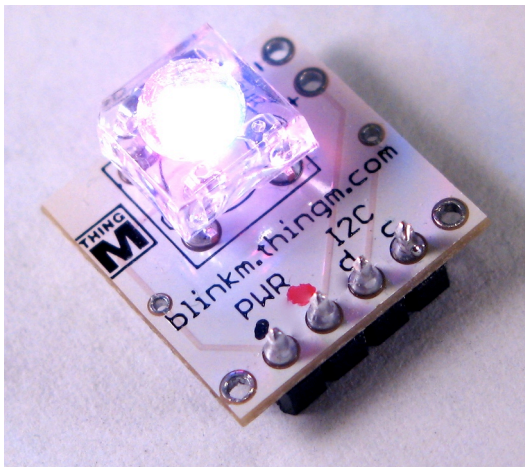
BLINKM v1 DATASHEET

blinkm.thingm.com



Description

BlinkM is a “Smart LED”, a networkable and programmable full-color RGB LED for hobbyists, industrial designers, prototypers, and experimenters.

It is designed to allow the easy addition of dynamic indicators, displays, and lighting to existing or new projects.

If you’ve used up all your microcontroller PWM channels controlling RGB LEDs and still want more, BlinkM is for you.

Features

- 8000 mcd 140º full-color RGB LED with 24-bit color control
- Specify colors by 24-bit RGB or HSB
- Fade between colors with variable timing and fade speeds
- Randomized color selection, with ranges and based on previous color
- 32 built-in light scripts (sequences)
- Create and save light scripts of up to 49 commands long
- Stand-alone operation: No microcontroller needed for light script playback
- Can plug directly into Arduino, no wiring or other components needed!
- Two-wire (aka “I2C”) remote commanding
- Up to 127 BlinkMs on a single two-wire network
- Responds to “general call” broadcast for simultaneous commanding
- Reconfigurable network address
- Firmware upgradable
- 5-volt standard TTL inputs
- Low power consumption

Application Ideas

- Prototype and Industrial Design indicators
- Personalized color accent lights
- Casemod lighting
- Programmable holiday lighting
- Safe tea lights
- Custom bike lights

blinkm.thingm.com

BLINKM v1 DATASHEET

Table of Contents

BLINKM v1 DATASHEET

1. Introduction

BlinkM is an example of a Smart Interface Component, an uplifting of traditionally dumb interface components into devices that embed within themselves domain-specific knowledge about their functioning. In this case, BlinkM is a Smart LED and knows how to turn 24-bit RGB or HSB color values into the corresponding high-frequency PWM signals needed to drive its super-bright RGB LED. It also goes one step further by embedding time-varying color sequences called “light scripts” that can be triggered with a single command, allowing complex light displays to occur with minimal overhead by whatever CPU is controlling BlinkM. Several BlinkMs can be controlled simultaneously or individually using only two signal lines from the controlling CPU. The control is in the form of a simple I2C command set usable with a variety of controllers.

1.1 Anatomy

Figure 1.1: BlinkM layout

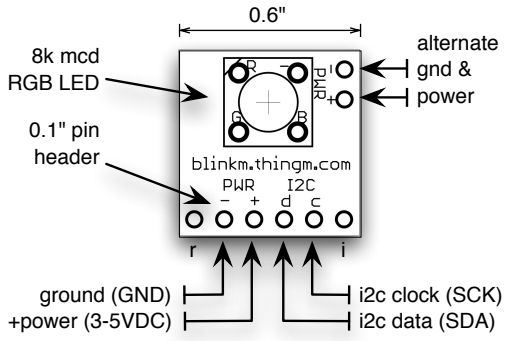
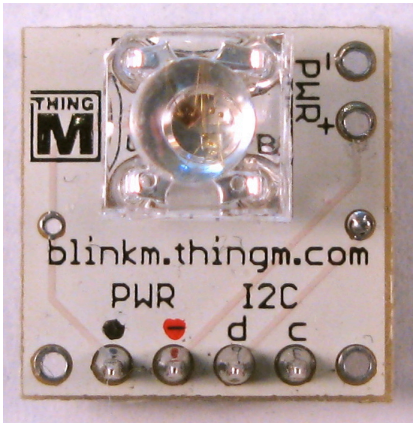


Photo 1.1: BlinkM



1.2 Connections

PWR –	Ground, aka “Gnd”
PWR +	3-5VDC regulated power input, aka “Vcc”
I2C d	I2C data input/output, aka “SDA”
I2C c	I2C clock input/output, aka “SCK”

The two outer connections are normally only used when programming the BlinkM CPU. Those connections are labeled on the bottom of the BlinkM:

r	Reset, normally unused. Used for programming
i	MOSI, normally unused. Used for programming

Note: normally I2C lines SDA & SCK require 4.7kΩ pull-up resistors. For short cable runs, the pull-up resistors inside most microcontrollers (like Arduino's AVR) are sufficient.

Note: The BlinkM CPU will operate down the 2.7V. However, not all LEDs will be able to turn fully on. The blue and green LEDs need approximately 3.5V to turn on completely. Running BlinkM at <3.5V doesn't harm anything but could reduce color accuracy.

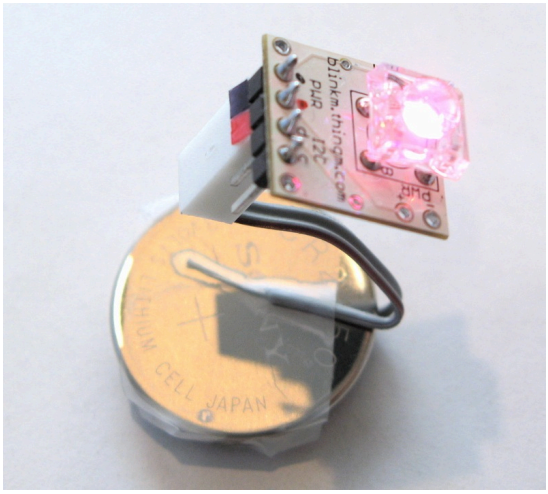
2. Getting Started

There are two main ways of using BlinkM: as a peripheral to an existing project or as a stand-alone object.

2.1. Stand-alone Operation

To test basic functionality, connect BlinkM's "PWR +" pin to +3-5VDC and "PWR –" pin to ground, as in Photo 2.1. BlinkM will play its default startup light script: white→red→green→blue→off. This startup script can be customized, see Section 2.3 "BlinkM Sequencer" or Section 4.2 "Playing Light Scripts".

Photo 2.1: BlinkM Stand-alone Operation



2.2 Peripheral Operation

The steps to start working with BlinkM as a peripheral are:

1. Connect power, ground, & I2C data lines between BlinkM and the I2C master device.
2. Power up BlinkM and the master device.

3. Send commands to BlinkM over I2C bus.

BlinkM is an I2C slave device just like the many other I2C devices on the market. Any device that can be an I2C master can control BlinkM. This includes Arduino, Basic Stamp, and USB-to-I2C adapters.

Perhaps the easiest way to get started programming a BlinkM is using an Arduino board. Arduino is a fun and easy-to-use microcontroller platform. To learn more about Arduino, visit the Arduino homepage: http://arduino.cc/. Arduino boards are available from Sparkfun (http://sparkfun.com/), Adafruit (http://adafruit.com/), and the Maker store (http://store.makezine.com/)

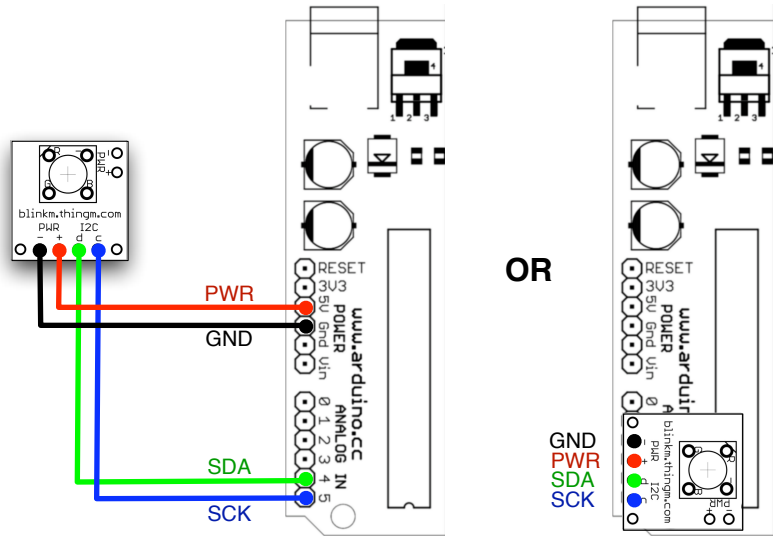
2.2.1 Connecting BlinkM

BlinkM needs two wires for power and two for data. Figure 2.2.1a shows the connections needed when hooked up to an Arduino. The Arduino "analog in" pins 4 & 5 also double as the I2C data signal ("SDA") and clock signal ("SCK"), respectively.

Even easier is to plug the BlinkM directly into Arduino, as in Figure 2.2.1b. In this configuration, power is drawn from the Arduino's analog in pins 2 & 3, configured via software to act as power pins. This does not damage the pins, but does reduce the maximum brightness of BlinkM by about 30%.

See "Other Circuits" below for details on how to connect BlinkM to a Basic Stamp 2.

Figure 2.2.1: Connecting BlinkM to Arduino



a. b.

2.2.2 Sending Commands to BlinkM

Once BlinkM is connected to an Arduino or other I2C master, commands can be sent to it. All I2C commanding follows the same basic structure:

1. Initialize the I2C subsystem of the master device
2. Join the I2C bus and indicate the address of the slave device to talk to
3. Send the bytes containing the command
4. Leave the I2C bus

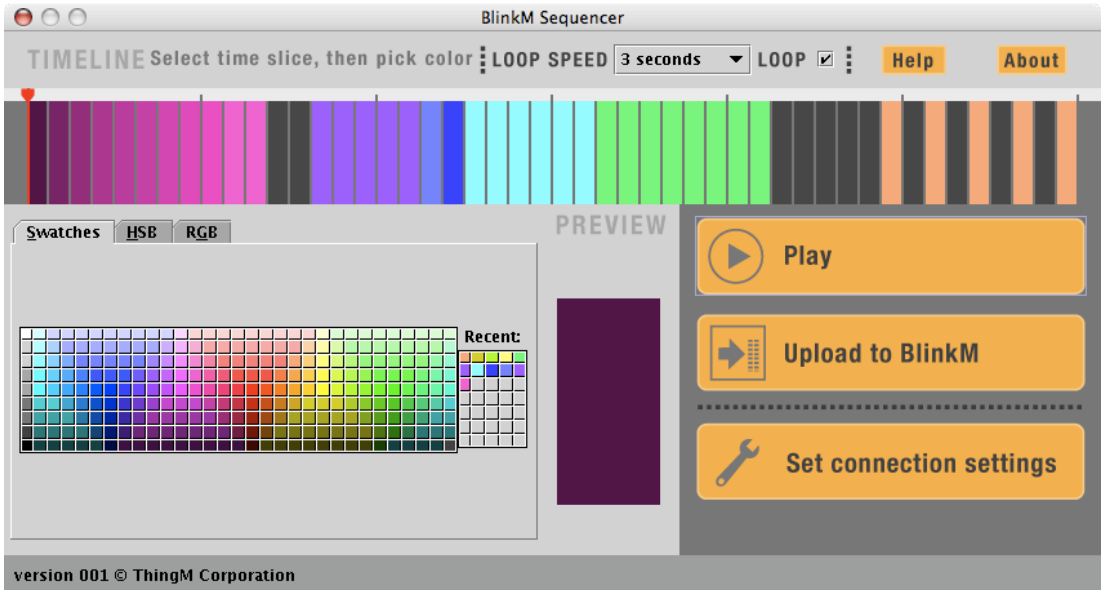
When using the “Wire” library for Arduino, such a sequence looks like:

```
Wire.begin();                 // set up I2C
Wire.beginTransmission(0x09);// join I2C, talk to BlinkM 0x09
Wire.send('c');               // 'c' == fade to color
Wire.send(0xff);              // value for red channel
Wire.send(0xc4);              // value for blue channel
Wire.send(0x30);              // value for green channel
Wire.endTransmission();       // leave I2C bus
```

2.3 BlinkM Sequencer

To start playing with BlinkM commanding immediately, there is the handy BlinkM Sequencer program for Mac OS X, Windows, and Linux available at http://blinkm.thingm.com/. It allows the creation of light scripts using a drum machine-style metaphor and requires no programming or hardware experience. All that’s required is an Arduino and a BlinkM.

Figure 2.3: BlinkM Sequencer



3. BlinkM Commands

All commanding of BlinkM is done via the I2C bus. For more information about I2C, see http://www.best-microcontroller-projects.com/i2c-tutorial.html. When using Arduino or AVR microcontrollers, the Wiring "Wire" library can be used to make I2C communication simpler. See the Wire reference at http://wiring.org.co/reference/libraries/Wire/ for more details.

3.1 Command Structure

BlinkM commands consist of a one byte command code and zero or more argument bytes. The command code byte's ASCII value is mnemonically related to the action performed.

3.1.1 I2C Addresses

The default BlinkM address is 0x09. It can be changed at any time with the "Set Address"("A") command described below. BlinkM responds to both its defined I2C address and the "general call" broadcast address (0x00).

The general call address can also be used to address all BlinkMs simultaneously, as most all BlinkM commands do not return a value (which is not allowed when using general call).

The use of general call may not work on I2C networks with other devices that also use general call.

3.1.2 Time ticks, units

The time unit used in BlinkM commands and durations is a "tick", which is equal to 1/30th of a second (33.33 milliseconds).

3.1.3 Numbering Conventions

Numbers are interchangeably represented as decimal or hexadecimal, and in some cases, ASCII characters. Hexadecimal numbers are represented with either a "0x" prefix (to indicate use in code, like "{0xff,0x00,0x9a}") or "#" prefix (to indicate a color, like "#FF00FF");

3.2 Command List Summary

Below are all commands recognized by BlinkM, along with how many arguments they take, the return values they produce, and a command format overview.

The "cmd char" is the command byte in ASCII character form. The character is chosen to be mnemonic of the command's function. The "cmd byte" column is the actual byte value of the ASCII character value sent over the wire.

command name	cmd char	cmd byte	# args	# ret vals	format
Go to RGB Color Now	n	0x6e	3	0	{'n',R,G,B}
Fade to RGB Color	c	0x63	3	0	{'c',R,G,B}
Fade to HSB Color	h	0x68	3	0	{'h',H,S,B}
Fade to Random RGB Color	C	0x43	3	0	{'C',R,G,B}
Fade to Random HSB Color	H	0x48	3	0	{'H',H,S,B}
Play Light Script	p	0x70	3	0	{'p',n,r,p}
Stop Script	o	0x6f	0	0	{'o'}
Set Fade Speed	f	0x66	1	0	{'f',f}
Set Time Adjust	t	0x74	1	0	{'t',t}
Get Current RGB Color	g	0x67	0	3	{'g'}
Write Script Line	W	0x57	7	0	{'W',n,p,...}
Read Script Line	R	0x52	2	5	{'R',n,p}
Set Script Length & Repeats	L	0x4c	3	0	{'L',n,l,r}
Set BlinkM Address	A	0x41	4	0	{'A',a...}
Get BlinkM Address	a	0x61	0	1	{'a'}
Get BlinkM Firmware Version	Z	0x5a	0	1	{'z'}
Set Startup Parameters	B	0x42	4	0	{'B',m,n,f,t}

3.3 Command Details

Each command below has its format listed on the right edge. This format resembles a byte array used in Java, C, and Arduino, and is a mnemonic for succinctly noting the layout of a command that can also be copied almost verbatim and used as code.

Go to RGB Color Now `format: {'n',R,G,B}`

This command sets the BlinkM to a particular RGB color immediately. The command takes three argument bytes, one each for setting the levels of the red, green, and blue channels. Each value ranges from 0-255 (`0x00-0xff` in hexadecimal), with 0 being off and 255 being maximum brightness, just like web colors. For more information about the RGB color model, see Section 4.3 “Color Models” below.

This command does not return a value.

Examples:

```
{'n', 0xff,0xff,0xff}   // set full on (bright white) now
{'n', 0x00,0x00,0x00}   // set full off (dark)
{'n', 0xff,0x00,0x00}   // set full red
{'n', 0x00,0xff,0x00}   // set full green
{'n', 0x00,0x00,0xff}   // set full blue
```

Fade to RGB Color

format: `{'c',R,G,B}`

This command tells BlinkM to fade from the current color to the specified RGB color. The command takes three argument bytes, one each for setting the levels of the red, green, and blue channels. Each value ranges from 0-255 (`0x00-0xff` in hexadecimal), with 0 being off and 255 being maximum brightness, just like web colors. For more information about the RGB color model, see Section 4.3 “Color Models” below.

The rate at which the fading occurs is controlled by the “Set Fade Speed” (`'f'`) command. The default fade time is 15 time units.

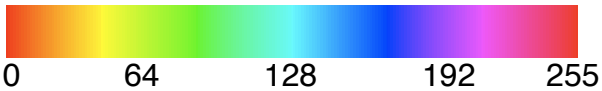
Examples

```
{'n', 0xff,0xff,0xff}   // set full on (bright white) now
{'c', 0x00,0x00,0xff}   // fade to bright blue
{'c', 0xff,0xff,0x00}   // fade to yellow
```

Fade to HSB Color

format: `{'h',H,S,B}`

This command will fade from the current color to the specified HSB color. The command takes three bytes as arguments. The first argument byte is the hue (or raw color), with the following mapping from 0-255.



The second argument is the saturation, or vividness, of the color. A saturation of 0 means a very light/white color and a saturation of 255 means a very vivid color. The third argument is

the brightness of the resulting color, where 0 is totally dark and 255 means maximally bright. For more information about the HSB color space, see Section 4.3 “Color Models” below.

The rate at which the fading occurs is controlled by the “Set Fade Speed” (`'f'`) command. The default fade time is 15 time units.

Examples:

```
{'h', 128, 0xff,0xff}   // fade to cyan
{'h', 172, 0xff,0xff}   // fade to bright blue
{'h',  43, 0xff,0xff}   // fade to yellow
{'h',  43, 0x00,0xff}   // fade to white
```

Fade to Random RGB Color

format: `{'C',r,g,b}`

This command fades from the current color to a random color. It takes 3 bytes as arguments, one for each R,G,B channel. Each argument is the range or amount of randomness for each of the R,G,B channels from which to deviate from the current color.

A setting of 0 for a channel means to not change it at all.

This command is good for creating randomly fading colors like a mood light.

Examples:

```
{'n', 0xff,0xff,0x00}   // set color to yellow now
{'C', 0xff,0x00,0xff}   // random fade to a purplish color
```

Fade to Random HSB Color

format: `{'H',h,s,b}`

This command fades from the current color to a random color. It takes 3 bytes as arguments, one for each H,S, B value. Each argument is the range or “degree” of randomness to deviate from the current H,S,B color.

A setting of 0 for a channel means to not change it at all.

This command is good for creating randomly fading colors like a mood light.

Examples:

```
{'n', 0xff,0xff,0x00}   // set color to yellow now
{'H', 0xff,0x00,0x00}   // fade to random hue, keep sat & bright
```

Play Light Script

format: {'p',n,r,p}

This command will play the specified light script immediately, stopping any currently playing script. The command takes two bytes as arguments. The first byte is the script id of the script to play. A list of the available scripts is below. The second argument is the number of repeats to play the script. A repeats value of 0 means play the script forever. The last argument is the script line number to start playing from. A value of 0 means play the script from the start.

To adjust the playback speed of a script that's running, adjust the fade speed ("Set Fade Speed", `'f'`) and time adjust ("Set Time Adjust", `'t'`) to taste. Altering these values can greatly alter the lighting effect for the built-in light scripts.

For more conceptual information about BlinkM light scripts, see Section 4.2 "Light Scripts" below.

Examples:

```
{'c', 0x00,0x00,0x00}   // fade to black
{'p', 0x01,0x05,0x00}   // play script 1 five times
```

Pre-defined light scripts

For exact commands used to produce these light scripts, see the "`blinkm_nonvol_data.h`" header file available on blinkm.thingm.com.

id	description	color sequence
0	eeprom script default startup	white→red→green→blue→off (can be reprogrammed)
1	RGB	red→green→blue
2	white flash	white→off
3	red flash	red→off
4	green flash	green→off
5	cyan flash	cyan→off
6	magenta flash	magenta→off
7	yellow flash	yellow→off
8	black	off

id	description	color sequence
9	hue cycle	red→yellow→green→cyan→blue→purple
10	mood light	random hue→random hue
11	virtual candle	random yellows
12	virtual water	random blues
13	the seasons	spring colors→summer→fall→winter
14	thunderstorm	random blues & purples→white flashes
15	stop light	red→green→yellow
16	morse code	S.O.S in white
17...	...surprises...	

Stop Script

format: {'o'}

This command stops any currently playing script. If no script is playing, this command has no effect. It takes no arguments.

Examples:

```
{'p', 0x06,0x00,0x00}   // play script 6 forever
...                     // watch it for awhile
{'o'}                   // stop the script
```

Set Fade Speed

format: {'f',f}

This command sets the rate at which color fading happens. It takes one argument that is the fade speed from 1-255. The slowest fading occurs when the fade speed is 1. To change colors instantly, set the fade speed to 255. A value of 0 is invalid and is reserved for a future "Smart Fade" feature.

This command does not return a value.

Examples:

```
{'p', 0x06,0x00,0x00}   // play script 6 forever
{'f', 15}               // set fade speed to 15
```

Set Time Adjust

format: {'t',t}

This command adjusts the playback speed of a light script. It takes one byte as an argument, a signed number between -128 and 127. The argument is treated as an additive adjustment to all durations of the script being played.

A value of 0 resets the playback speed to the default.

This command does not return a value.

Examples:

```
{'p', 0x03,0x08,0x00}   // play script 3 eight times
{'t', -10}              // but at a faster speed
```

Get Current RGB Color

format: {'g'}

return values: {R,G,B}

This command returns the current color in RGB format. The command takes no argument bytes but returns 3 bytes representing the current values of the red, green and blue channels.

Note: that if the BlinkM is currently fading between colors, this command returns the instantaneous current color value, not the destination color.

Examples:

```
{'c', 0x99,0x33,0xcc}   // set color to #9933cc now
{'g'}                   // get color back (should be 9933cc)
```

Write Script Line

format: {'W',n,p, d,c,a1,a2,a3}

This command writes a light script line. The first argument is which script id to write to. Currently, only script id 0 can be written to. The second argument is which line in the script to change, and can range from 0-49. The third argument is the duration in ticks for that command to last. The next four arguments are the BlinkM command and its arguments. Any command with less than 3 arguments should fill out the remaining arguments slots with zeros.

Once all the lines of the desired script are written, set the script length with the "Set Script Length" ("L") command.

This command does not return a value.

Examples:

```
// write to line 3 a “fade to purple” command w/ duration 20
{‘W’,0,3, 20, ‘c’,0xff,0x00,0xff}   // write to line 3
```

Read Script Line

format: {‘R’,n,p}

return values: {d,c,a1,a2,a3}

This command reads a script line and returns the script line’s values. The first argument is the script id to read from. Script id 0 is the eeprom script that can be written to, Script ids >0 refer to the built-in ROM scripts. The second argument is the number of the script line to read back.

There are 5 bytes of return values: d = duration in ticks, c = command, a1,2,3 = arguments for command. If an invalid script id or script line number is given, all return values are zeros.

Examples:

```
// read line 3 of script id 0
{‘R’,0,3}   // read line 3
```

Set Script Length & Repeats

format: {‘L’,n,l,r}

This command sets the length of a written script. The first argument is the script id to set, currently only script id of 0 is valid. The second argument is the length of the script, and the third argument is the number of repeats for the script.

This command does not return a value.

Examples:

```
{‘L’, 0x00,10,0x01}   // set script id 0 to a len. of 10, one repeat
```

Set BlinkM Address

format: {‘A’,a,0xd0,0x0d,a}

This command sets the I2C address of a BlinkM. It takes four arguments. The first and last argument are the new address, and the second and third arguments are {0xd0,0x0d}. These two arguments are used as a check against inadvertent address changing. This command can be used with the I2C “general call” broadcast address to change the address of a BlinkM if the previous address is not known. When using general call, only have one BlinkM powered up on the bus at a time or they will all change their address.

This command does not return a value.

Examples:

```
{‘A’, 0x12,0xd0,0x0d,0x12}   // change address to 0x12
```

Get BlinkM Address

format: {‘a’}

return values: {a}

Returns the I2C address.

Examples:

```
{‘a’}                    // get address (default is 0x09)
```

Get BlinkM Firmware Version

format: {‘z’}

return values: {v1,v2}

Returns the BlinkM firmware version. The first byte is the major version, the second byte is the minor version.

Examples:

```
{‘z’}                    // get version (default is ‘a’,’a’)
```

Set Startup Parameters

format: {‘B’,m,n,f,t}

This command sets the startup (or “boot”) action for BlinkM. The command takes four arguments. The first argument is the startup mode: 0 means do nothing, 1 means play a script. The second argument is which script id to play. The third argument is the fade time to use for the script, and the last argument is the time adjust to use with the script.

This command does not return a value.

Examples:

```
// on startup, play script 0 ten times,
// with fadespeed = 0x20 and time adjust of -5
{‘B’,1,0,10,0x20,-5}
```

4. BlinkM Concepts

BlinkM has many features. This section goes into some of the conceptual aspects of how several key BlinkM features work.

4.1 I2C Addressing

BlinkM ships with a default I2C address of 0x09. Feel free to change this address so it doesn't collide with any other I2C devices present on the I2C bus.

The BlinkM address can be changed if the current address is unknown. The "Set BlinkM Address" ('A') command can be sent to the I2C "general call" (i.e. broadcast) address. The general call address is 0x00. This allows changing of a BlinkM's address without knowledge of its prior address. Be sure to only have one BlinkM powered up on the I2C bus when using general call.

See "Set BlinkM Address" and "Get BlinkM Address" commands for more details.

4.2 Light Scripts

BlinkM Light scripts can be used to create complex patterns of light that are triggered via a single command. There are several built-in "ROM" light scripts and one light script that can be reprogrammed.

A light script is a sequence of timed BlinkM commands ("script lines"), as well as the script length in script lines and the number of repeats it should naturally last.

The possible commands can be any combination of the commands:

- "n" – Set RGB color now
- "c" – Fade to RGB color
- "h" – Fade to HSB color
- "C" – Fade to Random RGB color
- "H" – Fade to Random HSB color
- "f" – Set Fade time
- "t" – Set Time Adjust
- "p" – Play light script

Each "line" in a light script describes a duration for that script line and a BlinkM command with up to 3 arguments. The duration value is in 'ticks' (1/30th of a second), and can range from 1 to 255. The BlinkM command and args are the ones listed above and described in Section 3. If a BlinkM command has less than three arguments, the remaining argument slots should be filled with zeros.

When a script is played, each line is played one after the other until the end of the script. If the script is set to loop, it restarts playing from the first script line. When playing a script line,

its command is invoked and then BlinkM will wait for line's duration to be up before going on to the next script line.

If a script contains a "p" command, it will start playing a new script and forget the currently playing one.

For details on how to play and write light scripts, see "Play Script" ("p"), "Write Script Line" ("W"), "Read Script Line" ("R"), and "Set Startup Parameters" ("B").

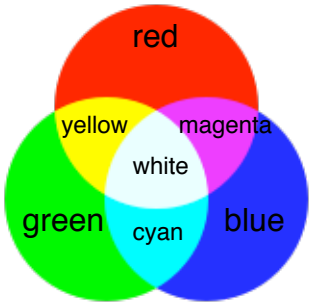
4.3 Color Models

BlinkM supports two different color models: RGB and HSB. RGB is the color model most people are familiar with. It's used to specify colors on web pages. The color "#FF0088" (a reddish purple) are the three components of red, green, and blue. The HSB color model uses one number for color, or hue, and then two other numbers to specify the lightness/ darkness of the color and vividness of the color.

4.3.1 About the RGB Color Model

When dealing with RGB LEDs, the simplest way to describe a color is to describe the percentage of light from each of the Red, Green, and Blue primary colored components. Various combinations of R,G,B can create any color in the spectrum. If equal intensities of red, green, and blue colors are mixed, the result will be white. Figure 4.3.1 shows this RGB additive color mixing for the secondary colors cyan, yellow, and magenta. White results when equal parts red, green, and blue are mixed.

Figure 4.3.1: RGB additive color model

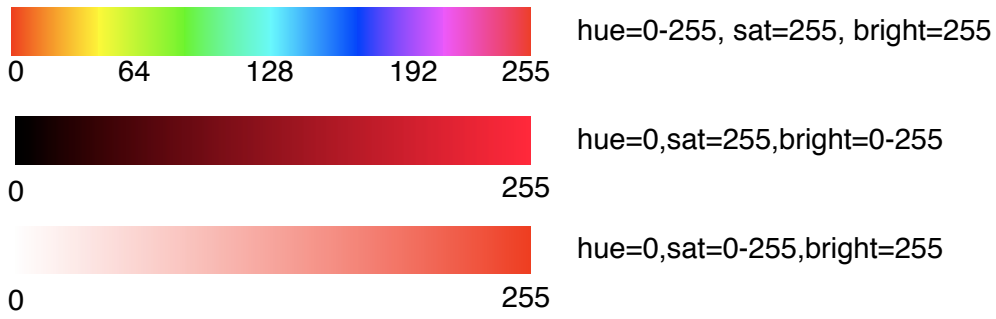


4.3.2 About the HSB Color Model

An alternate way of describing color instead of its R,G,B components is to specify its hue ("H"), how vivid, or saturated, that hue is ("S"), and how bright the color is ("B"). This manner of describing color is called the "HSB" or "HSV" color space. ("V" == value == brightness)

The HSB color model is useful when adjusting only the brightness of a color, without affecting its hue, or vice versa.

Figure 4.3.2: Hue, Saturation, & Brightness values for HSB color model



This is equivalent to going around the edge of the color wheel but instead of ranging from 0º to 360º, the hue value ranges from 0-255.

When experimenting with HSB, it's best to set saturation and brightness to both 255 to dial in the color desired. After the desired hue is reached, adjust brightness and saturation to taste.

4.3.3 Color Response and Calibration

BlinkM is not a color-calibrated device. The RGB or HSB values sent to it will not match exactly the same values on a computer screen. There are few reasons for this. Partly it is because of the logarithmic brightness response of LEDs. Even when this logarithmic response is taken into account, there are 1-5% variation in the component values.

4.4 Timing variations

BlinkM uses an internal PLL RC oscillator with approximately 1% accuracy. This relatively low accuracy doesn't affect I2C communications but does become apparent when running multiple BlinkMs with long-duration light scripts. If synchronization is important, periodically resync the BlinkMs by either power cycling them (power up time is less than a millisecond) or sending "Play Script" or similar commands over I2C.

5. Other Circuits

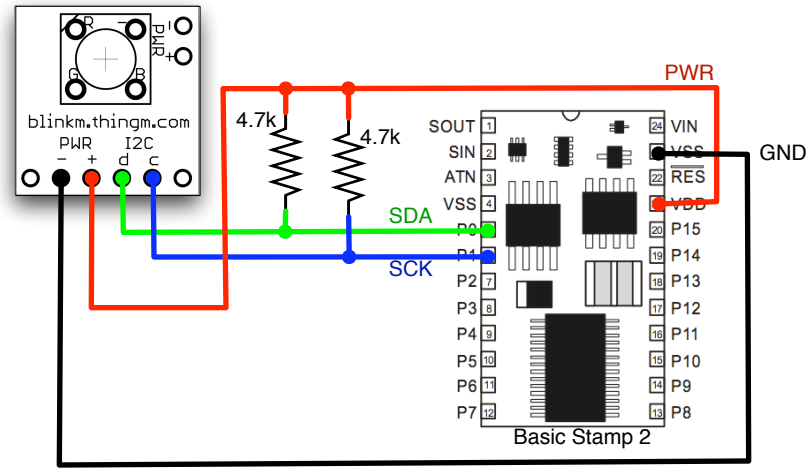
BlinkM can be used in many ways, with a wide variety of controllers and power sources.

5.1 Connecting BlinkM to a Basic Stamp

Unlike the Arduino, which has built-in pull-up resistors on the I2C lines, a Basic Stamp requires external pull-ups. Figure 5.1 shows one method of wiring up a BlinkM to a Basic

Stamp 2.. The BlinkM “I2C d” (SDA) line is connected to Basic Stamp P0 an the “I2C c” (SCK) line is connected to Basic Stamp P1. See blinkm.thingm.com for code examples.

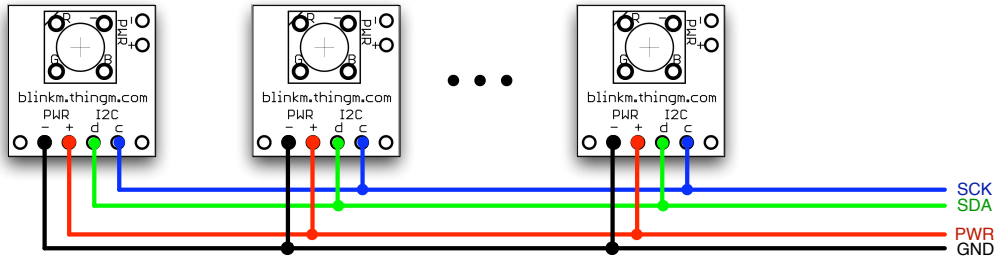
Figure 5.1: Connecting BlinkM to a Basic Stamp 2



5.2 Connecting Multiple BlinkMs

BlinkM communication is done via I2C, a simple network protocol. To add multiple BlinkMs to a circuit, connect them all their I2C data and clock lines together as in Figure 5.2.

Figure 5.2: Connecting Multiple BlinkMs



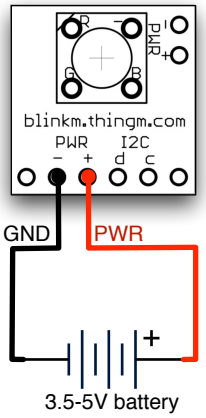
5.3 Battery Powered BlinkM

Once a light script is programmed in and set to run on startup, BlinkM can function entirely stand-alone and from a battery. This could be useful for custom bike lights and so on. To turn on and and off BlinkM, just apply and remove power. Any battery between 3V and 5V will work with BlinkM.

Coin cells that are between 3-5V will also work, as in Photo 2.1, but their high internal resistance means BlinkM maximum brightness is reduced. Also coin cells have a small capacity so will not last very long if driving a BlinkM that is always on.

In general, BlinkM brightness is inversely-correlated with battery life. If a BlinkM is always on and at full-brightness, battery life will be half of what it would be if the BlinkM was at half-brightness or blinking with a 50% duty-cycle.

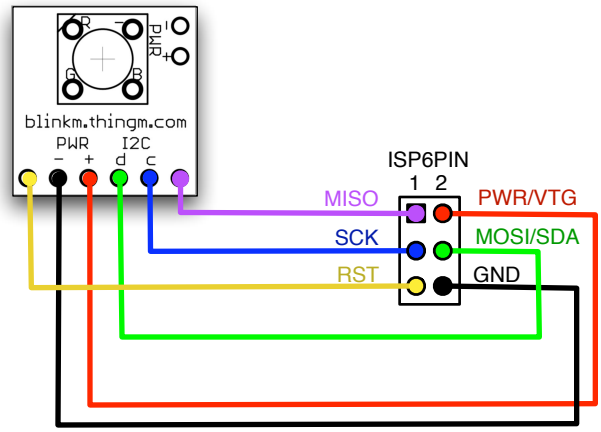
Figure 5.3: Battery Powered BlinkM



5.4 Reprogramming BlinkM's Flash Memory

BlinkM can also be used as a general AVR ATtiny45 development board. The ATtiny45 is very similar to most other AVR chips, like those in Arduino. The 6 connections at the bottom of BlinkM form a complete AVR-ISP set of connections (albeit in an alternate form factor). Figure 5.4 shows how to convert the BlinkM pins to a 6-pin AVR-ISP connector.

Figure 5.4: BlinkM to AVR-ISP wiring diagram



Note: most programmers do not supply power to the “VTG” (“PWR”) line, so power will need to be supplied to BlinkM in order to program it.

Note: do not try to sent I2C commands to a BlinkM while being programmed or the programming will fail.

6. Code Examples

These are code examples for Arduino/AVR, a common microcontroller platform. Other microcontrollers (with or without built-in I2C) such as the Basic Stamp 2 will follow similar practices when communicating with BlinkM.

There are several complete code examples available at http://blinkm.thingm.com/.

6.1 Arduino/AVR

The Arduino/AVR examples use the Wiring “Wire” library to perform I2C operations. To use the Wire library, include it at the top of each sketch with “`#include “Wire.h”`”.

6.1.1 Basic commanding

Commands are sent to the BlinkM’s address (or the general call address). The bytes of the command are sent one after the other.

Thus to send the command “`{‘f’,0xff,0x00,0x00}`” (i.e. “Fade to full red”):

```
#include “Wire.h”
Wire.begin();                // set up I2C
Wire.beginTransmission(0x09);// join I2C bus, to BlinkM 0x09
Wire.send(‘f’);              // ‘f’ == fade to color
Wire.send(0xff);             // value for red channel
Wire.send(0x00);             // value for blue chan.
Wire.send(0x00);             // value for green chan.
Wire.endTransmission();      // leave I2C bus
```

6.1.2 Reading Command Responses

For the commands that return a response, a second “read” transaction follows the “write” transaction

```
#include “Wire.h”
Wire.begin();                 // set up I2C
Wire.beginTransmission(0x09); // join I2C bus, to BlinkM 0x09
Wire.send(‘g’);               // ‘g’ == get current RGB color
Wire.endTransmission();       // done with command send
if( Wire.available() ) {      // make sure there’s data
    byte r = Wire.receive();  // get red value
    byte g = Wire.receive();  // get blue value
```

```
    byte b = Wire.receive();  // get green value
}
```

6.1.3 Using the `BlinkM_funcs.h` library

To make communicating with BlinkM easier on Arduino, a library of useful functions is available from blinkm.thingm.com called "`BlinkM_funcs.h`".

Place this file in the same directory as the Arduino sketch and "`#include`" it at the top to import all the functions.

The above commands using `BlinkM_funcs.h` look like:

```
#include "BlinkM_funcs.h"
byte addr = 0x09;
byte r,g,b;
BlinkM_begin();                          // init BlinkM funcs
BlinkM_fadeToRGB(addr, 0xff,0x00,0x00); // fade to red
BlinkM_getRGBColor(addr, &r,&g,&b);     // get curr. RGB color
```

Every function except `BlinkM_begin()` has as its first argument the address of the BlinkM to control.

For more information about the `BlinkM_funcs.h` library, see the instructions at the top of that file.

6.1.4 Programming Light Scripts

Light scripts are the most complex aspect of talking to BlinkM, and are optional if BlinkMs in real-time under another processor's control. However, they do allow one to free up

For more information about light scripts, see "5.3 Light Scripts".

```
#include "BlinkM_funcs.h"
// a script line contains: {dur, {cmd, arg1,arg2,arg3}}
blinkm_script_line  script_lines[] = {
 {  1, {'f',   20,0x00,0x00}},  // set fade speed to 20
 { 20, {'c', 0x11,0x12,0x13}},  // fade to rgb #112233
 { 20, {'c', 0xff,0xcc,0xee}},  // fade to rgb #ffccee
 { 20, {'c', 0x88,0x88,0x88}},  // fade to rgb #888888
 { 20, {'C', 0x00,0x7f,0x7f}},  // randomly alter grn & blu
};
byte addr = 0x09
byte script_id = 0;           // can only write to script 0
byte script_len = 5;          // number of lines in script
BlinkM_begin();               // init BlinkM funcs
BlinkM_writeScript(addr, script_id,script_len,&script_lines);
```

6.1.5 Talking to multiple BlinkMs

There are two ways to control multiple BlinkMs: addressing each directly or using the I2C "general call" address ("0",zero) to address them all simultaneously. The general call method is only useful for those command that do not return a value (this represents all color control and light script playing commands). The general call is thus like a broadcast address that can be used to synchronize a set of BlinkMs.

When addressing each BlinkM independently, just specify the address of a specific BlinkM. Using `BlinkM_funcs.h`, controlling multiple BlinkMs is straightforward:

```
#include "BlinkM_funcs.h"
byte addr1 = 0x09;   // the first blinkm
byte addr2 = 0x12;   // the second blinkm
BlinkM_begin();                          // init BlinkM funcs
BlinkM_fadeToRGB(addr1,0xff,0x00,0x00); // fade 1st to red
BlinkM_fadeToRGB(addr2,0x00,0x00,0xff); // fade 2nd to blue
BlinkM_fadeToRGB(0, 0xff,0xff,0xff);    // fade all to white
```

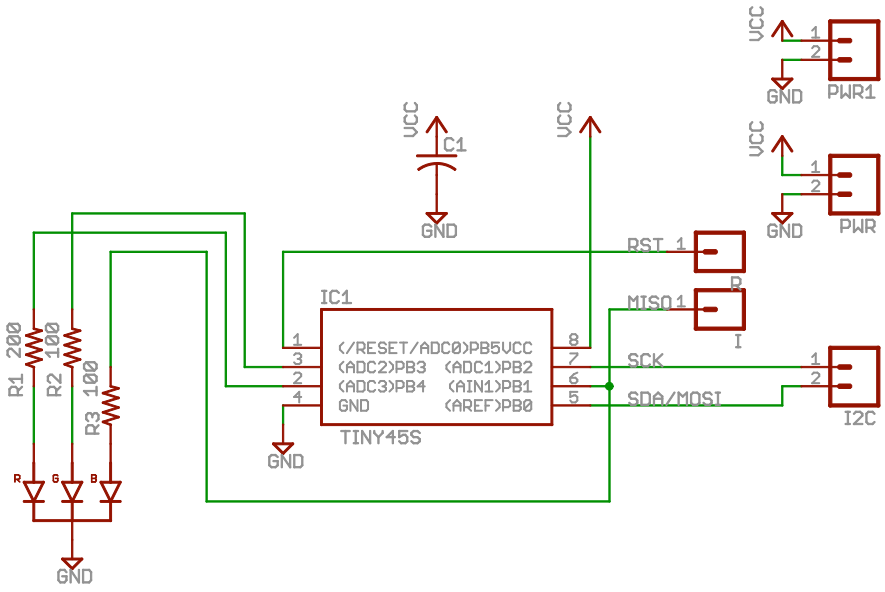
7. Electrical Characteristics

symbol	parameter	Condition	min	typ	max	units
Vcc	Operating Voltage		3*	5	5.5	V
Icc	Power Supply Current	LED full dark			1.5	mA
		LED full bright			60	mA
		RESET held low			1	mA

*Note: LEDs might not fully turn on at voltages below 3.5V.

All other electrical characteristics are the same as those for Atmel's ATtiny45 AVR microcontroller. See http://atmel.com/avr/ for more details.

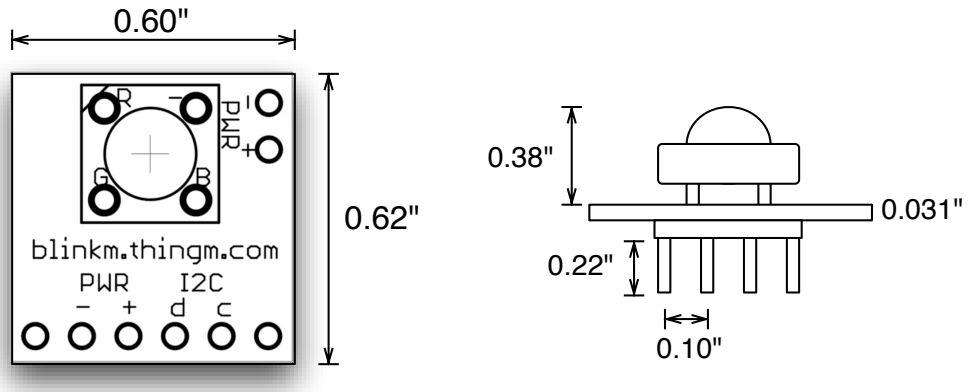
8. BlinkM Schematic



9. Packaging Information

All units in inches.

Figure 8: Packaging Information



THINGM LABS

http://thingm.com/

address
1126 Palm Terrace
Pasadena, CA 91104