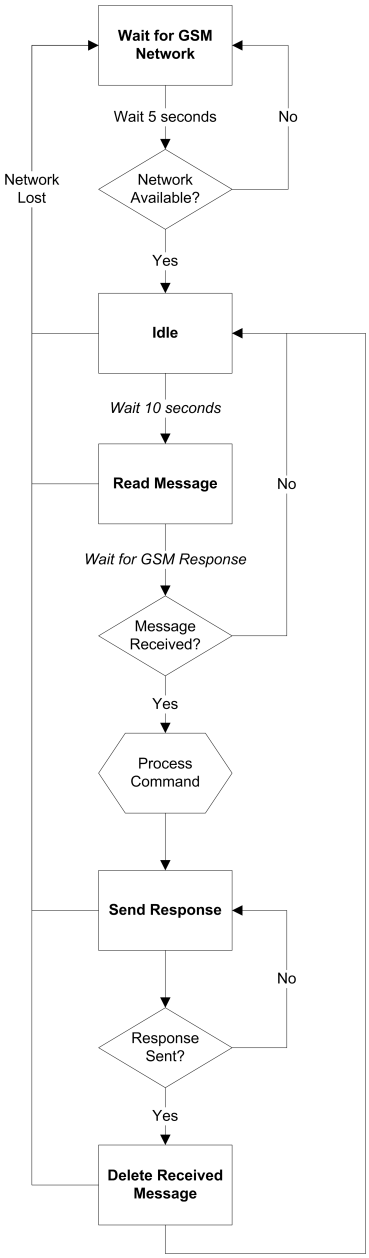
Periodically, the service will query the GSM driver for new messages. If a message is received, the SMS Responder task will process the message contents and sender phone number; if a SD Card is inserted, the sender's phone number is checked against the list of authorized users. If the phone number of the message sender is not in the list, an error message will be sent in response instead of processing the command.

Unknown commands will be silently ignored by the system, so that provider messages (such as automated messages on account balance) do not trigger a message loop.

For authorized users issuing known commands, the system will take any appropriate action(s), querying, and signaling other drivers and services as needed to perform the requested command and generate the appropriate response.

Internally, the service uses callback functions and a state machine to manage the various services states for the reading and processing messages in an asynchronous manner, as shown below.

Figure 3-4. SMS Responder Service Flowchart



3.14 Status Service

The Status service collates module states from all the drivers and services and updates the user interface (UI) - the board LEDs - as required. The Status service is a push type service; other modules signal changes of status to the Status service via the `status_notify()` function, rather than having the Status service query all other modules repeatedly to gather the system state. This design reduces the amount of logic required in other modules to store and retrieve the various module states, reducing the complexity of the system as a whole.

When a module signals a status change, the Status service stores the new state into an internal state vector. The state vector is protected by a critical section (section of code with interrupts globally suppressed) to allow other modules to call the `status_notify()` function from all contexts safely. A series of tests are performed on the new state values to determine the new board UI LED states (on, flashing, or off), before the deferred processing task is woken via a semaphore.

When woken by the released semaphore, the deferred processing task checks the current status button state and the current system state vector to determine if the status LEDs should be visible or not. If a visibility change is

required, the task will fade in/out the status LEDs via the common LED anode using an 8-bit hardware timer PWM output pin of the SAM D20.

The status button is configured when the service is initialized, via an External Interrupt (EXTINT) pin. Hardware filtering is enabled to reduce system wake-ups in response to transients on the input pin. If the button is pressed, a callback function fires to start the deferred processing task by releasing the same semaphore as the notify function that the Status task is blocked by.

Flashing of the UI LEDs that require it is performed by a periodic FreeRTOS software timer that is configured and started when the service is initialized.

3.15 Time Service

The Time service is responsible for tracking accurate time and date information for time-stamping within the application. Periodically, the Time service will request the current time and date from the GPS (a source of accurate time and date information from the GPS satellites when locked) and will update its internal state with the new time to ensure it is kept in sync.

The set Universal Coordinated Time (UTC) time and date is stored into the Real Time Clock (RTC) module of the SAM D20, running in Calendar mode. The RTC is clocked from the external 32.768kHz watch crystal to ensure minimal drift between periodic updates. By using the calendar mode of the SAM D20 RTC, leap year correction and time/date rollover is handled automatically by the hardware to lower the service's CPU usage.

4. Hardware Setup

This chapter describes the hardware setup and physical configurations of the *GPS Tracker Xplained*.

4.1 Board Configurations

The *GPS Tracker Xplained* can be used in either standalone or Xplained Pro wing configuration, depending on the components mounted. Full device schematics for both configurations are supplied with the project.

Warning The *GPS Tracker Xplained* should only be used in the hardware configuration set via the mounted components; do not connect a board configured for Standalone operation to a *SAM D20 Xplained Pro*.

4.1.1 Wing Configuration

In the *Xplained Pro Wing* configuration, the *GPS Tracker Xplained* has no on-board processor or related components mounted, and is designed to be attached to a *SAM D20 Xplained Pro* board on the EXT1 and EXT2 headers.

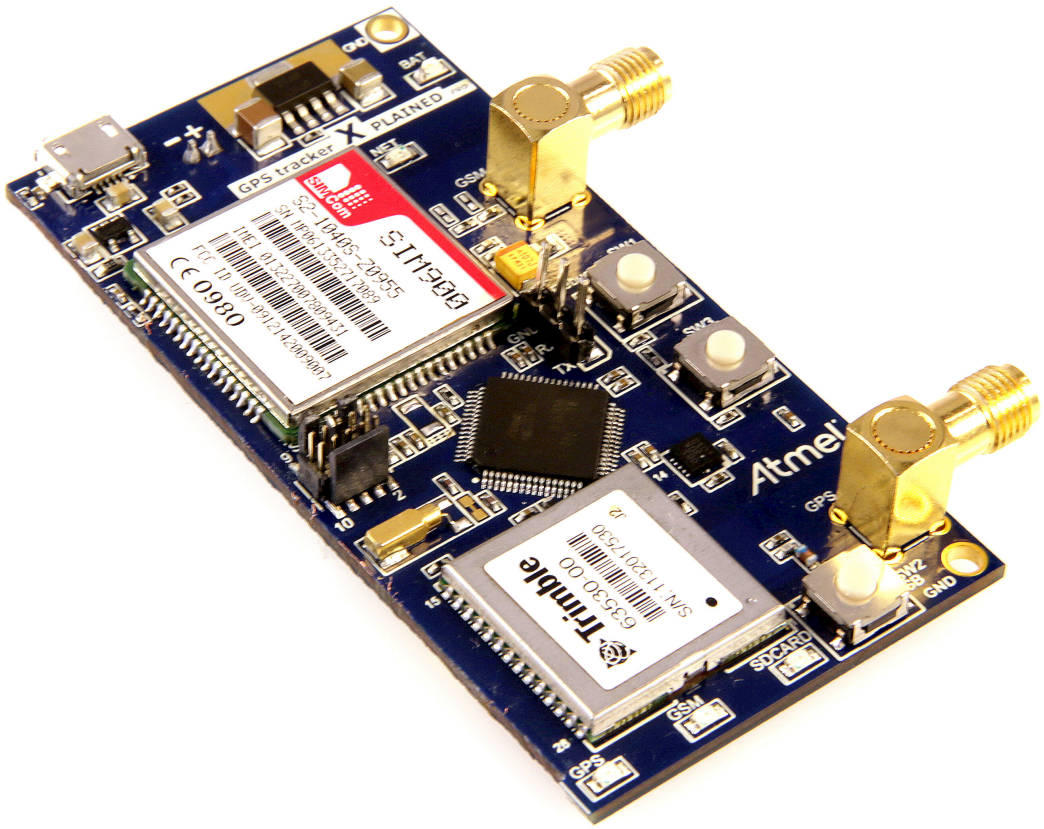
Note The *GPS Tracker Xplained* is intended to function as a wing for the *SAM D20 Xplained Pro* only; other *Xplained Pro* series boards are not compatible.

Figure 4-1. GPS Tracker Xplained in Xplained Pro Wing Configuration

4.1.2 Standalone Configuration

In standalone configuration, the *GPS Tracker Xplained* has an on-board SAM D20 processor, SWD debug connector, status button, 32.768kHz crystal, and several discrete components mounted. In this configuration, the firmware can be loaded directly onto the on-board processor and the product evaluated as-is. The unused Xplained Pro headers can be either not-mounted or the entire connector section removed at the V-scored cut in the PCB.

Figure 4-2. GPS Tracker Xplained in Standalone Configuration



4.2 Board Setup

To use the *GPS Tracker Xplained*, connect a Lithium Ion battery (3.7V nominal) to the battery connector. A USB cable may be inserted into the *GPS Tracker Xplained* to charge the battery while the system is running.

Note A battery is required for both Standalone and Wing modes, due to the high peak current requirements of the GSM and GPS modules.

When used in wing mode, a USB cable must be plugged into the *SAM D20 Xplained Pro* to correctly power the main processor. If not inserted, the application will not run correctly.

GPS and GSM antennas should be attached to the two connectors on the board and placed with a clear line of sight to the sky to ensure best possible signal reception.

A full-sized SIM card should be inserted into the SIM receptacle with the board unpowered.

Note The SIM card used in the tracker should be unlocked, and should not require a PIN authentication code to send or receive messages.

4.3 Pin Connections to the SAM D20

Connections between the board modules and the on-board or external SAM D20 microcontroller are described below. For boards configured to work as a wing extension to a *SAM D20 Xplained Pro*, the equivalent extension header pin-outs are also shown.

4.3.1 Accelerometer

The connections to the BMA150 module are as follows:

Accelerometer Pin	SAM D20 Pin	SAM D20 Xplained Pro
INT	PB14	EXT2 pin 9
SDA	PA08	EXT2 pin 11
SCL	PA09	EXT2 pin 12

4.3.2 Battery Interface

The connections to the battery monitor circuits are as follows:

Battery Signal	SAM D20 Pin	SAM D20 Xplained Pro
Battery VBAT/2 Voltage	PA11	EXT2 pin 4
Battery Charger /CHG	PB04	EXT1 pin 9

4.3.3 GPS Module

The connections to the GPS module are as follows:

GPS Module Pin	SAM D20 Pin	SAM D20 Xplained Pro
TXD-B	PB13	EXT2 pin 14
RXD-B	PB12	EXT2 pin 13

4.3.4 GSM Module

The connections to the GSM module are as follows:

GSM Module Pin	SAM D20 Pin	SAM D20 Xplained Pro
TXD	PA19	EXT2 pin 18
RXD	PA18	EXT2 pin 16
DTR	PA17	EXT2 pin 15
PWRKEY	PA16	EXT2 pin 17
STATUS	PB02	EXT2 pin 7

4.3.5 SD Card Module

The connections to the SD Card are as follows:

SD Card Connector Pin	SAM D20 Pin	SAM D20 Xplained Pro
/SS	PA05	EXT1 pin 15
MOSI	PA06	EXT1 pin 16
MISO	PA04	EXT1 pin 17

SD Card Connector Pin	SAM D20 Pin	SAM D20 Xplained Pro
SCK	PA07	EXT1 pin 18
CARDDETECT	PB05	EXT1 pin 10

4.3.6 User Interface

The connections to the UI buttons are as follows:

Button	SAM D20 Pin	SAM D20 Xplained Pro
Status Button	PA15	SW0
Write Suppress Button	PB06	EXT1 pin 5
Panic Button	PA23	EXT2 pin 8

The connections to the UI LEDs are as follows:

LED	SAM D20 Pin	SAM D20 Xplained Pro
Battery LED	PA21	EXT2 pin 6
GPS LED	PB15	EXT1 pin 10
GSM LED	PA20	EXT2 pin 5
SD Card LED	PA10	EXT2 pin 3
Brightness Control	PA22	EXT2 pin 7

5. Software Setup

5.1 Setting the correct SIM card PIN

SIM cards are often protected by a four digit PIN code, this pin code can be set in the conf_applicaiton.h file under the define GSM_PIN.

WARNING Make sure to provide the correct pin code as most SIM cards has a limited amount of tries before requesting the PUK code, this software does not have support for entering a PUK code. The SIM card then needs to be removed and unlocked using an external device e.g. a mobile phone.

6. Application Usage

Once all the hardware is configured according to Hardware Setup, the code can be loaded into the device. On startup, the system will initialize all hardware and begin listening for UI events and new GPS or GSM data.

6.1 Debug Interface

The current system log can be obtained via the debug channel, listing the internal device software module activity.

When the application is run on a board configured for wing mode, the debug output is presented via the USB CDC interface of the *SAM D20 Xplained Pro* on-board Embedded Debugger. When in standalone mode, the 3.3V logic-level serial data is available on the dedicated debug header, as the USB connector of the *GPS Tracker Xplained* is for battery charging only.

To observe the debug output, open a COM port in a terminal emulator (such as *PuTTY*) with the settings of:

- 115200 baud
- 8 data bits
- 1 stop bit
- No parity

Sample debug output:

```
*- ATMEL SAM D20 GPS TRACKER -*
Initializing drivers
Initializing services
Starting scheduler
 SD Card > SD Card inserted, mounting
 SD Card > Initializing SD Card
 SD Card > SD Card initialized
     GPS > GPS time/date locked
     GPS > GPS position locked
     GSM > Found network provider
    Time > Time set via GPS: 2013-11-04 11:20:31 UTC
     GPS > GPS sleeping
 Logging > 2013-11-04,11:20:36,00148,63.36417,10.37168
 Logging > 2013-11-04,11:20:51,00148,63.36417,10.37168
 Logging > 2013-11-04,11:21:06,00148,63.36417,10.37168
 Logging > 2013-11-04,11:21:21,00148,63.36417,10.37168
```

6.2 Logging GPS Location to a SD Card

If a SD card with a valid FAT32 file system is inserted, the current GPS location will be periodically logged to a CSV file with the format **YYYYMMDD** (such as *20130813.CSV* for a log created on 13-08-2013). Log entires for the same day are appended to the same CSV file.

Inserted SD cards must be pre-formatted with a FAT32 file system before being used with the tracker. Both SD and SD-HC type SD cards are supported.

Warning If a write operation is in progress when the SD card is removed, the file system may become corrupt. To prevent this, always hold down the Write Suppress button and wait for all pending write operations to complete (SD Card status LED begins flashing) before removing the SD card.

6.2.1 CSV Format

Each log entry will contain the **Date, Time, Altitude, Latitude, and Longitude** with the format:

```
YYYY-MM-DD,HH:MM:SS,mmmmm,DD.ddddd,DD.ddddd
```

For example:

```
2013-11-04,11:23:21,00149,63.36415,10.37162
```

Where:

- **YYYY-MM-DD** is the current UTC date
- **HH:MM:SS** is the current UTC time
- **HH:MM:SS** is the current UTC time
- **mmmmm** is the current GPS altitude in an absolute number of meters
- **DD.ddddd** is the current GPS latitude/longitude as an integer and fractional number of degrees

6.2.2 GPX Format

The GPS tracker CSV log data can be converted into the open GPX standard format via a Python script, included in the accompanying software package of this Application Node. The GPX format is an open industry standard for GPS data which is supported in most GPS related applications, such as *Google Earth*™.

To convert a log file, run the following from a command prompt:

```
python tracker-gpx-converter.py {input}.csv {output}.gpx
```

This will require Python 2.7 to be installed, along with the Python lxml XML library.

6.3 Receiving Geofence Alerts

The GPS Tracker can be configured to send alert SMS messages to one or more recipients each time the tracker enters or exits a rectangular set of GPS coordinates. To configure the Geofence module, a **GEOFENCE.txt** file must be stored on the inserted SD Card. This file should contain a single line with the format:

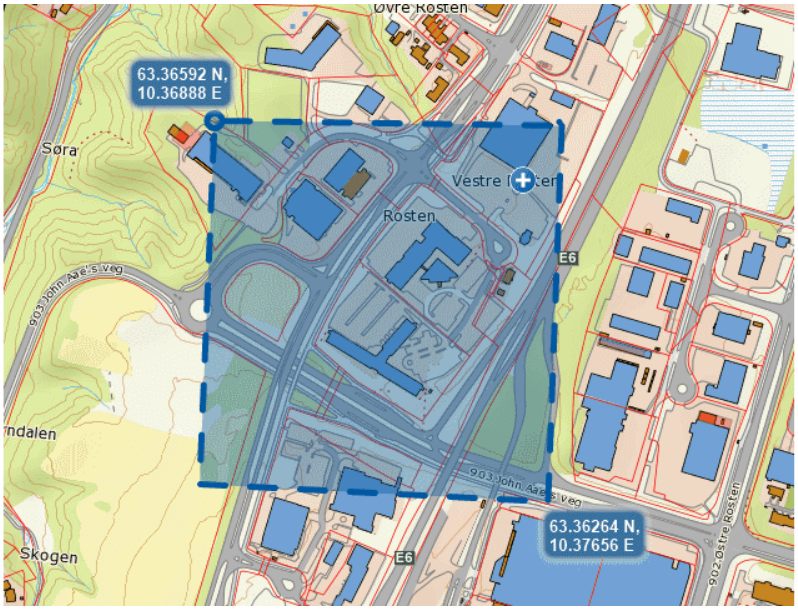
```
DD.ddddd,DD.ddddd DD.ddddd,DD.ddddd
```

For example:

```
63.36592,10.36888 63.2640,10.37656
```

To set the latitude and longitude of the top-left and bottom right coordinates of the Geofence bounding box.

Figure 6-1. Determining the bounding coordinates of a Geofence



Every minute, the Geofence service will check the current GPS location and compare it against the set of stored coordinates. If the tracker exits or enters the Geofence area, alert messages will be sent to the list of phone numbers read from the inserted SD Card, stored in a file named **GEONUM.TXT**. Each phone number should be placed on a separate line, and non-digit or non-'+' characters are ignored.

Note As non-digit and non-'+' characters are stripped from the file when it is read, contact names can be placed before or after each recipient number as long as they contain non-phone number characters only.

6.4 Controlling the System Via SMS

When a GSM modem is connected with a valid SIM card, the system can be controlled via SMS. Control messages must start with a valid command (listed below). The commands are not case-sensitive.

Note Only one command may be processed at a time; if a second command is received before the first is processed, it will be ignored.

The system will only accept commands issued by a remote number listed in the inserted SD Card, stored in a file named **RESPNUM.TXT**. Each phone number that is allowed to issue commands should be placed on a separate line, and non-digit or non-'+' characters are ignored. If the authorization list contains no valid entries, or if no card is inserted, all remote numbers will be allowed to issue commands to the system.

Note As non-digit and non-'+' characters are stripped from the file when it is read, contact names can be placed before or after each recipient number as long as they contain non-phone number characters only.

Commands are prefixed with a '*' character to ensure that they are not accidentally triggered by automated provider messages (such as account balance information) sent to the device.

6.4.1 *LOCATION Command

The "location" command returns the current device location, as a link to a web service such as Google Maps™.

6.4.2 *STATUS Command

The "status" command returns a summary of the device status, including the GPS lock state, the current GPS location, current system time, SD card state, and other relevant information.

6.4.3 *STARTLOG Command

The "startlog" command resumes logging of the current location to the SD card, if one is inserted.

6.4.4 *STOPLOG Command

The "stoplog" command pauses logging of the current location to the SD card, if one is inserted.

6.4.5 *STARTFENCE Command

The "startfence" command resumes comparing of the current location to the Geofence coordinates stored on the SD card, if one is inserted.

6.4.6 *STOPFENCE Command

The "stopfence" command pauses comparisons of the current location to the Geofence coordinates stored on the SD card, if one is inserted.

6.5 Panic Button

When held for longer than the minimum duration, the panic button will attempt to read a list of phone numbers from the inserted SD Card, stored in a file named **PANICNUM.TXT**. Each phone number should be placed on a separate line, and non-digit or non-'+' characters are ignored.

Note As non-digit and non-'+' characters are stripped from the file when it is read, contact names can be placed before or after each recipient number as long as they contain non-phone number characters only.

When activated, a text message will be sent to each recipient in the panic number list, with the current tracker time and location.

6.6 Write Suppression Button

To prevent corruption of the SD Card during removal, the Write Suppress button should be activated. When enabled, this prevents the firmware from writing to the card (beyond the current operation) so that it may be safely removed.

6.7 Status Button

While pressed, the status button will display the system status on the status LEDs, turning them off after the status button is released. If a charger is connected, the status LEDs will remain visible at all times.

There are four system status LEDs:

- **SD Card:** Indicates if a SD Card is mounted with a valid file system *(on)*, card has an unrecognized file system or write-suppression is active *(flashing)*, or no card inserted *(off)*
- **GSM:** Indicates if the GSM modem has established a connection to a network provider *(on)*, waiting for provider *(flashing)*, or initializing *(off)*
- **GPS:** Indicates if the GPS has both a valid time/date and location lock *(on)*, partial lock *(flashing)*, or no lock *(off)*
- **Battery:** Indicates if the battery is currently charging *(on)* or the voltage is low *(flashing)*

During a panic sequence, the status LEDs will be always visible to show the current system state, until the panic sequence has completed.

7. Abbreviations

The table below presents the abbreviations used in this application note:

Abbreviation	Description
ADC	Analog-to-Digital Converter
ASCII	American Standard Code for Information Interchange
ASF	Atmel Software Framework
CDC	Communication Device Class
COM	Serial port interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
EIC	External Interrupt Controller
EXTINT	External Interrupt
FATFS	File Allocation Table Filing System
FIFO	First In First Out
GCC	GNU Compiler Collection
GNU	GNU's Not Unix
GND	Ground
GPS	Global Position System
GPIO	General Purpose In-Out
GPX	GPS Exchange
GSM	Global System for Mobile
I^2C	Inter-Integrated Circuit
LED	Light-Emitting Diode
LDO	Low Drop-Out
NMEA	National Marine Electronics Association
OEM	Original Equipment Manufacturer
PC	Personal Computer
PWM	Pulse-Width Modulation
RTC	Real-Time Counter
RTOS	Real-Time Operating System
SD	Secure Digital
SIM	Subscriber Identity Module
SMS	Short Message Service
SPI	Serial Peripheral Interface
UI	User Interface
USART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UTC	Universal Coordinated Time
XOR	Exclusive Or

8. Revision History

Doc. Rev.	Date	Comments
42212D	08/2014	Added information about how to set the correct pin code for the SIM card
42212C	06/2014	• Corrected information about kit availability, kit not available through Atmel store or distributors at the moment • Fixed document typos
42212B	01/2014	Added figure 5-1
42212A	12/2013	Initial release

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