blinkm.thingm.com

BLINKM DATASHEET

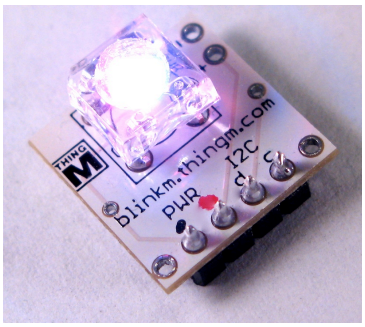
BlinkM
BlinkM MinM
BlinkM MaxM

Description

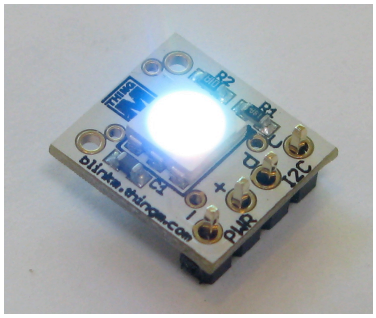
BlinkM is a “Smart LED”: a networkable and programmable full-color RGB LED for hobbyists, industrial designers, prototypers, and experimenters. It is designed to allow the easy addition of dynamic indicators, displays, and lighting to existing or new projects.

The BlinkM family of Smart LEDs makes it easy to add dynamic stand-alone lighting no matter what your project.

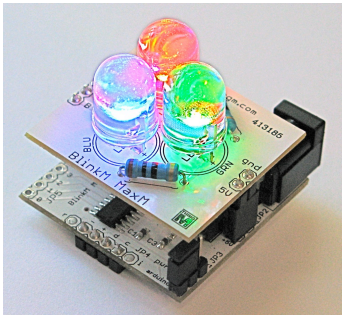
If you’ve used up all your microcontroller PWM channels controlling RGB LEDs and still want more, BlinkM is for you.



BlinkM



BlinkM MinM



BlinkM MaxM

BlinkM Features

- 8000 mcd 140° full-color RGB LED w/ 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
- 18 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

BlinkM MaxM Features

- All the Smart LED features of BlinkM, with additional features of:
- Three high-power 5-12VDC @ 2A PWM outputs to drive large common-anode LED arrays
- Included 445,000 mcd RGB LED cluster
- Four 8-bit analog inputs
- New light script commands to act on inputs for stand-alone dynamics
- Built-in 5V voltage regulator and DC adapter connector for stand-alone operation

BlinkM MinM Features

- All the Smart LED features of BlinkM, includes additional features of:
- Smaller 0.45” package
- Higher-color accuracy 6000mcd SMD RGB LED
- Two digital inputs
- New light script commands for long delays and random delays
- Digital inputs for stand-alone dynamics
- Simple I2C master for syncing BlinkMs

Application Ideas

- Indicators for Prototypes and Industrial Design
- Personalized color accent lights
- Casemod lighting
- Programmable holiday lighting
- Automotive lighting
- Wearables / Smart clothing

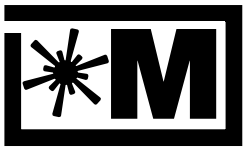
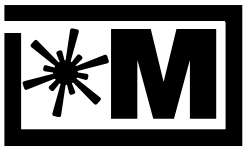


Table of Contents



Differences Between BlinkM, BlinkM MaxM, and BlinkM MinM

BlinkM, BlinkM MaxM, and BlinkM MinM have very similar functionality. Regular BlinkM is treated as the default. Any MaxM-specific differences are noted with a " **MaxM** " icon along the right edge. Any MinM-specific differences are noted with a " **MinM** " icon along the right edge.

Online Resources

The primary site for all things BlinkM is the BlinkM homepage: http://blinkm.thingm.com/.

A support forum is available at http://getsatisfaction.com/thingm/.

Open source examples of how to use BlinkM are available from http://code.google.com/p/blinkm-projects/.

Additional, more DIY-oriented BlinkM information can be found at http://labs.thingm.com/.

Warning: Voltage- and Static-sensitive devices

Both BlinkM and BlinkM MaxM are delicate, static-sensitive devices requiring proper voltage

- Observe proper anti-static techniques when handling them.
- Disconnect all power before connecting or disconnecting BlinkMs to a circuit.
- Apply no more than 5.5 V to the "PWR +" / "5V" input to the voltage inputs
- Do not reverse polarity of the power connections, it could destroy the device.

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.

BlinkM MaxM (or just “MaxM”) is a high-power kind of BlinkM. It functions like a regular BlinkM, but is designed for more advanced, stand-alone installations. It has three high-power MOSFET transistors to drive large LED arrays at 12V and up to 2A, a built-in power supply for the BlinkM microcontroller brain, and four 8-bit analog inputs. This main driver part of MaxM is called the “Master”. The other part of MaxM is a removable LED board called the “Blaster” that contains a blindingly bright RGB LED cluster.

1.1 BlinkM Anatomy and Connections

BlinkM is a 0.6” bare circuit board with a 4-pin 0.1” header connector used to power and program it. On the top is a full-color RGB LED. On the bottom is a tiny computer running at 8MHz.

Figure 1.1: BlinkM layout

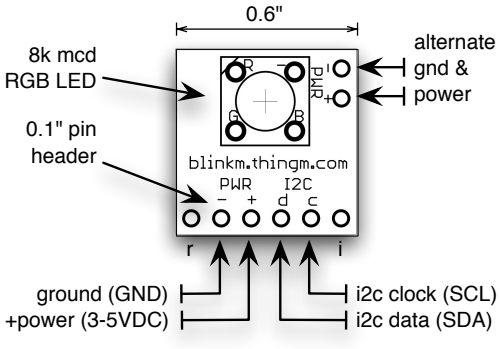
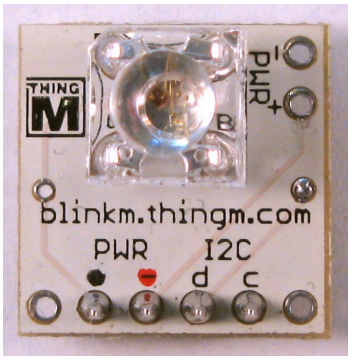


Photo 1.1: BlinkM



1.1.2 BlinkM 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 “SCL”

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 & SCL 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 does reduce color accuracy.

Note: The alternate power and ground holes on the side are provided as a convenience for certain solderless breadboarding layouts. They do not need to be connected to anything if the main power and ground connections are used.

1.2 MinM Anatomy and Connections

MinM

MinM is a 0.4” bare circuit board with 4-holes at 0.1” that the included 4-pin header connector can be press-fit into. Like BlinkM it contains a full-color RGB LED on the top and a tiny 8MHz computer on the bottom. Unlike BlinkM, this LED is a smaller surface mount LED that offers better color mixing.

Figure 1.2: MinM layout

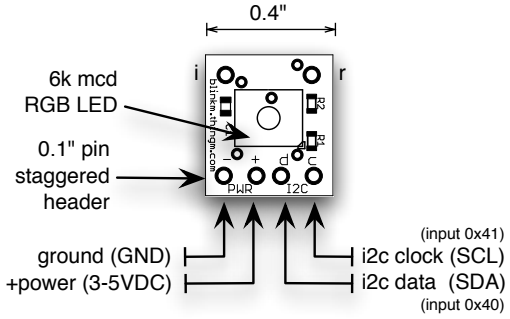
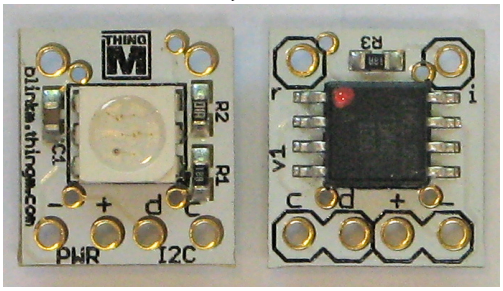


Photo 1.2: MinM, front & back



1.2.1 MinM Connections

The electrical connection details for MinM are exactly the same as a regular BlinkM. Refer to Section 1.1.2 above for connection specifics.

1.2.2. MinM Digital Inputs

MinM

The 'd' and 'c' lines (I2C data and clock) can also function as digital inputs. This allows MinMs to alter light script playback like MaxM. Normally these lines are HIGH, but can be pulled LOW through a 220 ohm resistor. In the input commands ('i' - "Input Read and Jump" and 'I' - "Input Jump Immediate"), these inputs are numbered 0x40 ('d') and 0x41 ('c').

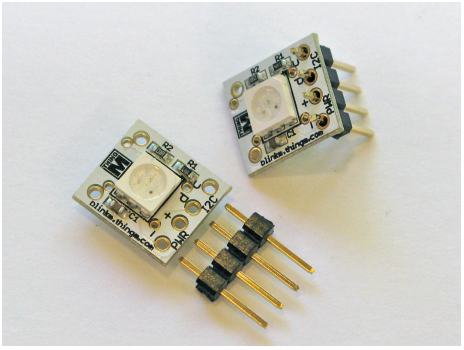
1.2.3. MinM Press-fit header holes

MinM

Unlike regular BlinkM, the included 4-pin header connector is not soldered down on MinM. This allows MinM to be used in wearable and other space-constrained applications. The holes on MinM are staggered slightly to allow the header connector to be press-fit onto MinM for temporary programming of light scripts.

Sometimes this connection isn't perfect however, so you can choose to solder the header on, solder a different connector on, or to lightly press the MinM at an angle to make a better connection.

Photo 1.2.3: MinM press-fit header



1.3 MaxM Anatomy and Connections

MaxM

BlinkM MaxM consists of two boards plugged together: the "Master" driver board and the "Blaster" high-power RGB LED array.

Figure 1.3.1: BlinkM MaxM "Master"

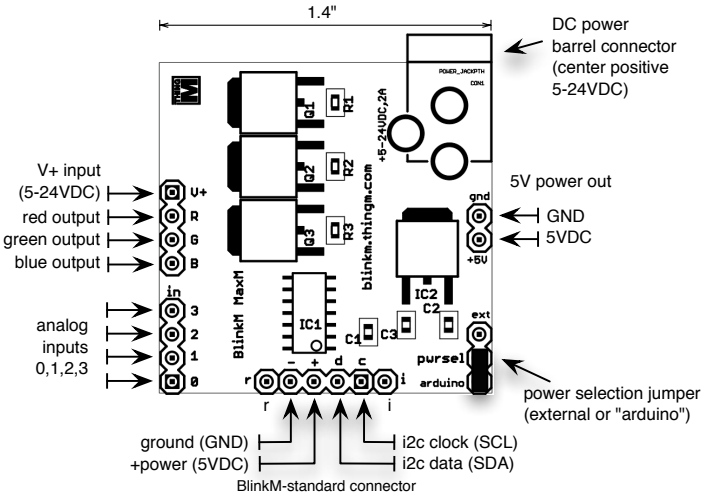


Photo 1.3.1: BlinkM MaxM "Master"

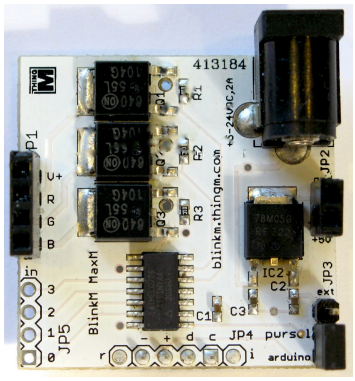


Figure 1.3.2: BlinkM MaxM "Blaster"

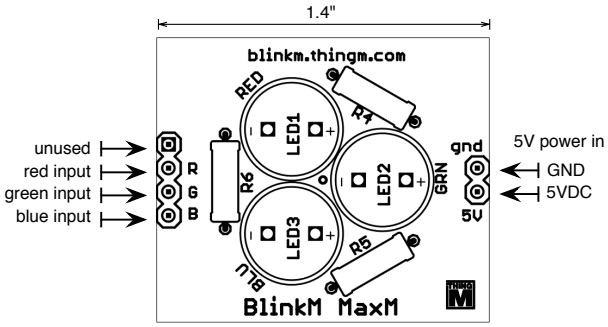


Photo 1.3.2: BlinkM MaxM "Blaster"

1.3.1 BlinkM MaxM Connections

MaxM

The BlinkM Master contains all the connectors needed to interface to a variety of LED arrays, microcontroller platforms, and power sources. The connectors are most accessible after removing the Blaster LED board.

1.3.2 BlinkM-standard Communications Connector

The 4-pin downward facing pin header is the same as on BlinkM, with connections:

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

The MaxM can be powered from this connector, but no power is driven to the “V+” line of the LED Drive header to drive LEDs. The 5V Power header can be used instead. This is how the Blaster part of MaxM is powered.

Note: The original BlinkM connector is designed to plug directly into an Arduino microcontroller board on Analog pins 2,3,4,5. MaxM can be plugged in in this fashion too, but the Blaster board draws too much power to be used in that fashion reliably.

1.3.3 LED Drive Connector Socket

On the right side of the Master is the 4-pin LED drive female header socket. It is designed for common-Anode RGB LED arrays but can be used to drive any three channels of high-power, statically current-limited LEDs. It’s connections are:

V+	Positive voltage to drive LED external LED array (5-12 VDC @ 2A)
R	Red channel drive output
G	Green channel drive output
B	Blue channel drive output

The format of this connector is compatible with a wide range of common-Anode RGB LED arrays from sources such as Sparkfun, DealExtreme, and Ikea’s DIODER.

1.3.4 Input Connector

On the lower left of the Master are four inputs on BlinkM have 8-bit resolution, measuring a voltage ranging between the “PWR +” / 5V power input and GND. Some of the inputs have special functions when used with certain light script commands:

input 0	analog input #0 Red adjust w/ “Knob Adjust RGB” command Hue adjust w/ “Knob Adjust HSB” command
input 1	analog input #1 Green adjust w/ “Knob Adjust RGB” command Saturation adjust w/ “Knob Adjust HSB” command
input 2	analog input #2 Blue adjust w/ “Knob Adjust RGB” command Brightness adjust w/ “Knob Adjust HSB” command
input 3	analog input #3

1.3.5 5V Power Connector Socket

On the right side of the Master is a two pin socket providing +5 VDC and GND connections. It is useful if powering other devices from the MaxM’s internal 5V power supply and is also used to drive the Blaster LED board when it is plugged in.

1.3.6 DC Power Input

At the top right of the Master is a standard 2.1mm DC barrel power connector, center positive. It accepts +5 to +12 VDC, at up to 2 Amps of power. This power is tied directly to the “V+” line of the LED Drive Connector.

1.3.7 Power Selection Jumper / “pwrsel”

The microcontroller brain in MaxM requires a source of +5V DC power to operate. The MaxM power selection jumper allows one to select whether or not the brain is powered by the onboard voltage regulator driven by the DC power connector(“ext”) or from a source of 5V power provided on the 4-pin communications header.

“ext”	5V supplied onboard voltage regulator using power from DC Power Input
“arduino”	5V supplied from 4-pin “BlinkM” Communications connector

2. Getting Started

There are two main ways of using BlinkM (or MinM or MaxM): as a peripheral to an existing device or as a stand-alone object. Stand-alone operation is the common use for BlinkM. To change the light script in a BlinkM, the BlinkM will temporarily be a peripheral to either an Arduino, LinkM adapter, or other controller. This document primarily describes how to connect a BlinkM-class device to an Arduino. To see how to connect it to a LinkM, see the LinkM datasheet at http://linkm.thingm.com/.

2.1 Stand-alone Operation

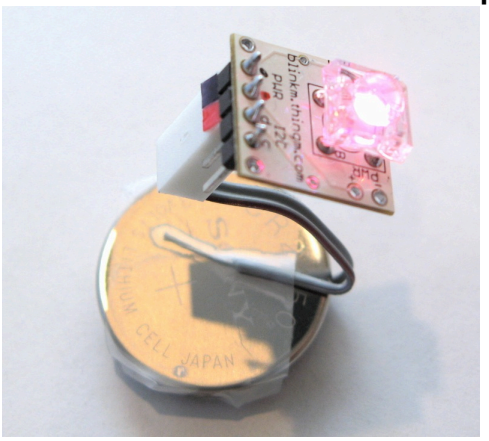
After a light script is programmed into a BlinkM or MaxM, it can be removed from its programmer and used without an external controller. All BlinkMs, MinMs, and MaxMs ship with a default startup light script so one can immediately see how it works.

2.1.1 Stand-alone Operation for BlinkM and MinM

To test basic functionality, connect BlinkM's "`PWR +`" pin to +3-5 VDC and "`PWR –`" pin to ground, as in Photo 2.1.1.

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 4.3 "**Light Scripts**".

Photo 2.1.1: BlinkM Stand-alone Operation

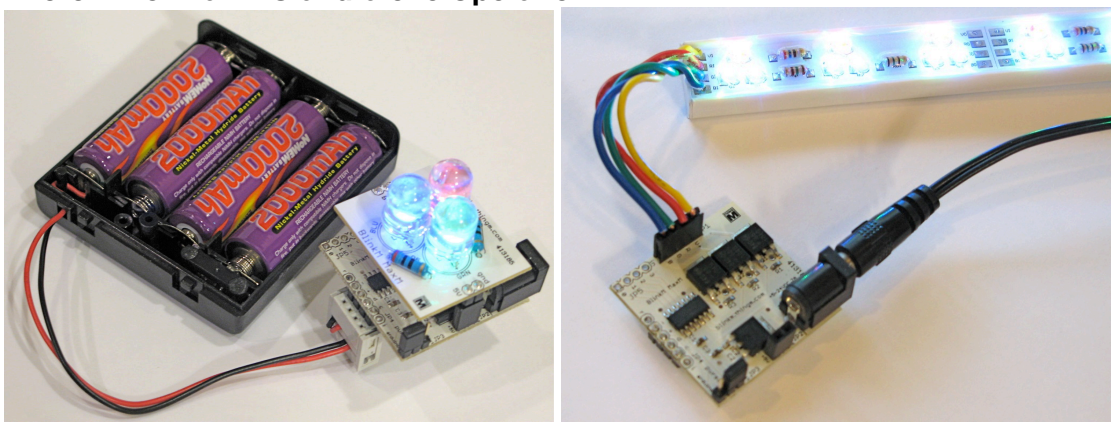


2.1.3 Stand-alone Operation for MaxM

MaxM

MaxM is more directly designed to be used in a stand-alone manner. Its DC power input allows for any input voltage between 5-12 VDC. This voltage is used to directly drive large LED clusters. Or, like BlinkM, it can be powered from its 4-pin BlinkM-compatible connector with a source of 5 VDC. In this mode, the Master part of MaxM can drive its Blaster LED board only.

Photo 2.1.3: MaxM Stand-alone Operation



2.2 Peripheral Operation

To change the light scripts in a BlinkM, it will be operating in peripheral mode to another device.

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, LinkM, 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 and BlinkMs are available from a variety of online retailers, such as FunGizmos (http://fungizmos.com/), SparkFun (http://sparkfun.com/), Adafruit (http://adafruit.com/), and the Make store (http://store.makezine.com/) The discussion below focusses on Arduino, but the techniques apply to any microcontroller with I2C.

2.2.1 Connecting BlinkM or MinM

MinM

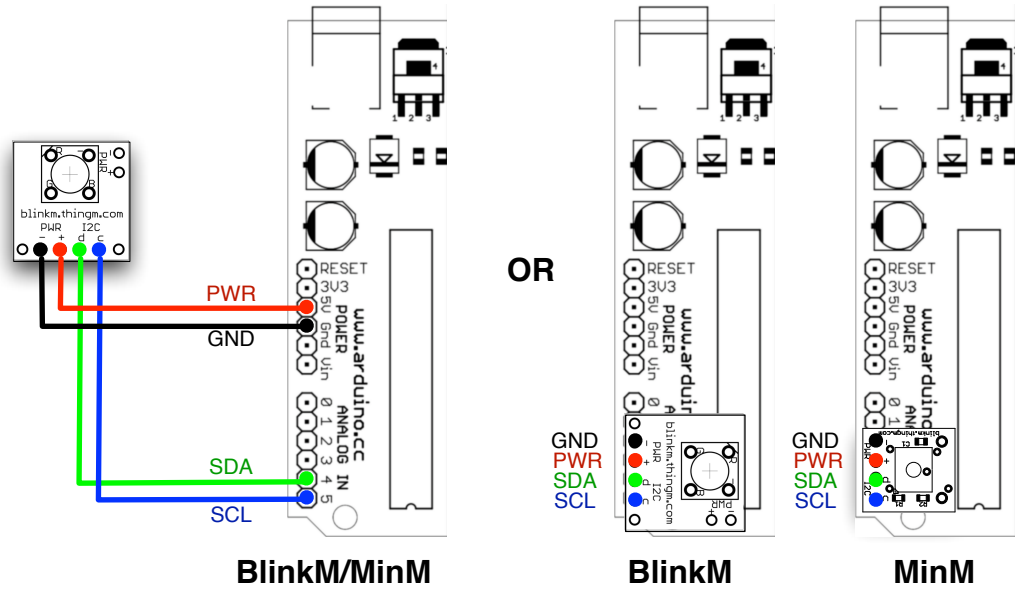
BlinkM needs two wires for power and two for data. Figure 2.2.1a shows the connections needed when hooked up to an Arduino. The Arduino “analog in” pins 4 & 5 also double as the I2C data signal (“SDA”) and clock signal (“SCL”), respectively.

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 or MinM to Arduino



2.2.2 Connecting MaxM

MaxM

Connecting a BlinkM MaxM is virtually the same as connecting a regular BlinkM. Like BlinkM, MaxM’s “BlinkM-compatible connector” needs two wires for power and two for data. Figure 2.2.2 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 (“SCL”), respectively.

Figure 2.2.2: Connecting MaxM & Arduino

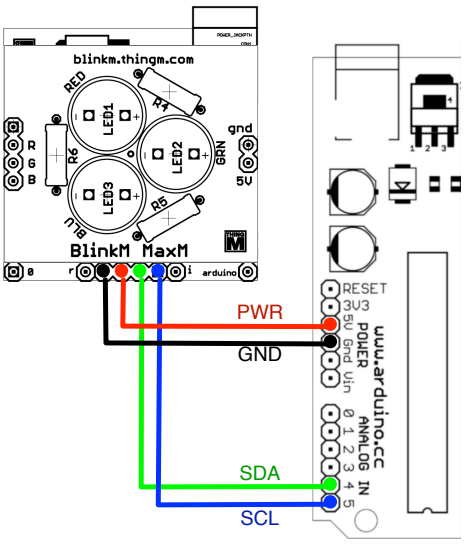


Photo 2.2.2: Connecting MaxM to Arduino

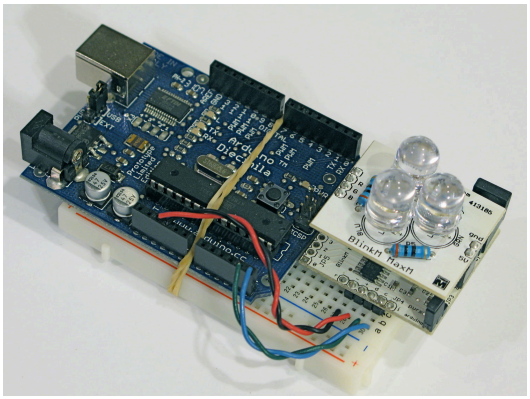
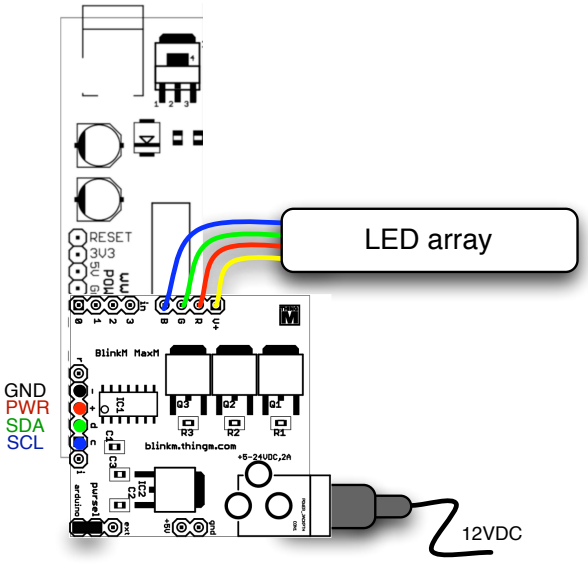


Photo 2.2.2 shows an example method of how to connect a MaxM to an Arduino. Note that unlike a BlinkM, a MaxM cannot be plugged directly into Arduino's Analog In pins 2,3,4,5 because the MaxM draws too much power.

However if the MaxM is powered from an external 12V power source, it can be plugged in like a BlinkM as in Figure 2.2.3.

Figure 2.2.3: Connecting MaxM Master-only to Arduino



2.3 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
```

For more details on the BlinkM command language, see Section 3, "**BlinkM Commands**" below. Several examples of how to communicate with BlinkM written in Arduino, Processing, Max/MSP and Basic Stamp are available from http://blinkm.thingm.com/.

2.4 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 2.4 shows what it looks like.

If you have a LinkM, ThingM's USB-to-I2C adapter made for use with the BlinkM family of Smart LEDs, then you can use the Multitrack Sequencer (aka "BlinkMSequencer2") as seen in Figure 2.4.1. The Multitrack Sequencer can be downloaded from http://linkm.thingm.com/.

Figure 2.4: BlinkM Sequencer for Arduino

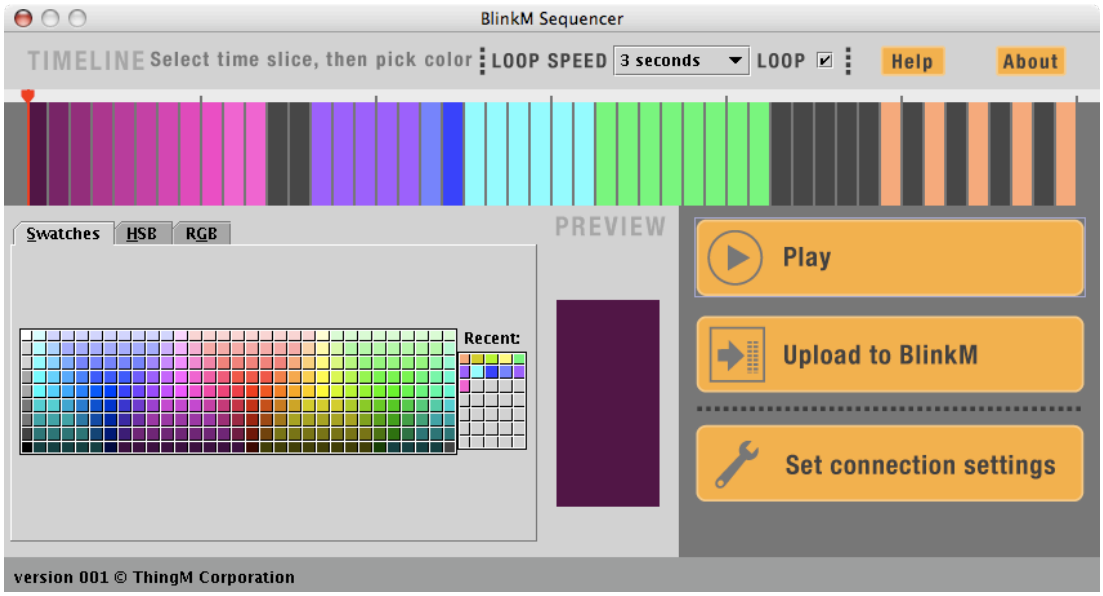
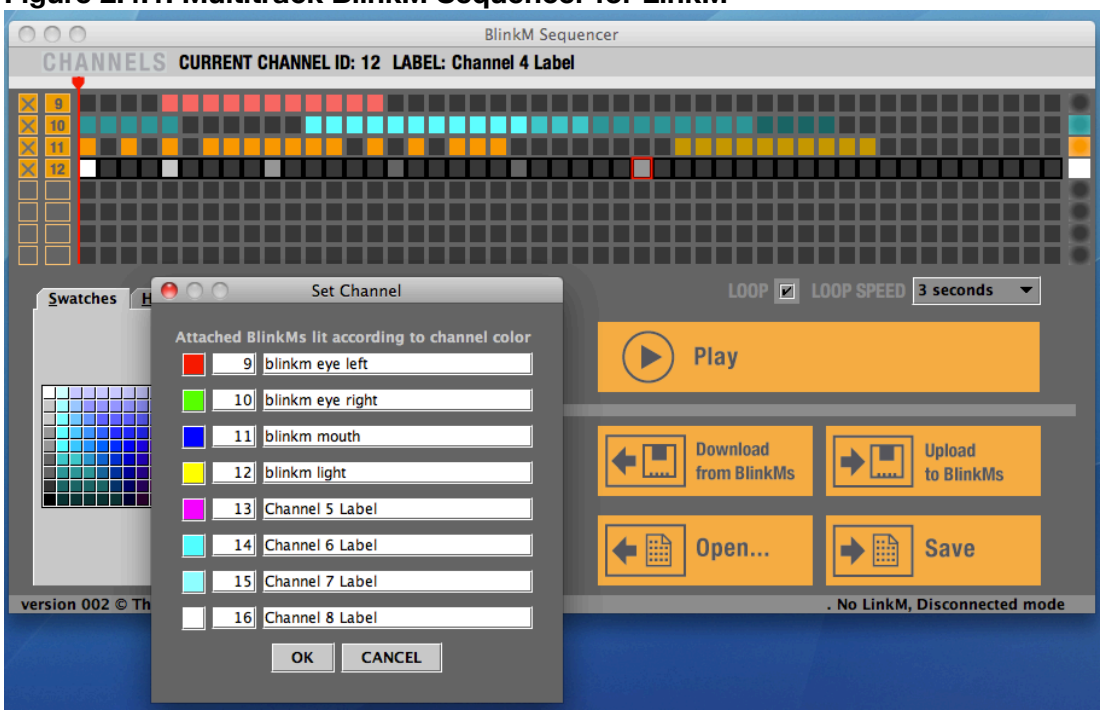


Figure 2.4.1: Multitrack BlinkM Sequencer for LinkM



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.

See "4.1 I2C Addressing" under "BlinkM Concepts" for more information about how BlinkM handles I2C addressing.

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. The command "format" listed on the right edge 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.

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	5	0	{'B',m,n,r,f,t}

New MaxM/MinM commands					**MaxM / MinM***
Knob Read RGB	k	0x6b	3	0	{'k',R,G,B}
Knob Read HSB	K	0x4b	3	0	{'K',H,S,B}
Jump, relative	j	0x6a	1	0	{'j',j}
Input Read & Jump	i	0x69	3	1	{'i',i,v,j}
Input Jump Immediate	I	0x49	3	0	{'I',i,v,J}

New MinM commands					MinM
Wait (long duration pause)	w	0x77	2	0	{'w',l,h}
Random Time Delay	T	0x54	1	0	{'T',t}
Send/Sync (I2C Master)	s	0x73	3	0	{'s',c,a,b}

* Note: The input commands only work on MinMs (and newer BlinkMs) with firmware version {'a','d'} and above.

3.3 Command Details

Each command below is listed with its "format" overview. This "format" is similar to how byte arrays are defined in Java (e.g. Processing) and C (e.g. Arduino). For example, the following is valid code for defining a "fade to RGB color" command in Arduino:

```
byte R=0xff, G=0xcc, B=0x33;
byte cmd = {'c', R,G,B};
```

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.

Note that this command only works after a previous 'h' command has been used to set an initial hue.

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

Examples:

```
{'h', 0x00,0xff,0xff}   // set color to bright red
{'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	blue flash	blue→off
6	cyan flash	cyan→off
7	magenta flash	magenta→off
8	yellow flash	yellow→off
9	black	off
10	hue cycle	red→yellow→green→cyan→blue→purple
11	mood light	random hue→random hue
12	virtual candle	random yellows
13	water reflections	random blues
14	old neon	random orangeish reds
15	the seasons	spring colors→summer→fall→winter
16	thunderstorm	random blues & purples→white flashes
17	stop light	red→green→yellow
18	morse code	S.O.S in white

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 and returns no value.

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: 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. This command takes approximately 20 milliseconds to complete, due to EEPROM write time.

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 takes approximately 15 milliseconds to complete, due to EEPROM write times.

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 takes approximately 15 milliseconds to complete, due to EEPROM write time and I2C stack reset.

See "4.1 I2C Addressing" for more details about how BlinkM handles I2C addresses.

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: {v,w}

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

Current BlinkM firmware versions:

{'a','a'}	BlinkM (original)

{‘a’,’b’}	BlinkM MaxM
{‘a’,’c’}	BlinkM MinM (original)
{‘a’,’d’}	BlinkM MinM (updated) and BlinkM (updated)
{‘b’,’a’}	CtrlM

Examples:

```
{‘Z’}                    // get version
```

Set Startup Parameters

format: `{‘B’,m,n,r,f,t}`

This command sets the startup (or “boot”) action for BlinkM. The command takes four arguments. The first argument ‘m’ is the startup mode: 0 means do nothing, 1 means play a script. The second argument ‘n’ is which script id to play. The third argument ‘f’ is the number of repetitions to play that script id. The fourth (‘f’) and fifth (‘t’) arguments are the fade speed and time adjust, respectively, to use with the script. This command takes about 20 milliseconds to complete, due to EEPROM write time.

Note: when turning off playing a script by setting the first argument ‘m’ to 0, the other arguments are saved but not loaded on startup and instead set to zero. This is most noticeable with the fade speed value. Thus if a “{‘B’,0,...}” is issued to disable startup script playing, be sure to issue a “{‘f’, 20}” command after BlinkM startup or color fading will not work.

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}
```

3.4 Command Details for new MaxM & MinM Commands

In addition to the standard set of BlinkM commands, MaxMs and MinMs have some additional commands dealing with input reading and light script flow control.

MaxM

Knob Read RGB

format: `{‘k’,R,G,B}`

This command allows inputs #0,#1,#2 to directly control R,G,B color output, as if those three inputs were connected to potentiometer “knobs”. Actual physical knobs are not required, in

fact buttons or switches can be used (See Section 4.5 “Reading Inputs with MaxM”), but it’s useful to think of these commands as having knobs.

The three arguments determine how much (what percentage, essentially) a given input can control the respective output, with 0xff enabling full control, 0x00 disabling control, and values in between for varying amounts of control.

This command does not return a value.

Note: This command is only valid in a light script and is not valid over I2C.

Note: This command is executed only at the time is occurs in the light script. If continuous adjustment is desired, a loop with the desired update rate should be created. (See the ‘j’ “Jump, relative” command)

Note: On MinM, this command is present but is of limited use due to MinM’s digital inputs shared with the I2C pins. **MinM**

Examples:

```
// Allow input #0 to fully adjust red channel, and
// allow input #2 to half adjust blue channel
// format: {dur, {cmd,arg1,arg2,arg3}}
{1, {‘k’,0xff,0x00,0x80}}  // input #0 and #2 control Red & Blue
// light script automatically loops back to first line
```

MaxM

Knob Read HSB

`format: {‘K’,H,S,B}`

This command is the HSB version of the previous RGB command. It allows inputs #0,#1,#2 to directly control H,S,B color output. The three arguments determine how much (what percentage, essentially) a given input can control the respective output, with 0xff enabling full control, 0x00 disabling control, and values in between for varying amounts of control.

Note that because HSB and RGB color spaces are largely independent in BlinkM (translation from HSB to RGB occurs, but not RGB to HSB), be sure to use the ‘h’ (“Fade to HSB”) command before using this command.

This command does not return a value.

Note: This command is only valid in a light script and is not valid over I2C.

Note: This command is executed only at the time is occurs in the light script. If continuous adjustment is desired, a loop with the desired update rate should be created. (See the ‘j’ “Jump, relative” command)

Note: On MinM, this command is present but is of limited use due to MinM’s digital inputs shared with the I2C pins. **MinM**

Examples:

```
// Set color to bright red,
// then only allow input #0 to adjust brightness
// format: {dur, {cmd,arg1,arg2,arg3}}
{1, {'h',0x01,0xff,0xff}}  // set hue to fully saturated bright red
{1, {'k',0xff,0x00,0x00}}  // input #0 controls brightness
// light script automatically loops back to first line
```

MaxM MinM

Jump, relative

format: {'j',j}

This command jumps a relative amount (positive or negative) in the currently playing light script. It can be used to divide a light script into a "setup" section and a "loop" section (like Arduino & Processing), or allow the creation of multiple mini-scripts inside of a single larger light script. The mini-scripts can be independently addressed using the position argument of the 'p' "Play Script" command. It is also useful for creating loops for reading inputs.

This command does not return a value.

Note: This command is only valid in a light script and is not valid over I2C.

Examples:

```
// create two independent light mini-scripts,
// one blinks white on/off, the other does RGB color fade
// to play one, send {'p',0,0,0}}, the 2nd with {'p',0,0,5}
// format: {dur, {cmd,arg1,arg2,arg3}}
// mini-script one
{ 1, {'f', 100,0x00,0x00}},  // set fade speed to 100
{25, {'c', 0xff,0xff,0xff}}, // full white
{25, {'c', 0x00,0x00,0x00}}, // all off
{ 1, {'j', -2,  0,   0   }}  // jump back two
// mini-script two
{  1, {'f',  7, 0x00,0x00}},  // set fade speed to 7
{100, {'c', 0xff,0x00,0x00},  // fade to red
{100, {'c', 0x00,0xff,0x00},  // fade to green
{100, {'c', 0x00,0x00,0xff},  // fade to blue
{ 1,  {'j',  -3, 0,   0   }}  // jump back three
```

MaxM MinM

Input Read & Jump

format: {'i',i,v,j}

This command has two flavors. Over I2C, it returns the value of the four inputs as four bytes. In a light script, it reads analog input number 'i', and if its value is above value 'v', the light script jumps the amount given by 'j'. The jump amount is treated the same as "Jump, relative". The "Input Read & Jump" command is a conditional statement for light scripts, allowing branching or looping depending on input. It is roughly equivalent to the C code:

```
if( inputs[i] > v ) script_position += j;
```

For more information on MaxM input reading circuitry, see Section 4.5 "Reading Inputs with MaxM". For more information on MinM inputs, see Section 4.6 "Reading Inputs with MinM".

This command does not return a value.

Note: The input is only read at the time the command is evaluated in the light script. So, if a light script contains many lines with long durations, you may not get the responsiveness you require. For immediate responsiveness, use the "Input Jump Immediate" command.

Note: When this command is received over I2C, it returns the value of the four inputs as four bytes.

Note: For MinM, the two valid input numbers are 0x40 ('d' pin) and 0x41 ('c' pin). **MinM**

Examples:

```
// display constant red, when input 1 goes > 0xA0 (NC button push)
// do a little glimmer, then go back to being constant red
// format: {dur, {cmd,arg1,arg2,arg3}}
{  1, {'f',   40,0x00,0x00}}, // set fade speed to 40
{  1, {'c', 0xff,0x00,0x00}}, // red
{  1, {'i',    1,0xA0,  -1}}, // on input 1, jump back one if > A0
{ 10, {'c', 0xff,0xff,0xff}}, // else, do this: white
{  5, {'c', 0x33,0x00,0xff}}, // blueish purple
{  2, {'c', 0xff,0xff,0xff}}, // white
{  5, {'c', 0x00,0x00,0xff}}, // blue
{  7, {'c', 0xff,0xff,0xff}}, // white
```

MaxM MinM

Input Jump Immediate

format: {'I',i,v,J}

In contrast to the "Input Jump & Read" command, which is only evaluated when it executed in a light script, this command sets up a global real-time input test that will jump to an absolute light script address when the test succeeds. It allows quick response for those light scripts that are based on user inputs rather than sensors, or for detecting transient input events.

It reads analog input number 'i', tests the value received against value 'v' and if it is greater, jumps to light script position 'J'. It is roughly equivalent to the C code:

```
if( inputs[i] > v ) script_position = j;
```

Only one global test is allowed at a time, and any subsequent instances of the command replaces the previous one. To turn off this command (usually the first command at the destination position), set its three arguments to 255 (0xff), e.g.: `{'I',255,255,255}.`

For more information on MaxM input reading circuitry, see Section 4.5 "Reading Inputs with MaxM"

This command does not return a value.

Note: This command is only valid in a light script and is not valid over I2C.

Note: For MinM, the two valid input numbers are 0x40 ('d' pin) and 0x41 ('c' pin). **MinM**

Examples:

```
// a two-mode light script
// mode 1 is a slow red-green-blue fading loop
// mode 2 is a flash to bright white, then fade to black
// a switch on input #1 chooses between the two
{0,  {'I',0,0xA0,6}},       // line 0: set global test: if #1 > 0xA0
{0,  {'f',20,0,0}},         // line 1: set fade speed to 20
{15, {'c',0xff,0x00,0x00}}, // line 2: fade to red
{15, {'c',0x00,0xff,0x00}}, // line 3: fade to blue
{15, {'c',0x00,0x00,0xff}}, // line 4: fade to green
{0,  {'j',-3,0,0}},         // line 5: jump back 3
{0,  {'I',255,255,255}},    // line 6: turn off "I" test
{0,  {'f',100,0,0}},        // line 7: set fadespeed to a fast 100
{2,  {'c',0xff,0xff,0xff}}, // line 8: white
{0,  {'f',20,0,0}},         // line 9: set fade to a slow 20
{20, {'c',0x00,0x00,0x00}}, // line10: fade to black
// light script by default loops back to line 0
```

3.5 Command Details for new MinM Commands **MinM**

In addition to the new commands in Section 3.4, MinMs also have a few new commands specific to them.

MinM

Wait (long duration pause) format: {'w',l,h}

Normally, the time between events in a light script can be most 255 ticks. At 30 ticks per second, this is 8.5 seconds. For longer durations, the "Wait" command can be used to insert some number of five-second delays. The complete argument to "Wait" is a 16-bit number, representing the number of five-second durations to wait. The low byte the first argument and

the high byte is the second argument. With the 16-bit number, this means you can have delays up to 227 days if you really wanted.

Example:

```
{1,  {'c',0xff,0x00,0xff}}, // change to purple
{1,  {'w',12,0,0}},         // wait 60 seconds
{1,  {'c',0xff,0x00,0x00}}, // change to red
{1,  {'w',104,1,0}},        // wait 1800 seconds (30 minutes)
```

Random Time Delay

format: {‘T’,t}

To give more organic changes in light patterns, a random time delay can be added in between two light script commands. This command modifies the base duration of its light script line a random amount, based on its single argument.

Example:

```
{1,  {'c',0xff,0x00,0xff}}, // change to purple
{100,{‘T’,10,0,0}           // randomize duration 10 ticks around 100
{1,  {'c',0xff,0x00,0x00}}, // change to red
```

Send/Sync (I2C Master)

format: {‘s’,c,a,b}

Because each BlinkM device has its own internal clock that is not synced to a shared clock, multiple BlinkMs will get out of sync with each other, even when running the same light script.

One solution to combat this is to have an external I2C master like an Arduino or LinkM periodically sync the BlinkMs. Having an extra controller is bulky however. Now MinMs can be simple I2C masters.

Use the “Send/Sync” command to let MinM become an I2C master for a moment. Sub-commands of this are:

{‘s’,'p’,n,p} — send Play Script, id #n, position p

{‘s’,'o’,0,0} — send Stop Script

{‘s’,'f’,f,0} — send Set Fadespeed with fadespeed f

{‘s’,'t’,t,0} — send Set TimeAdj with timeadj t

{‘s’,'a’,a,0} — change i2c address to send on (normally 0)

Note: When using this command be sure to have a delay at the start of the light script otherwise the BlinkM will take over the bus and you won’t be able to communicate with it.

Note: When connecting multiple BlinkMs, two pull-up resistors still need to be connected between ‘d’ & ‘+’ and ‘c’ & ‘+’ as in Figure 5.1.

Example:

```
{10, {'c',0xff,0x00,0xff}}, // change to purple
{10, {‘w’, 12, 0, 0}},      // wait 60 seconds
{10, {'s',’p’,0,0}},        // tell other BlinkMs to play script #0
```

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.

Note: While I2C addresses are 7-bits long and thus can range from 0 to 127, some environments use a "left-shifted" display of I2C addresses. These shifted addresses range from 0-254, but only exist for even address values (0,2,4,6,...). The left-shifted version came about because the address gets shift left by one bit upon transmission. (Left-shifting by one bit is the same as multiplying by 2) Like Arduino, BlinkM uses the non-shifted 0-127 format of I2C addresses. The default BlinkM address of 9 (0x09) looks like address 18 (0x12) when used with the left-shifted style of addressing.

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

For MaxM, there additional new commands can also be in a light script: **MaxM**

- "k" – Knob Adjust RGB
- "K" – Knob Adjust HSB
- "j" – Jump, relative
- "i" – Input Read & Jump
- "I" – Input Jump Immediate

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. (0 to 255 in MaxM) 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 immediately start playing the new script.

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"). Also see the BlinkMScriptTool Processing sketch in the example code bundle.

4.3 Color Models

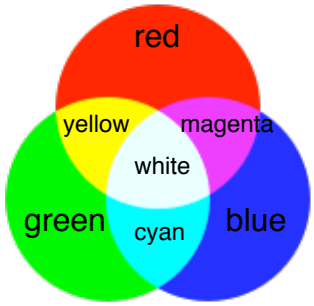
BlinkM supports two different color models: RGB and HSB. RGB is the color model most people are familiar with. It's used to specify colors on web pages. The color "#FF0088" (a reddish purple) are the three components of red, green, and blue. The HSB color model uses one number for color, or hue, and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

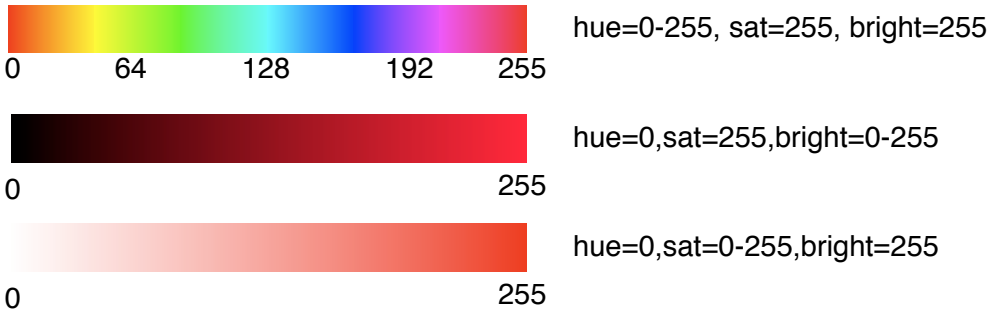


4.3.2 About the HSB Color Model

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 and BlinkM MaxM use 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), sending "Play Script" or similar commands over I2C, or in the case of MaxM, writing a light script to trigger off input changes.

4.5 Reading Inputs with MaxM

MaxM

Each of MaxM's 4 analog inputs produce a value from 0-255 (0x00-0xff), based on a voltage that ranges from GND to "PWR +" or 5V (depending on the setting of the "pwrsel" jumper). Any voltage within that range can be read and acted upon with the light script commands "Knob Adjust RGB", "Knob Adjust HSB", "Input Read & Jump" and "Input Jump Immediate" to create dynamic lighting when MaxM's in stand-alone mode.

There are many sensors that can be interfaced to MaxM's inputs. If a sensor is advertised with having "5V TTL-compatible outputs" then it will likely work with MaxM. The efforts of the Arduino and Basic Stamp communities can be used as a source of inspiration here, as they have done much research on creating input devices and sensors.

The 4-pin input header is purposefully not populated to allow a wide range of input connection techniques. For experimenting with a wide range of inputs, a 4-pin female header socket can be soldered to the input connector. To test out a new sensor or user-interface hooked to MaxM, the 'i' command over I2C will return the values of the four inputs as four 8-bit numbers. The BlinkMTester Arduino sketch gives an example of this functionality.

Two of the easiest sensors to add is a potentiometer knob (an analog control) and a button (a digital control). Figure 4.5 shows how to connect a potentiometer ("pot") to MaxM. The exact value of the pot is not critical, but should be over 5k ohm and below 100k ohm.

Figure 4.5: Connecting a Potentiometer Knob to MaxM input #2

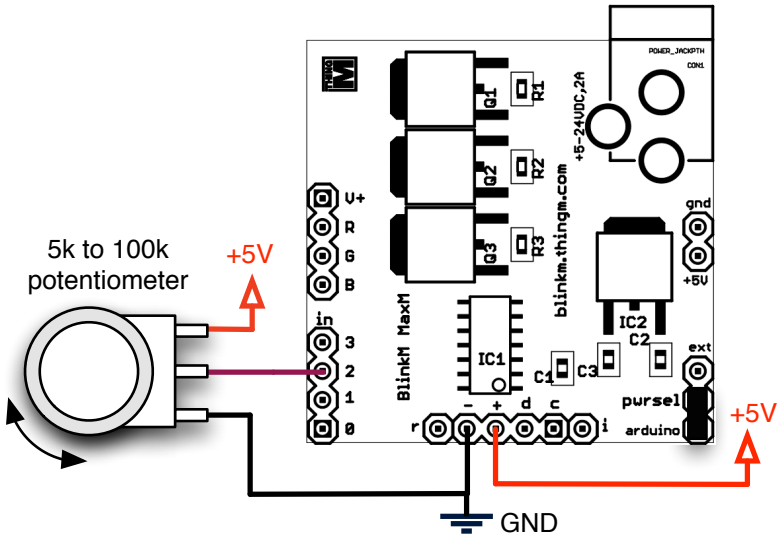
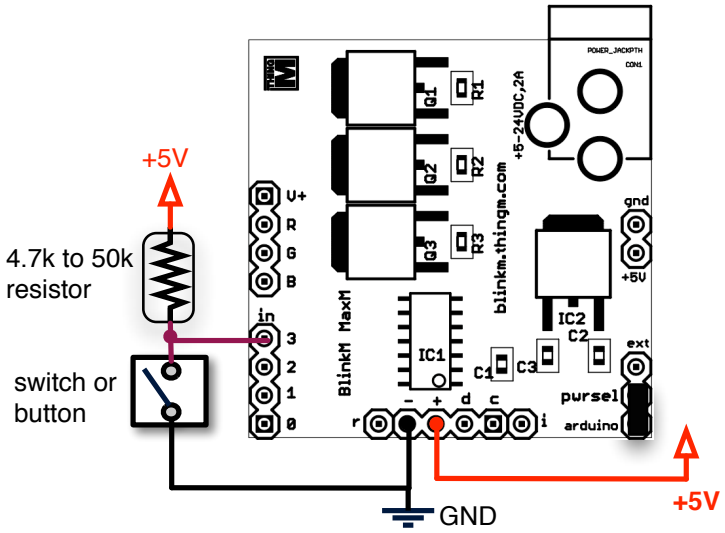


Figure 4.5.1 shows how to add a button to create a digital control. The resistor is used to set the default (non-pressed) value to be the highest possible voltage. Reading the input without pressing the button would give a value around 255 (0xff). The button is also wired so that when it is pushed, it overrides the resistor to produce the lowest possible voltage, which would read as being near 0. The exact value of the resistor is not critical, but should be above 4.7k and below 100k.

Figure 4.5.1: Connecting a Button to MaxM input #3

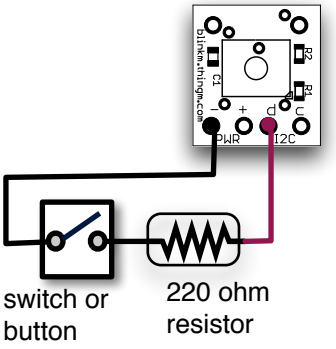


4.6 Reading Inputs with MinM

MinM

MinMs can use their ‘d’ and ‘c’ I2C pins as digital inputs. Because they are shared with I2C, they will normally read HIGH but can be pulled LOW through a 220 ohm resistor. Figure 4.6 shows a diagram of an example button added to MinM.

Figure 4.6: Connecting a Button to MinM input #0x40



5. Other Circuits

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

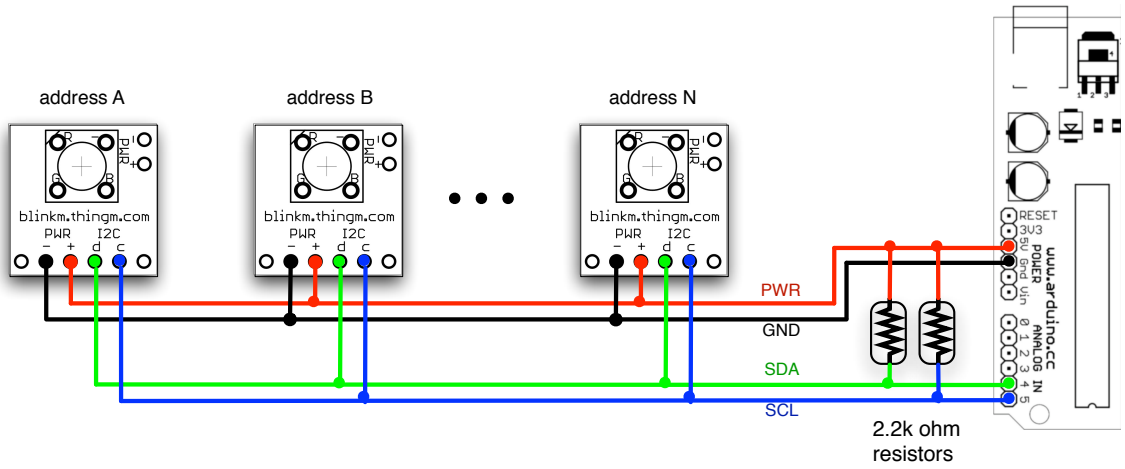
5.1 Connecting Multiple BlinkMs

BlinkM communication is done via I2C, a simple network protocol that can have up to 127 devices with just two lines, SDA (serial data) and SCL (serial clock). To add multiple BlinkMs to a circuit, connect their I2C SDA and SCL together and run those two lines to the SDA and SCL pins of the I2C master controller. Figure 5.1. shows an example of multiple BlinkMs connected to an Arduino. The Arduino is the I2C master. The two 2.2k “pull-up” resistors are necessary to help the Arduino talk to multiple BlinkMs.

If independent control of each BlinkM is required, set each ones address to a unique number between 1 and 127 using the “Set BlinkM Address” (‘A’) command. If multiple BlinkMs are connected when the “Set BlinkM Address”, they will all have the new address. To prevent this, power down the other BlinkMs or temporarily set their RESET (‘r’) pins to Gnd to prevent them from listening to I2C commands.

When multiple BlinkMs are connected and all have unique addresses, it is still possible to command them simultaneously using the special “general call” address 0x00. This broadcast address works with any BlinkM command that doesn’t return a response.

Figure 5.1: 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. Note: voltages below approximately 3.8V will not be sufficient to reliably turn on the blue and green LEDs of 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

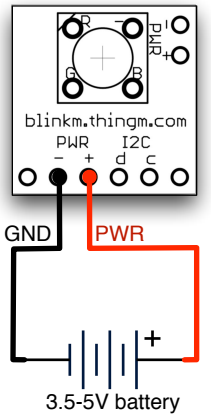
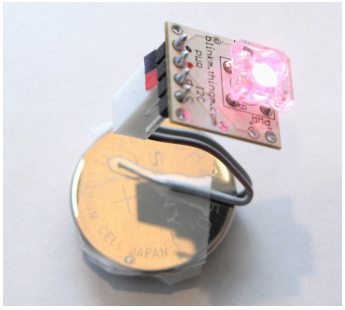


Photo 5.3: Battery Powered BlinkM



5.3.1 Battery Powered MaxM

MaxM

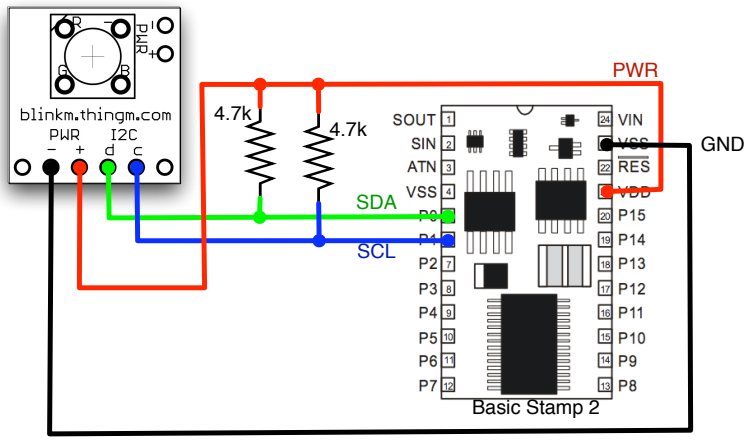
When using a MaxM in Master+Blaster configuration, it is so bright and draws so much power that only somewhat large battery packs should be used. A good portable configuration is a 4xAA battery pack which should give several hours of operation.

The brain on the MaxM Master board draws about 2x the amount of power as a regular BlinkM. Thus, it is possible to power a MaxM Master from a small battery as above, with a larger power source for the LED drivers and LED array. However, since MaxM contains an on-board 5V voltage regulator, it usually easiest to power both the MaxM and the LEDs from the same power source.

5.3 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” (SCL) line is connected to Basic Stamp P1. See http://blinkm.thingm.com/ for code examples.

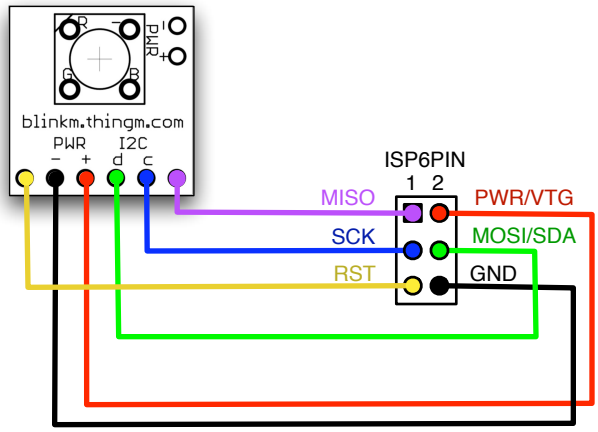
Figure 5.3: Connecting BlinkM to a Basic Stamp 2



5.4 Reprogramming BlinkM's Flash Memory

BlinkM and MaxM can also be used as a general AVR ATtiny45 development board. The ATtiny45/85 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 for download 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 example code download.

6.1.4 Programming Light Scripts

Light scripts are the most complex aspect of talking to BlinkM, and are optional if BlinkMs are controlled in real-time from another processor. Light scripts do however allow the reduction of BlinkM-related overhead by bundling up an often-repeated series of BlinkM commands into one command.

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_stopScript(addr1);                // stop startup script
BlinkM_stopScript(addr2);                // stop startup script
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
```

6.1.6 Using Jump Statements

MaxM

The “Jump, relative” command is useful by itself as a way to create multiple mini light scripts within one larger light scripts. By creating several loops in the light script, each loop can be independently accessed with the “position” argument of the “Play Script” command. For example, the following light script contains two mini-scripts of different lengths and with different fade speeds.

```
// to play one, send {‘p’,0,0,0}}, the 2nd with {‘p’,0,0,5}
// format: {dur, {cmd,arg1,arg2,arg3}}
// mini-script one
{ 1, {'f', 100,0x00,0x00}},  // set fade speed to 100
{25, {'c', 0xff,0xff,0xff}}, // full white
{25, {'c', 0x00,0x00,0x00}}, // all off
{ 1, {‘j’, -2,  0,   0   }}  // jump back two
// mini-script two
{  1, {'f',  7, 0x00,0x00}},  // set fade speed to 7
{100, {'c', 0xff,0x00,0x00},  // fade to red
{100, {'c', 0x00,0xff,0x00},  // fade to green
{100, {'c', 0x00,0x00,0xff},  // fade to blue
{ 1,  {‘j’,  -3, 0,   0   }}  // jump back three
```

The “Jump, relative” command is also useful with the input commands as a means of providing an “else” to the if-like conditional they provide.

6.1.7 Input Handling Logic

MaxM

The electrical details of hooking up input devices to MaxM is described in Section 4.5 “Reading Inputs with MaxM” and the two input commands “Input Read & Jump” and “Input Jump Immediate” are described in their sections. This section is to describe some of the logic constructs available with these commands.

A common programming form is the “if-else” statement: “if a condition occurs, do action A, else do action B”. In MaxM, the “condition” is whether or not a particular analog input’s value is above a certain user-defined threshold. With this, one can create a branch in the flow of the light script for different behavior to occur. For example:

```
{ 1, {‘h’, 0x80,0xff,0xff}}, // set hue
{ 1, {'K', 0x00,0x00,0xff}}, // only let input 3 control brightness
{ 1, {'i', 3,   0x40,   2}}, // if input #3 > 0x40, jump +2 (to red)
{ 1, {'j',-2,   0,      0}}, // else, jump -2 (back to 'K'nob)
{ 1, {'h', 0x00,0xff,0xff}}, // red
```

An extension of this is to create a series of “if-else” statements to divide the analog input value into a discrete set of values. Order is important however, make sure largest values are first:

```
{ 1, {'i', 3,  192,  8}}, // if input #3 > 192, jump +8
{ 1, {'i', 3,  128, 13}}, // if input #3 > 128, jump +13
{ 1, {'i', 3,   64, 17}}, // if input #3 > 64, jump +17
{ 1, {'i', 3,    0, 21}}, // if input #3 > 0, jump +21
```

7. Electrical Characteristics

7.1 BlinkM and MinM

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 3.8V.

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

7.2 BlinkM MaxM

symbol	parameter	Condition	min	typ	max	units
Vcc	Operating Voltage		3*	5	5.5	V
Icc	Power Supply Current	LED full dark			5	mA
		LED full bright			250	mA
		RESET held low			4	mA
V+	LED Drive Voltage		5		12	V
I+	LED Drive Current				2	A

Electrical characteristics for inputs and I2C connectivity are the same as those for Atmel's ATtiny44 AVR microcontroller. See http://atmel.com/avr/ for more details.

8. Schematics

Figure 8.1: BlinkM and BlinkM MinM

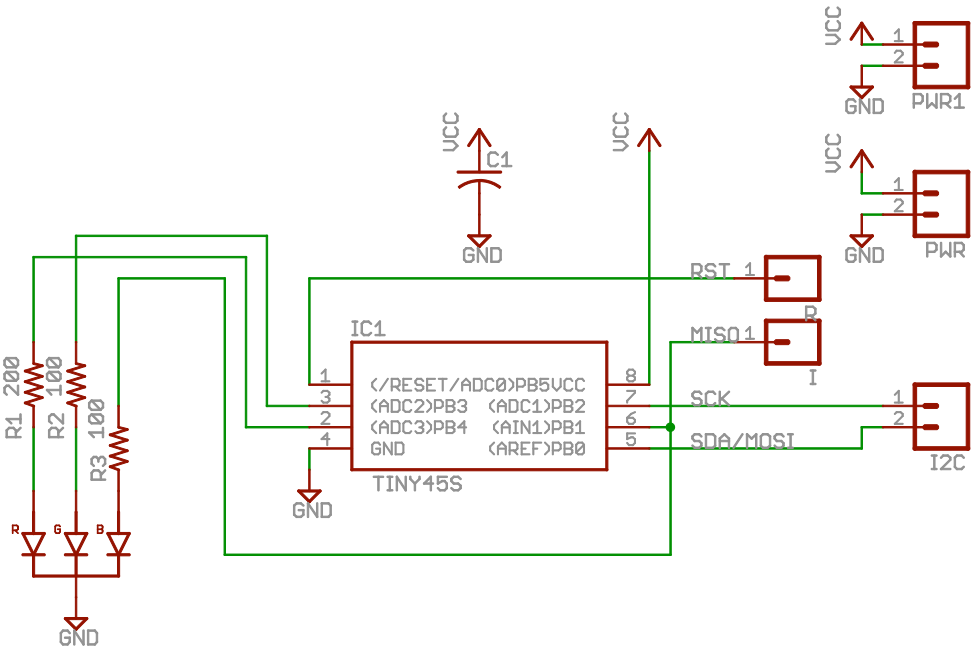


Figure 8.2: BlinkM MaxM "Master" driver board

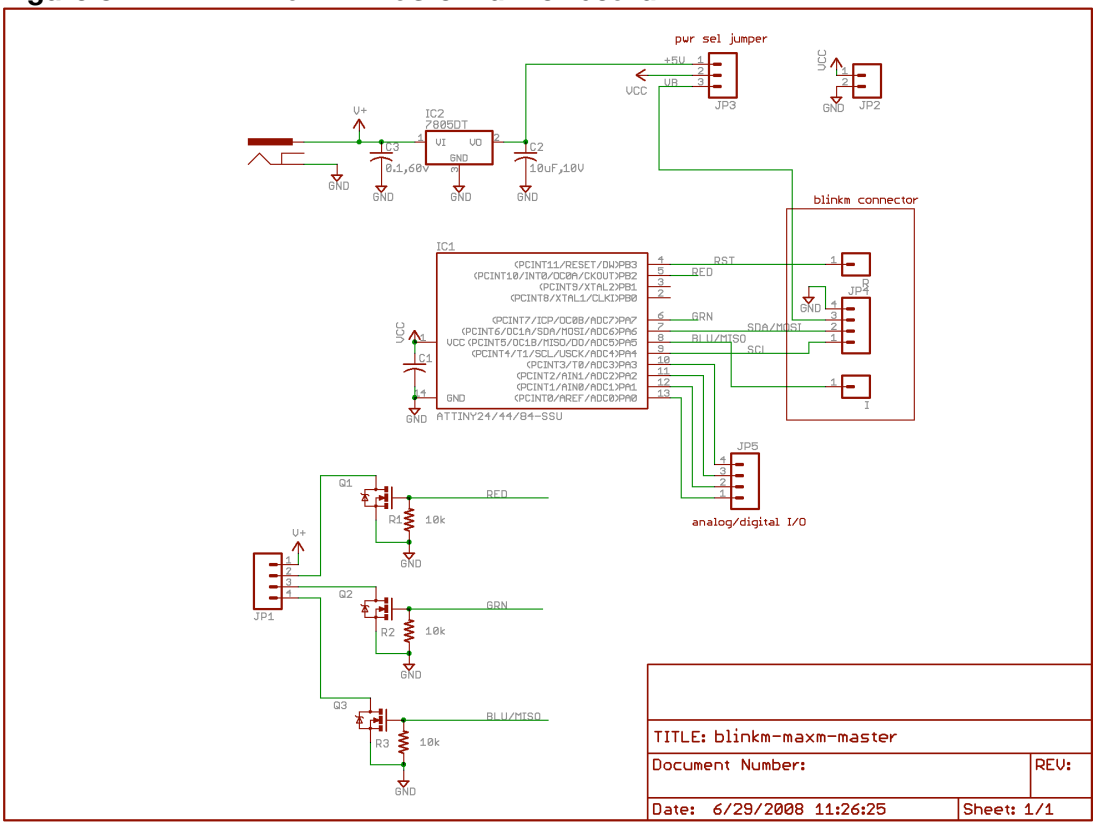
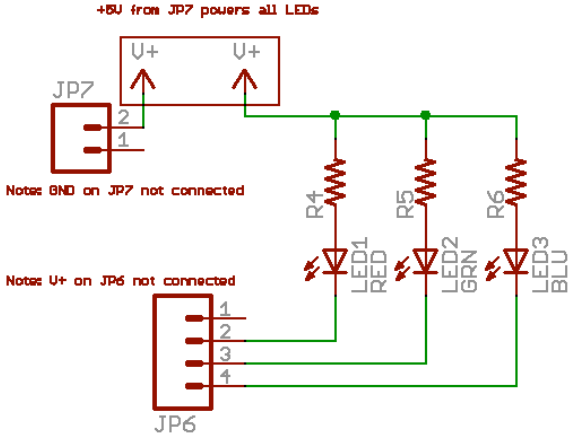


Figure 8.3: BlinkM MaxM "Blaster" LED board



9. Packaging Information

All units in inches.

Figure 9.1: BlinkM Packaging Information

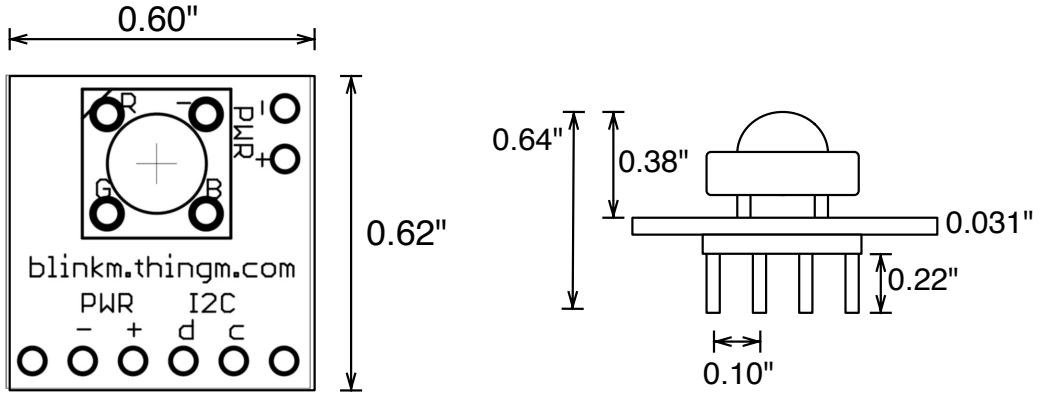


Figure 9.2: MinM Packaging Information

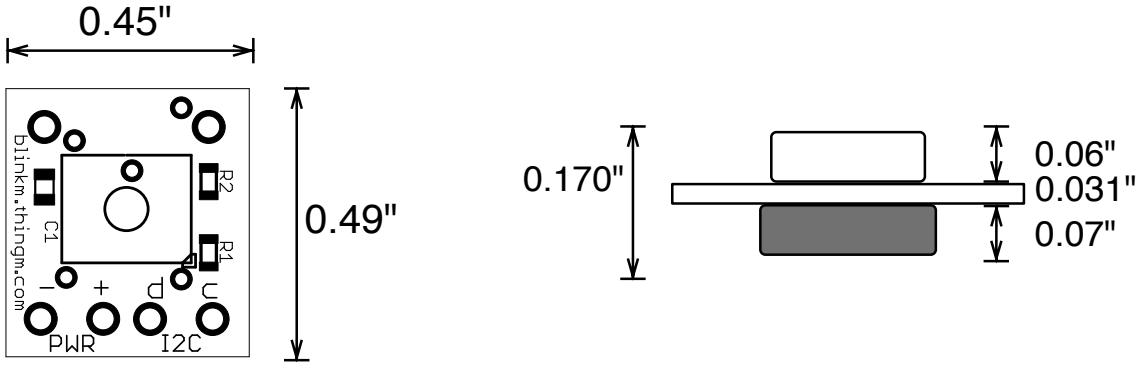


Figure 9.3: BlinkM MaxM "Master" Packaging Information

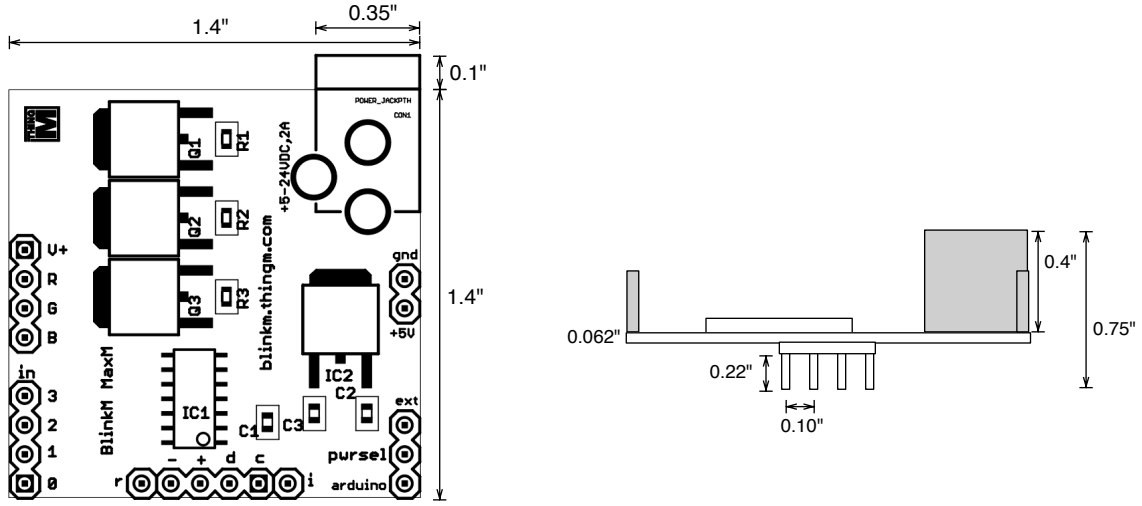


Figure 9.4: BlinkM MaxM "Blaster" Packaging Information

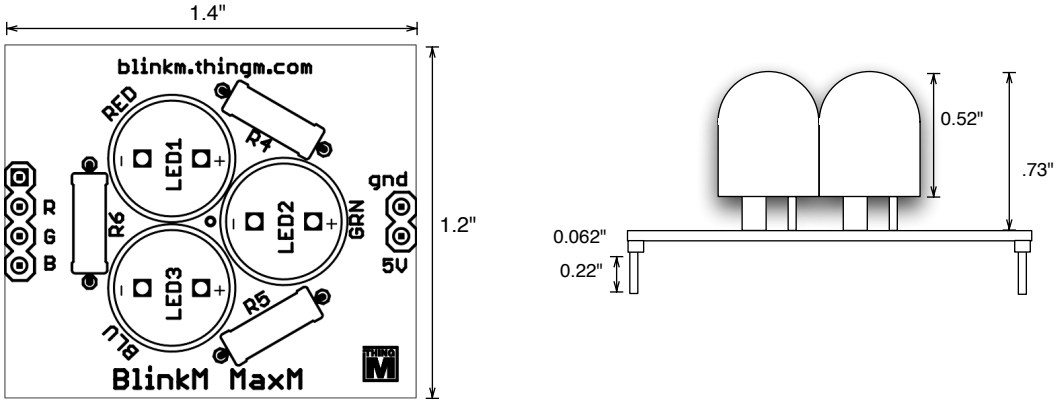
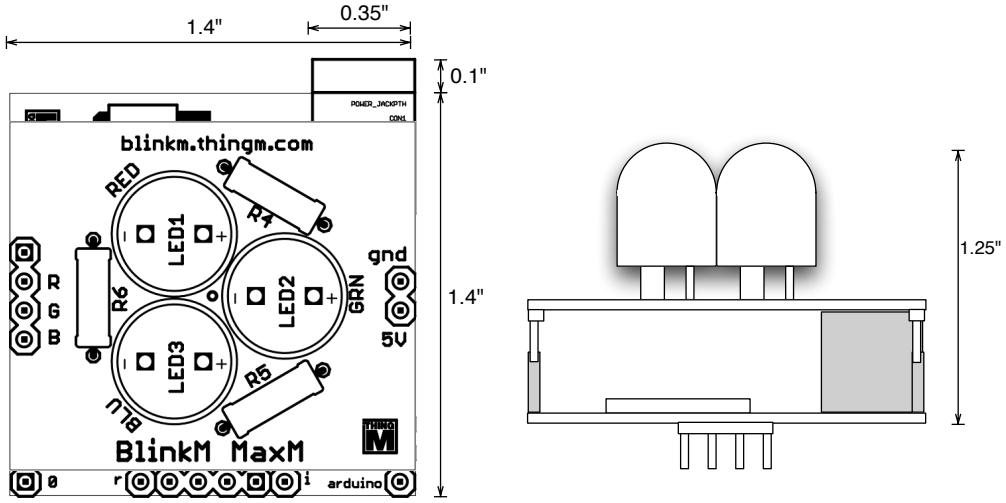


Figure 9.3: BlinkM MaxM "Master+Blaster" Packaging Information



10. Updates to this Datasheet

20070102 – initial release

20070111 – expanded Section 5.2 on multiple BlinkMs description & updated diagram

20070113 – changed SCK to SCL to better match I2C nomenclature

20070130 – added example about “formats. fixed error on “Set Startup Parameters” description, added note about {‘B’,0,...} command limitations

20081101 – added MaxM

20090111 – added clarification about I2C addressing

20090720 – fixed typo on “Get Address” command (‘Z’ vs ‘z’)

20100810 – added MinM, various reformat and improvements

20110511 – Fixed pwr/gnd connections in Figure 2.2.1.

20151027 – Reduced input voltage spec to 12V to prevent potential damage issues

THINGM LABS

http://thingm.com/

1126 Palm Terrace
Pasadena, CA 91104