The Freefall Manual

Stingworks

September 16, 2010

Stingworks reserves the right to make changes without further notice to this document and/or to the product to improve reliability, function, or design. Stingworks does not assume any liability arising out of the application or use of any product or application described herein.

Stress above one or more of the limiting values specified in this document or in the referenced component datasheets may cause permanent damage to the product or a component of the product. Exposure to limiting values for extended periods may affect reliability.

This product is not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Stingworks customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Stingworks for any damages resulting from such improper use or sale.

Contents

Abbreviations and Acronyms

A = Ampere(s)
API = Application Programming Interface
CDS = CaDmium Sulfide light sensor
GND = electrical GrouND
HID = Human Input Device
IDE = Integrated Development Environment
ISP = In-System Programming
LED = Light Emitting Diode
MiRF = Miniature Radio Frequency module
MISO = Master-In Slave-Out
MMA = MicroMachined Accelerometer
MOSI = Master-Out Slave-In
NiMH = Nickel Metal Hydride
PCB = Printed Circuit Board
PDIP = Parallel Dual Inline Package
PMAN = Power MANagement module
RAM = Random Access Memory
RS232 = Recommended Standard 232 (legacy serial protocol)
RX = Receive
SCK = SPI Clock
SPI = Serial Peripheral Interface bus
TX = Transmit
USB = Universal Serial Bus
V = Volt(s)

Chapter 1

Freefall Basics

1.1 Overview

The Freefall is a unique union of hardware, firmware, and software enabling rapid prototyping of battery-powered accelerometer-based applications. It allows the developer to quickly move from concept to application, whether or not the breadboard is an intermediate step.

This document provides detailed information on the hardware, software, and firmware components of Freefall and should be treated as a reference. Throughout, it is assumed that the reader has some experience programming for embedded systems. For more general information, consult a book on microcontroller system development basics. For details on the functioning of a subcomponent of the Freefall such as the microcontroller or accelerometer, consult the manufacturers' datasheets, enumerated at the end of this document.

For frequently asked questions, source code, and the latest version of this document visit the Stingworks website at `http://www.stingworks.com/`.

1.1.1 Hardware

In the hardware dimension, the Freefall is a 24-pin, 900mil-wide parallel dual inline package (PDIP) device designed for use in a breadboard or directly in an embedded device. The printed circuit board (PCB) contains the following major hardware components:

- The Freefall has at its core an **ATMEL AVR ATmega168 microcontroller**. This AVR's crucial features are:
  - 8-bit Reduced Instruction Set (RISC) core

  - clocked with a 12MHz quartz crystal
  - 16 kilobytes of built-in flash program memory
  - 1024 bytes of volatile memory (SRAM)
  - 512 bytes of flash data memory (EEPROM)
  - programmable in assembler or C
  - high-quality, free, and open-source development tools

- For acceleration sensing, the **Freescale MMA7455 micromachined accelerometer** is employed:
  - three-axis, simultaneously read
  - ±8g range
  - 10-bit resolution
  - 250Hz sampling rate (125Hz bandwidth)
  - suitable for tilt or dynamic acceleration sensing

- The Freefall boasts a built-in **battery recharing module** with the following capabilities:
  - can charge 3xAAA Nickel Metal Hydride (NiMH) cells from USB
  - charge rate adjustable at run-time
  - can recharge while running
  - batteries can be omitted for strictly bus-powered applications

- There is **no Serial-to-USB converter**, unlike most AVR development systems. Instead, a firmware-only USB driver is used. The only USB-related hardware is a few impedance-matching resistors and the Mini-B connector.

- While there is no wireless capability built-in to the Freefall, wireless may be easily added by way of a low-cost add-on module based on the **Nordic Semiconductor nRF24L01+**. This module provides robust wireless communication on the 2.4GHz ISM band at up to 2Mbps. All pins required for communication with the wireless module are lined-up, and example code is provided.

1.1.2 Software

On the PC software side, a powerful open-source toolchain is at your fingertips. The basic workflow is:

write your `C`-code in your favorite editor
→ compile the code with `AVR-GCC`
→ program the flash with `AVRDUDE`
→ get live feedback through `SteamingPot`.

For more information on how to write code for the Freefall, see the Stingworks website.

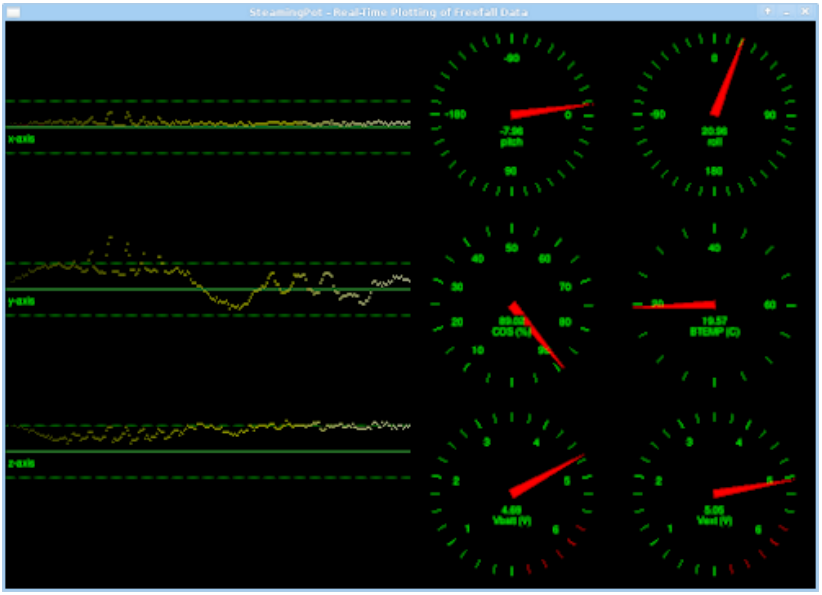

Figure 1.1: Example PC-side software for live feedback from the Freefall. This software and its source-code are provided on the Stingworks website.

Alternatively, the Arduino language and development environment may be used to develop for the Freefall. However, the example code that comes bundled with Freefall requires modification to run in the Arduino IDE, with the exception of the `Blink` arduino example. See Section 2.9 for details.

1.1.3 Firmware

The Freefall ships with a USB bootloader programmed into the special 2K bootloader of the AVR's program memory. This allows reprogramming the firmware without an in-circuit programmer (ISP).

Furthermore, the Freefall comes with a library of functions and drivers for immediate access to the hardware on the board, e.g.:

- Serial Peripheral Interface (SPI) driver
- Pulse-Width Modulation (PWM) driver
- Analog-to-Digital Conversion (ADC) driver
- MMA7455 accelerometer driver
- Nordic nRF24L01+ 2.4GHz Wireless driver

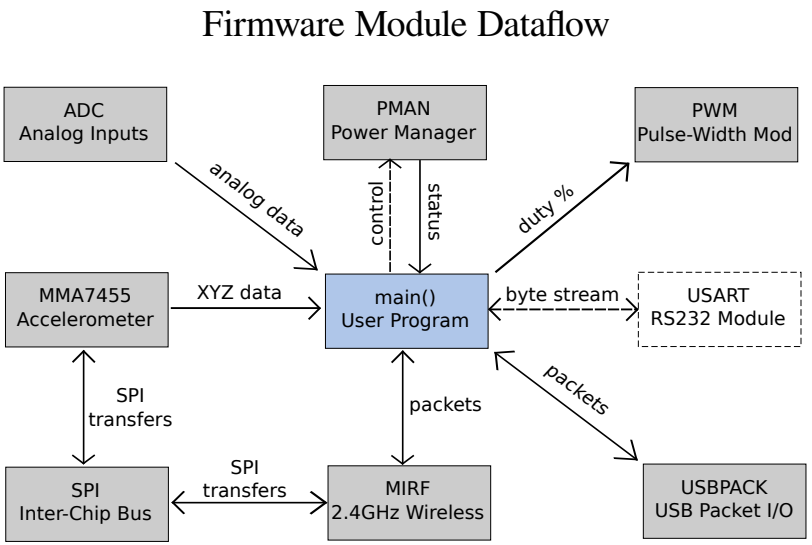

Figure 1.2: Interaction between the several code modules provided with the Freefall.

For examples of how to write firmware applications to run on the Freefall, see the `applications` directory of the Freefall software bundle or in the Source Code chapter of this document. However, the basic structure of a firmware program is as follows:

```
#include "freefall.h"
void main() {
        freefall_init();
        /* insert additional initialization code here */
        while(1) {
                /* insert code to run periodically here */
                wdt_reset();
        }
}
```

1.1.4 Power Distribution

The Freefall is designed for versitile, low-power operation. Any combination of USB-bus power and battery power may be used. As a result, there are several different voltage levels simultaneously present on the Freefall PCB. The following diagram may help to clarify the relationship among these voltages, and how the power distribution works.

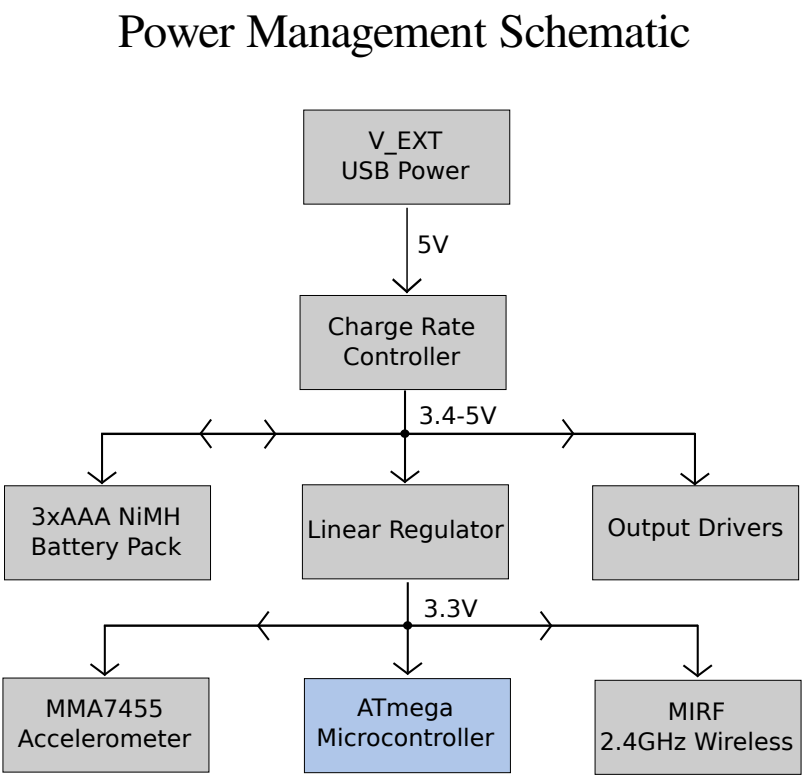

Figure 1.3: The various power levels and their interactions are visualized.

1.1.5 Pin Descriptions

The Freefall communicates with its environment by way of pins located along its sides. The following diagram gives a visual reference for pin assignments.

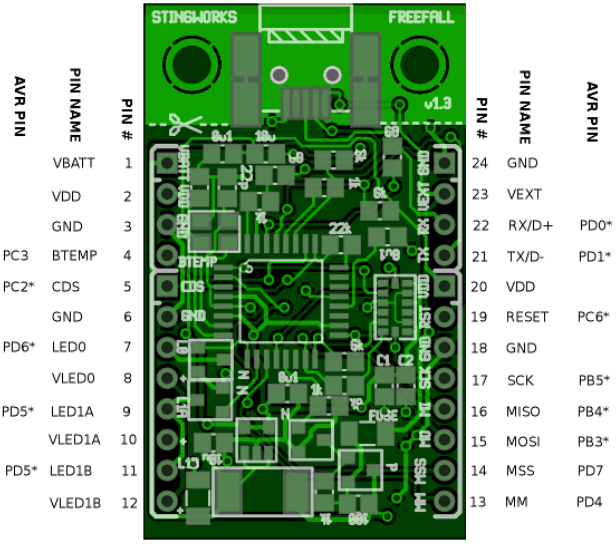

Figure 1.4: Pinout for Freefall. (*) indicates that the Freefall pin function is significantly different than the direct AVR pin function; see the detailed pin descriptions below.

In detail, the function of each pin is:

1. **VBATT - Battery Voltage Input**
   Nominally 3.6V power input from a 3xAAA Nickel Metal Hydride (NiMH) rechargeable battery pack. Voltage must be in the range of 3.0-5.0V.

2. **VDD - Regulated 3.3V Output to Battery Temperature Sensor** Powers temperature sensor in battery pack. Voltage will be 3.3V or VBATT-150mV, whichever is lower.

3. **GND - Common Ground to Battery Pack**
   Ground shared with all devices connected to the Freefall.

4. **BTEMP - Battery Temperature Analog Input**
   Analog input from battery temperature sensor. AVR Pin PC3.

5. **CDS - (CdS Light Sensor) Analog Input**
   An analog input with selectable 22k pullup/pulldown. Designed to be used with a Cadmium Sulfide (CdS) light sensor. See Section 2.7.2 for details. AVR Pin PC2.

6. **GND - Common Ground to CDS Sensor**
   Ground shared with all devices connected to the Freefall.

7. **LED0 - LED Driver Output 0**
   A high-current open-drain output selectable between High Impedance (HZ) and Low, with optional 8-bit PWM. Designed to be used to sink current from one or more LEDs or other peripherals. To be used in conjunction with VLED0. See Section 2.6 for details. AVR Pin PD6.

8. **VLED0 - Fused Current Source for LED0**
   Nominally 3.6V power output from battery pack. Protected by a resettable fuse (PTC-type) with 1.5 ampere hold current and 2.2 ampere trip current, shared with other VLED outputs. Designed to provide power to a small number of LEDs, relays, or other high-current peripherals.

9. **LED1a - LED Driver Output 1a**
   A high-current output selectable between High Impedance (HZ) and Low, with 8-bit PWM. Designed to be used to sink current from one or more LEDs. To be used in conjunction with VLED1a. See Section 2.6 for details. AVR Pin PD5.

10. **VLED1a - Fused Battery Voltage Output for LED1a**
    Electrically identical to VLED0.

11. **LED1b - LED Driver Output 1b**
    Electrically identical to LED1a.

12. **VLED1b - Fused Battery Voltage Output for LED1b**
    Electrically identical to VLED0.

13. **MM - "MiRF" Module Mode Select Output**
    Required for optional MiRF 2.4GHz wireless module. See Section 2.8 for details. In the absence of the MiRF, may be used as a general purpose input/output pin. AVR Pin PD4.

14. **MSS - "MiRF" Module Slave Select Output**
    Required for optional MiRF 2.4GHz wireless module. See Section 2.8 for details. In the absence of the MiRF, may be used as a general purpose input/output pin. AVR Pin PD7.

15. **MOSI - SPI Master-Out Slave-In (MOSI) Output**
Used to interface with programmer and other SPI peripherals. Available for general purpose input/output only if SPI (including accelerometer) is unused. AVR Pin PB3.

16. **MISO - SPI Master-In Slave-Out (MISO) Input**
Used to interface with programmer and other SPI peripherals. Available for general purpose input/output only if SPI (including accelerometer) is unused. AVR Pin PB4.

17. **SCK - SPI Clock Output**
Used to interface with programmer and other SPI peripherals. Available for general purpose input/output only if SPI (including accelerometer) is unused. AVR Pin PB5.

18. **GND - Common Ground to Programmer**
Ground shared with all devices connected to the Freefall.

19. **RST - AVR Reset Input**
Strongly pulling this pin to ground to overcome the internal 7k pullup causes the AVR to enter reset. Used to reset the device or to invoke bootloader for USB programming. Used by all SPI ISP programmers to initiate SPI programming session.

20. **VDD - Regulated 3.3V Output to Programmer**
Provides power to ISP programmer, if needed. Voltage will be 3.3V or VBATT-150mV, whichever is lower.

21. **TX - USART Transmit Output / USB D- Input/Output**
Operates as serial transmit (TX) pin in RS232 mode, or as USB Data- (D-) pin in USB mode. May also be used as general purpose input/output if neither USB nor RS232 are used. AVR Pin PD1.

22. **RX - USART Receive Input / USB D+ Input/Output**
Operates as serial receive (RX) pin in RS232 mode, or as USB Data+ (D+) pin in USB mode. May also be used as general purpose input/output if neither USB nor RS232 are used. AVR Pin PD0.

23. **VEXT - External Charger or USB Voltage Input**
Placing an external voltage of 5-9V on this pin allows the Freefall to operate while optionally charging a NiMH battery pack. This pin is protected by a resettable PTC-type fuse with 1.5 ampere hold current.

24. **GND - Common Ground to USB or Serial Port**
Ground shared with all devices connected to the Freefall.

1.2 Programming

We assume that the user is familiar with the `C` programming language. Given this, you need only to know how to compile C programs and load them onto run the Freefall. The included example programs all contain a file called `main.c` which holds the main C program. This file is combined with other code located in header files (.h) and other code files (.c) and compiled into a hex-file (.hex) which is an embedded binary. This hex-file is then loaded ("programmed") onto the Freefall using one of the three available programming methods. This process of compilation and programming is automated by the `make` program, which derives its instructions from the `Makefile`, located in each example directory. The user is encouraged to experiment with the example C files and Makefiles to get comfortable with this development paradigm.

We will now detail the three available programming methods.

1.2.1 USB Bootloading

The Freefall is shipped with a USB bootloader.[1] This software allows the Freefall to be reprogrammed from a PC over USB.

Hardware

The only required hardware is a standard USB cable. The correspondence between USB wires or connector pins and device pins are as follows:

USB wire/pin	Freefall pin
1 +5V (red)	23 VEXT
2 D+ (white)	21 TX
3 D- (green)	22 RX
4* GND (black)	24 GND

* Note that GND is pin 5 on a USB Mini-B connector.

Software

To program any of the example projects, reset the Freefall (either by grounding the RST pin or by cycling power) and run

```
make boot
```

[1] Based on USBaspLoader. See `www.stingworks.com/freefall` for details and source code.

in the project directory on your PC within 3 seconds of the reset. For details, see the `Makefile` for the example.

If you wish to program a hex file directly, say `example.hex`, reset the Freefall and within 3 seconds run AVRDUDE with the following command-line:

```
avrdude -c usbasp -p m168 -U flash:w:example.hex
```.

The hex file will be programmed over USB. The bootloader will then exit, and the newly-loaded application will run.

Note that depending on your system configuration, you may need administrator privileges ("root" on a Linux system) for the Freefall to be properly recognized.

1.2.2 SPI Programming

An SPI ISP programmer may be used to program the Freefall. We recommend the USBtinyISP available from Adafruit Industries, and instructions here are geared towards users of this programmer. For more information on SPI programming, consult your programmer's manual or ATMEL's application note on SPI programming.

Hardware

Most AVR ISP programmers use either a 6-pin or a 10-pin ribbon connector. We recommend using a 10-pin connector of no more than 30cm in length. The correspondence between ISP wires/pins and pins on the Freefall are as follows:

ISP pin (6 or 10-pin connector)	**Freefall pin**
+Vcc	20 VDD
RESET	19 RST
Ground	18 GND
SCK	17 SCK
MISO	16 MI
MOSI	15 MO

Warning: The Freefall is designed to provide power to the ISP programmer on the VDD pin at 3.3 volts. Therefore, during ISP programming, either your programmer must be configured to receive power from the Freefall or you must not connect the VDD pin so as to avoid power supply contention. In the case of the USBtinyISP, this means that the USB PWR jumper must be removed.

Software

To program any of the example projects, simply run

```
make freefall_isp
```

within the project directory. The Freefall will be automatically reset by the programmer and will be programmed. For details, see the `Makefile` for the example.

If you wish to program a hex file directly, say `example.hex`, run AVRDUDE with the following command-line:

```
avrdude -c usbtiny -p m168 -e -U flash:w:example.hex
```.

Warning: Care must be taken when changing the value of the AVR's configuration fuses. Most importantly, the `SPIEN` fuse must be programmed (set to the value of 0) and the `RSTDISBL` fuse must be left unprogrammed (set to the value of 1). Changing either of these fuses will render the AVR permanently unprogrammable. Furthermore, the stock USB bootloader requires the default settings for the `WDTON` and clock source fuses; therefore, if you do not have access to an SPI programmer, you should not modify those fuses.

1.2.3 RS232 Bootloading

A third programming option is available for those who do not wish to use USB but prefer to use legacy RS232 serial communication. The RS232 bootloader is STK500-compatible, meaning it can be used with many software programs including the Arduino IDE.

Installing the Bootloader

The legacy bootloader must be installed on the Freefall via SPI programming. To do so, enter the `old-boot` directory and run the command

```
make isp
```.

Hardware

The RS232 bootloader requires the GND, RX, and TX pins to be connected to a serial port operating at TTL voltage levels (0-5V or 0-3.3V).

Warning: Do not connect the Freefall directly to a PC serial port! PC serial ports typically operate at $\pm 12V$ which will likely damage the AVR. A level-converter (such as a MAX232) or a TTL-level USB-to-serial adapter must be used.

Software

To program any of the example projects using the RS232 bootloader, reset the Freefall and within 3 seconds run

```
make old-boot
```

in the project directory. The Freefall will be automatically reset by the programmer and will be programmed. For details, see the `Makefile` for the example.

If you wish to program a hex file directly, say `example.hex`, run AVRDUDE with the following command-line:

```
avrdude -c avrisp -p m168 -P /dev/ttyS0 -b19200 -U flash:w:example.hex
```

changing the serial port specified after `-P` if nessisary. If you use a USB-to-serial converter, the serial port name is most likely `/dev/ttyUSB0`.

Chapter 2

Freefall API Reference

2.1 Freefall General Functions

- `freefall_init()` - this function must be called at the beginning of any firmware application to initialize the various modules in the Freefall. The correct data directions and default I/O pin levels are set, the ADC, CDS, PWM, LED, MMA, and PMAN modules are initialized, and the LED output drivers are set to high-Z mode.
- `force_reset()` - performs a soft reset, entering the bootloader. This is accomplished by allowing the watchdog timer to reset the AVR.
- `uint8_t pack_float(float f, float min, float max)` - returns an 8-bit compressed form of a floating-point input `f`, on the range between `min` and `max`.
- `float unpack_float(uint8_t packed, float min, float max)` - decompresses a floating-point number which was compressed with `pack_float`.

2.2 Accelerometer (MMA module)

The Freefall is equipped with an on-board $\pm 8g$ 3-axis digital micromachined accelerometer (MMA7455) capable of reporting 10-bit acceleration data at 250 samples/sec.

2.2.1 Accelerometer Coordinate System

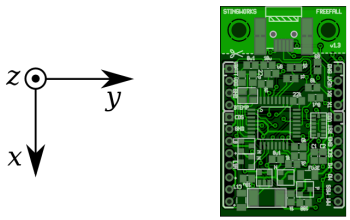

Figure 2.1: Direction of xyz-axes. Note that the z-axis is coming out of the page.

2.2.2 Using the Accelerometer in Firmware

The accelerometer is communicated with via the following C functions defined in `mma7455.h`. Note that `freefall_init()` must be called to initialize the accelerometer before using any of the following functions.

- `mma7455_data_ready()` - checks to see if a new measurement is available
- `mma7455_data_overrun()` - checks to see if measurements were missed due to slow reading
- `mma7455_read_x()` - returns a signed 8-bit reading of x-axis acceleration
- `mma7455_read_y()` - returns a signed 8-bit reading of y-axis acceleration
- `mma7455_read_z()` - returns a signed 8-bit reading of z-axis acceleration

- `mma7455_calibrate()` - may be called to recalibrate the accelerometer, storing result in EEPROM.[1] The board must be reseting in standard orientation (Z-axis facing up).

2.2.3 Advanced Accelerometer Features

The MMA7455 provides other advanced features such as built-in tilt, level, and freefall detection with configurable interrupts. The Freefall is wired so that the accelerometer may interrupt the AVR, for example, to wake it from sleep. However, the current version of the C driver does not implement these features. For details on implementing these features yourself, see the MMA7455 datasheet [5].

2.2.4 Sample Data

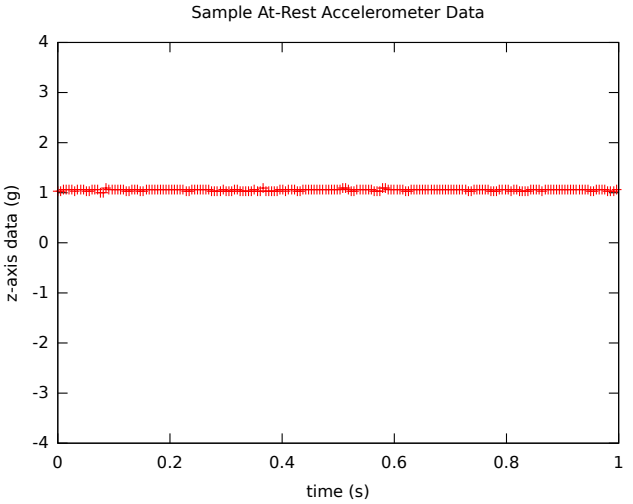

Figure 2.2: Sample data from z-axis of accelerometer in standard orientation. AVR was clocked at 12MHz and reported data live over USB. Noise level is $\pm 30mg$. Lower noise levels may be achieved by making measurements during AVR sleep or while the processor is set to a lower clock speed. Plotting was done using `gnuplot` on output from `hidstream`.

[1] This calibration information will remain in EEPROM even after reprogramming using the USB bootloader.

2.3 Power Management (PMAN module)

The Freefall is designed to run off a battery pack and/or bus power from a USB port. On-board battery charging circuitry may be employed to recharge the battery pack while the Freefall is running on USB power. The rate of charge may be controlled through PWM, and battery charge status may be monitored by battery voltage and temperature measurements.

A library of power management functions is provided to ease the implementation of battery recharging in a firmware application.

- `pman_init()` - sets proper data direction and slow charge mode. This function must be called shortly after reset if batteries are used.
- `pman_set_rate(uint8_t duty)` - sets charge rate duty cycle between 0 (slowest) and 255 (fastest).
- `pman_fast()` - sets fastest charge mode (equivalent to `pman_set_rate(255)`).
- `pman_slow()` - sets slowest charge mode (equivalent to `pman_set_rate(0)`).
- `pman_vext()` - returns the voltage of the Vext pin (i.e., the USB bus voltage) as a float in units of volts.
- `pman_vbatt()` - returns the voltage of the Vbatt pin (i.e., the battery pack, if attached) as a float in units of volts.
- `pman_vbatt()` - returns the voltage of the Vbatt pin (i.e., the battery pack, if attached) as a float in units of volts.
- `btemp_voltage()` - returns the voltage of the BTEMP pin as a float in units of volts.
- `btemp_C()` - returns the temperature read by the BTEMP temperature sensor[2] in degrees Celsius.

[2]This sensor must be a MCP9700 for this function to return the correct temperature reading. If you use some other sensor, you will have to change the formula in this function.

2.4 USB Connectivity

The Freefall is optimized for use with VUSB, a software USB stack developed by Objective Development, GmbH. Examples are provided for configuring the Freefall as a HID device, which operates with all major PC operating systems without drivers. In this configuration, one can emulate a real human-interface device (a keyboard, mouse, or joystick) or use the protocol for the exchange of arbitrary data between the Freefall and PC software.

2.4.1 Hardware

To use USB connectivity, the Freefall must be connected to a USB host (PC) via a USB cable, according to the pin descriptions given in Section 1.2.1.1.

2.4.2 Firmware for USB Mouse

To behave as a USB mouse, a firmware application must do the following:

- Include the `usbmouse.h` header file.
- Initialize the USB device by running `usbmouse_init()`.
- Periodically send x and y deltas to the PC, using the `usbmouse_update(x,y)`.

For details, see firmware example called `tilt-mouse`, which moves the mouse pointer on your PC according to how the Freefall is tilted.

2.4.3 Firmware for Custom Packet I/O

To send and receive custom data packets, a firmware application must do the following:

- Include the `usbpack.h` header file.
- Initialize the USB device by running `usbpack_init()`.
- Set the values to be sent in the outgoing packet, called `uint8_t packet_out[8]`, as desired.
- Periodically send the packet by running `usbpack_update()`.
- Read received values from the incomming packet, called `uint8_t packet_in[8]`.

- The usbpack driver updates the counters `uint8_t packets_received` and `uint8_t packets_sent`, which may be used if desired.

For details, see the firmware example called `sensor-report`, which sends the values read from the x, y, and z accelerometer axes, the CdS light sensor, and the several power management sensors to the PC. This firmware example also receives data from the PC: the 0th and 1st bytes of the received packet set the LED0 and LED1 duty cycles, respectively.

2.4.4 Software for Custom Packet I/O

Raw Data in Text-Mode

The `hid-stream` program may be used on the PC to read data sent by the Freefall using `usbpack` and send it to standard output. To do so, in the `hid-stream` directory run:

```
sudo ./hidstream
```.

Likewise, you can send a packet to the Freefall by sending it to `hidstream` on standard input.

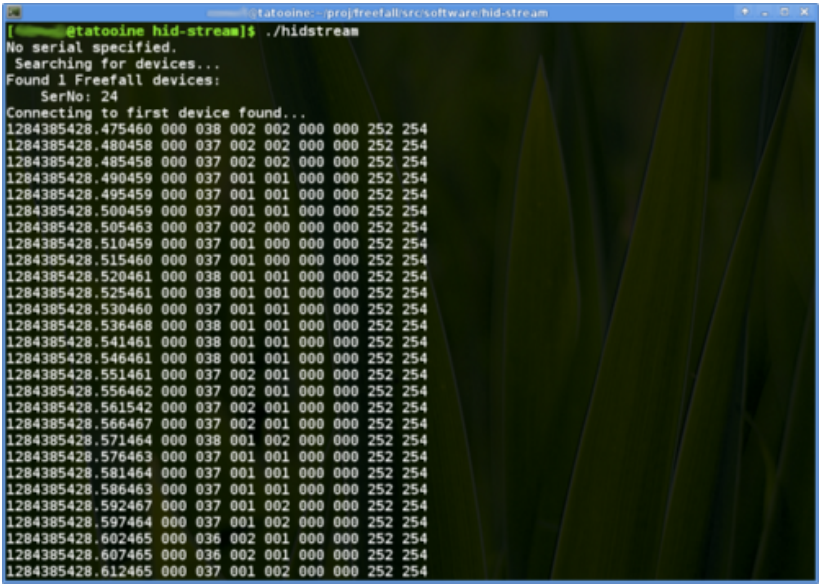

Figure 2.3: Example `hidstream` output. The first column is the UNIX timestamp of the incoming packet. The following columns are sent by the user's firmware application.

Real-Time Plots

Other software may be then used to display and analyze the data. For example, the `SteamingPot` application may be used to graph sensor data in real-time. For example, in the `steamingpot` directory, run:

```
steamingpot
```

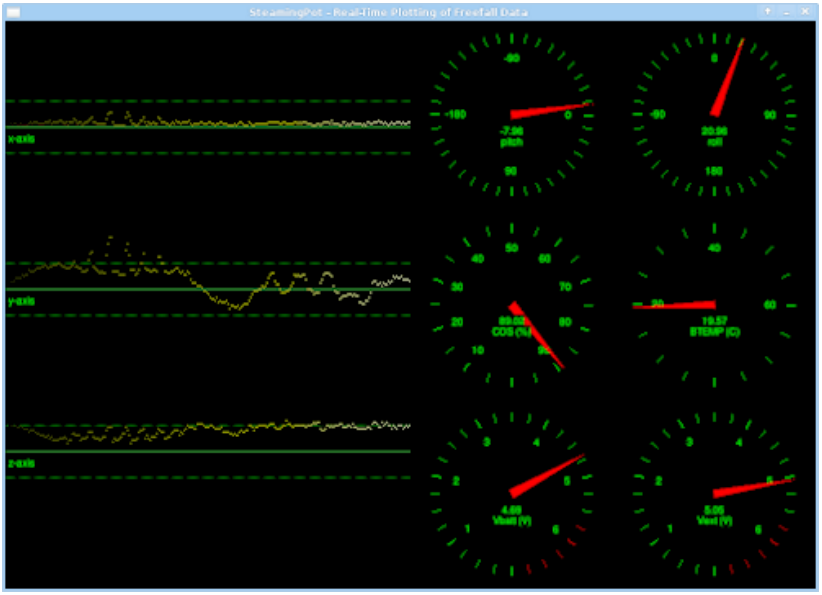

Figure 2.4: Example `steamingpot` output.

2.5 RS232 USART Connectivity

The Freefall can also communicate over legacy RS232 using the AVR's hardware USART.

2.5.1 Hardware

To use RS232 connectivity, the Freefall must be connected to a PC using either a level converter (e.g., MAX232) or a USB-to-serial converter.

2.5.2 Firmware for Sending Custom Packets

To send custom data packets over RS232, a firmware application must do the following:

- Include the `usartpack.h` header file.
- Initialize the USART by running `usartpack_init()`.
- Set the values in the packet, called `uint8_t packet_data[8]`, as desired.
- Periodically send the packet by running `usartpack_update()`.

2.5.3 Software for Receiving Custom Packets

Raw Data in Text-Mode

The `usart-stream` program may be used on the PC to read data sent by the Freefall using `usartpack` and send it to standard output. To do so, open `putty` and set the baudrate for `/dev/ttyUSB0` to 57600 baud; then, in the `usart-stream` directory run:

```
cat /dev/ttyUSB0 | ./usartstream
```.

Other software may be then used to display and analyze the data, as with `hid-stream`. For example, the `SteamingPot` application may be used to graph sensor data in real-time. See Section 2.4.4.2.

2.6 LED Drivers (LED module)

The Freefall is equipped with two 3 ampere LED driver outputs. Each driver may be placed in one of two states, or in a Pulse-Width Modulated (PWMed) combination of the two.

The default state, corresponding to sending a logic 0 to a driver, is called "off", "open", or "high-z". In this off state the corresponding LEDx pin acts as if it were disconnected. Sending a logic 1 to the driver turns on a N-channel MOSFET, connecting the corresponding LEDx pin to ground with very low resistance. This is the "on", "closed", or "grounded" state and it is what is used to turn an LED on.

Typically, the user wishes to set a brightness level for an LED somewhat less than that provided by the "on" driver state. This is accomplished by setting an 8-bit PWM duty cycle less than 255 (100%).

2.6.1 Hardware

The LED drivers are designed to power high-brightness LEDs with a nominal 3.4V forward voltage drop. The system has been extensively tested with a particular LED[3], however other high-brightness LEDs may serve as drop-in replacements.

To power a high-brightness LED, the positive lead of the LED should be connected to VLED while the negative lead is connected to one of the LEDn driver pins.

Warning: To drive an LED with a forward voltage drop less than 3.4V (such as standard colored LEDs with voltage drops near 0.7V), it is necessary to either place several LEDs in series or to use a series resistor to avoid an over-current condition.

2.6.2 Firmware

Firmware applications may control the LED drivers via the following functions:

- `led0_duty(duty)` - sets LED driver 0 to duty-cycle of $\frac{\texttt{duty}}{255} * 100\%$
- `led1_duty(duty)` - sets LED driver 1 to duty-cycle of $\frac{\texttt{duty}}{255} * 100\%$

[3]The C374T-WQN-CT0W0151 manufactured by Cree.

2.7 Light Sensor (CDS module)

The Freefall has a special analog input (pin 7, called CDS) designed for use with a Cadmium Sulfide light sensor (CdS). If you do not wish to use a light sensor, you can still use the CDS pin as a traditional analog input by setting the CDS pullup to HZ mode.

2.7.1 Hardware

By default the CDS module assumes that a CDS sensor is attached between the Freefall's CDS pin and ground. An on-board $22k\Omega$ pullup is enabled by default, forming a voltage divider with the light sensor. Higher light levels result in reduced sensor resistance and therefore lower voltage at the CDS pin.

2.7.2 Firmware

The following functions control the CDS module:

- `cds_init()` - sets data direction and enables an internal $22k\Omega$ pullup on the CDS pin. This function must be called before using the CDS module.
- `cds_read()` - returns a value between 0-255 which is read from the CDS analog input.
- `cds_voltage()` - returns the voltage of the CDS pin as a float in units of volts, derived from a 10-bit ADC measurement. By default, higher voltages correspond to lower light levels and vice versa.
- `cds_percent()` - returns the brightness level as a float in units of percent, derived from a 10-bit ADC measurement.
- `cds_mode_pullup()` - (re)configures the internal resistor to pull up. This is the default setting.
- `cds_mode_hz()` - disables the internal pullup/down resistor. This mode should be used if you want to use the CDS pin as a generic analog input.
- `cds_mode_pulldown()` - configures the internal resistor to pull down. In this configuration the CDS sensor should be wired between the CDS pin and a VDD pin.

2.8 Wireless (MiRF module)

If wireless communication is desired, the Freefall may be easily augmented with a wireless module (MIRF) based on the nRF24L01+ from Nordic Semiconductor. Two Freefall boards may communicate with eachother if they are both equipped with MIRF modules. Moreover, a MIRF-equipped Freefall board can communicate with other devices on the same band using the ANT protocol, although an ANT implementation is not provided.

2.8.1 Hardware

To attach a MIRF module to the Freefall, the following pins must be connected between the MIRF module and the Freefall using a cable or individual wires of length less than $10cm$:

MIRF pin	**Freefall pin**
VCC	20 VDD
CSN	14 MSS
MOSI	15 MO
IRQ	none
MISO	16 MI
SCK	17 SCK
CE	13 MM
GND	18 GND

2.8.2 Firmware

To communicate wirelessly using a MIRF module, the `mirf` driver is used. A simple firmware example is to repeatedly transmit sensor data to a PC for analysis. Such a transmitter program must do the following:

- Set the transceiver channel and packet size using defines at the top of your `main.c`, before including `freefall.h`; for example:

```
#define MIRF_CHANNEL 2
#define MIRF_PAYLOAD 8
```

- Allocate memory for a data packet of length `MIRF_PAYLOAD`; for example:

```
uint8_t packet[MIRF_PAYLOAD];
```

- Configure the MIRF module by running the `mirf_config()` function. This sets-up configuration registers inside the MIRF; for example:
- Load the data that you wish to transmit into into `packet`.
- Call `mirf_send(packet,MIRF_PAYLOAD)` to transmit the packet to another Freefall running a receiving program.

2.8.3 Software

A PC may be used to collect data sent by a Freefall board or to send data to a Freefall board over either USB or legacy RS232. If the Freefall board attached to the PC is equipped with a MIRF module, it is possible to establish wireless communication between the PC and other wireless boards.

2.9 Arduino-compatability

The Arduino programming language and development environment may be used to develop for the Freefall. See the `software/arduino/` directory for example sketches.

2.9.1 Configuring the Arduino IDE to Program the Freefall

To allow the Arduino IDE to program a Freefall board, replace the `hardware/boards.txt` from the Arduino software with the `software/arduino/boards.txt` file included with the Freefall software in the `arduino` directory.

After opening the Arduino IDE, navigate to the menu option `Tools > Board`sand select the "Freefall" option with the desired programming method.

2.9.2 Correspondence between Freefall Pins and Arduino Pins

The following table gives the corresponding Arduino pin for each Freefall pin whenever such a correspondence exists. Note that there are significant differences between the behavior of some pins on the Arduino and their correspondents on the Freefall. See Section 1.1.5 for details on individual pins.

Freefall pin	**Arduino Pin**
2 VDD	3v3
3 GND	GND
4 BTEMP	A3
7 CDS	A2
8 LED0	D6*
9 LED1A	D5*
11 LED1B	D5*
13 MM	D4
14 MSS	D7
15 MO	D11
16 MI	D12
17 SCK	D13
18 GND	GND
19 RST	reset
20 VDD	3v3
21 TX	D1
22 RX	D0
24 GND	GND

(*) pin behavior is significantly different than that on the Arduino. See Section 1.1.5.

2.10 Code Licensing

The user of the Freefall assumes all responsibility for proper licensing of his/her own firmware. Note using code from the several open-source libraries which are bundled with Freefall may obligate you to release your source code under a particular open-source license. For details, see the license information in any libraries you may be using. That said, we have taken pains to ensure that all code provided from Stingworks is licensed under compatible open-source licenses, such as the GNU Public License (GPL) or a BSD/MIT-style license.

Chapter 3

Source Code Examples

3.1 Blinkenlights

```
/*
 * Freefall Blinkenlights Example
 */

#include"freefall.h"

void main()
{
  // initialize the Freefall
  freefall_init();

  // begin main loop
  while (1)
  {
    led0_duty(0); // LED0 off
    _delay_ms(100); // wait 1/10th sec
    led0_duty(255); // LED0 on
    _delay_ms(100); // wait 1/10th sec

    // watch dog must be reset
    //  at least once every four seconds
    wdt_reset();
  }
}
```

3.2 Reading a Light Sensor

```
/*
 * Freefall CdS Light Sensor example
 */

#include"freefall.h"

void main()
{
  // initialize the Freefall
  freefall_init();

  // begin main loop
  while (1)
  {
    // LED0 duty cycle varies directly
    //  with observed brightness
    led0_duty(cds_read());

    // LED1 duty cycle varies inversely
    //  with observed brightness
    led1_duty(255-cds_read());

    wdt_reset();
  }
}
```

3.3 Reading the Accelerometer

```
/*
 * Freefall Accelerometer Example
 *
 * LED0 lights when the board is tilted to the
 *  left, and LED1 lights when the board is tilted
 *  to the right.
 *
 *  Note: this is an autistic program.
 */

#include"freefall.h"

void main()
{
  // initialize the Freefall
  freefall_init();

  // begin main loop
  while (1)
  {
    if (mma7455_read_y() > 0) {
      led0_duty(0);
      led1_duty(255);
    } else {
      led0_duty(255);
      led1_duty(0);
    }

    wdt_reset();
  }
}
```

3.4 USB Sensor Reporting and Control

```
/*
 * Freefall Sensor Reporting example
 *
 * Sensor data is read and sent over usb as HID
 *  packets. Run the SteamingPot or hidstream
 *  programs on the PC to receive, visualize,
 *  and store this data.
 */

#include"freefall.h"
#include"usbpack.h"

void main()
{
    long count;

    freefall_init();    // initialize Freefall
  usbpack_init();     // initialize usb packet module

  while (1) // main loop
  {
        count++;

        // read the accelerometer
    packet_out[FREEFALL_AX] = mma7455_read_x();
    packet_out[FREEFALL_AY] = mma7455_read_y();
    packet_out[FREEFALL_AZ] = mma7455_read_z();

    usbpack_update();   // communicate with PC
        pman_update();       // read power sensors
    wdt_reset();        // reset the watchdog timer
  }
}
```

3.5 Tilt-Controlled Mouse

```
/*
 * Freefall Tilt-Mouse example
 *
 * Tilt the Freefall to move the mouse pointer
 *  on the PC. Note that the Vendor/Product ID
 *  pair used in this example is taken from a
 *  Logitech mouse. This is for demonstration
 *  use only! Do not publish any hardware using
 *  these IDs.
*/

#include"freefall.h"
#include"usbmouse.h"

int main(void)
{
    uint8_t button = 0;
    uint8_t click_timer = 0;
    int8_t x,y,z;

    freefall_init();
    usbmouse_init();

    while(1) {
      // get readings from accelerometer
      y = mma7455_read_x();
      x = -mma7455_read_y();
      z = mma7455_read_z();

      // squelch noise axes
      if (abs(x) < 5) x = 0;
      if (abs(y) < 5) y = 0;

      // shake to click,
      //  invert to drag
      button = 0;
      if (z < 10) button = 1;

      // set the new mouse state
      usbmouse_update(x,y,button);

            wdt_reset();
    }
    return 0;
}
```

References

Microcontroller

[1] **ATmega48/88/168 Datasheet**, ATMEL, inc., Rev. 2545-AVR-07/09
http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf

[2] **AVR: In-System Programming**, ATMEL, inc., www.atmel.com/atmel/acrobat/doc0943.pdf

Battery Module

[3] **Low-Power Linear Active Thermistor ICs**, Microchip, inc., DS21942E
http://ww1.microchip.com/downloads/en/DeviceDoc/21942e.pdf

Wireless

[4] **nRF24L01+ Datasheet**, Nordic Semiconductor, Revision 1.0, Sept 2008
http://www.nordicsemi.com/files/Product/data_sheet/nRF24L01P_Product_Specification_1_0.pdf

Accelerometer

[5] **MMA7455L Datasheet**, Freescale Semiconductor, Rev 10, 12/2009
www.freescale.com/files/sensors/doc/data_sheet/MMA7455L.pdf

Index