blinkm.thingm.com

BLINKM DATASHEET

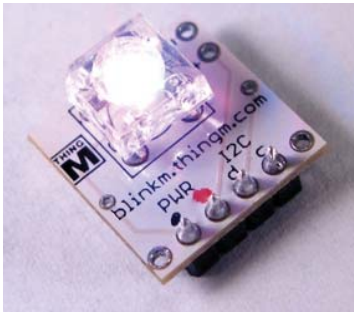
BlinkM
BlinkM MinM
BlinkM MaxM

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.

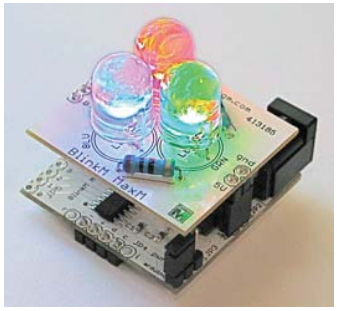
The BlinkM family of Smart LEDs makes it easy to add dynamic stand-alone lighting no matter what your project.

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 140º 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 5-24VDC @ 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

BLINKM DATASHEET
for BlinkM & BlinkM MaxM

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 24V 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

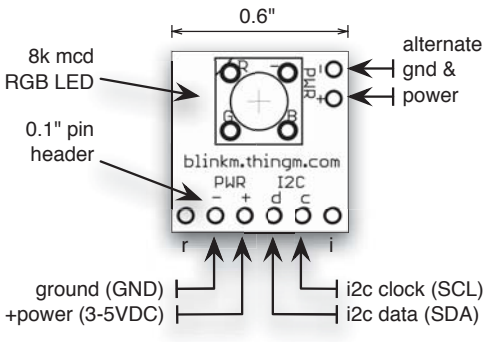


Photo 1.1: BlinkM

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

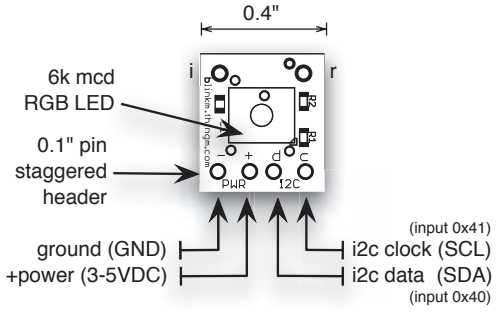
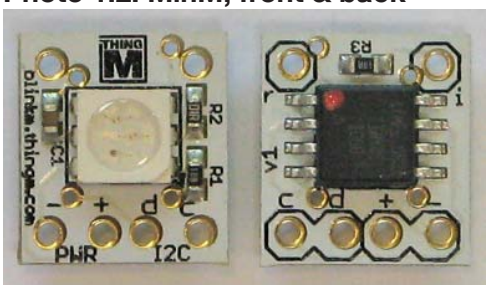


Photo 1.2: MinM, front & back



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').

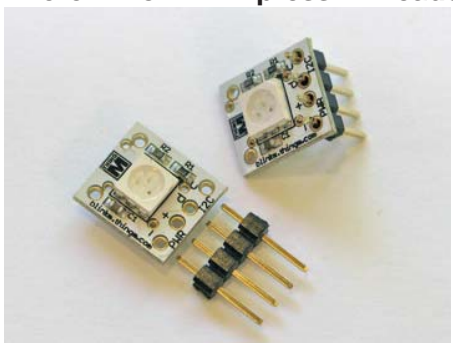
1.2.3. MinM Press-fit header holes

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"

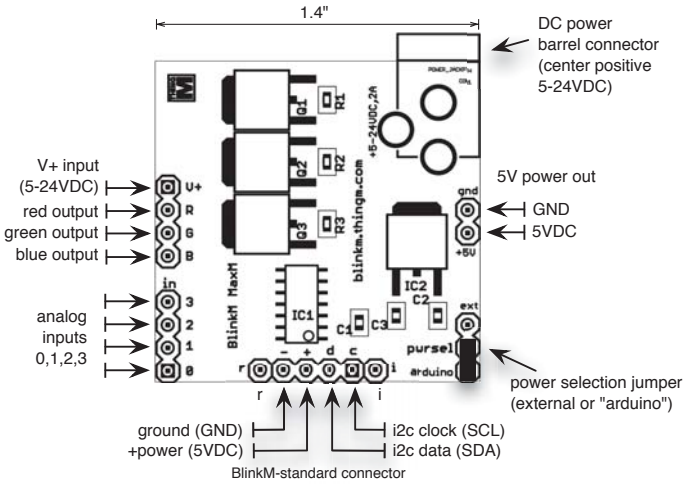


Photo 1.3.1: BlinkM MaxM "Master"

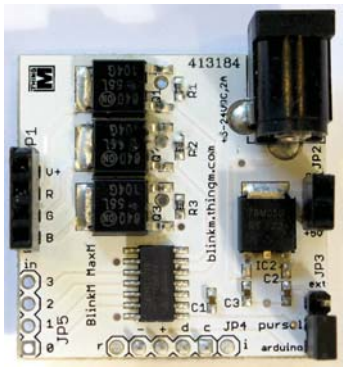


Figure 1.3.2: BlinkM MaxM "Blaster"

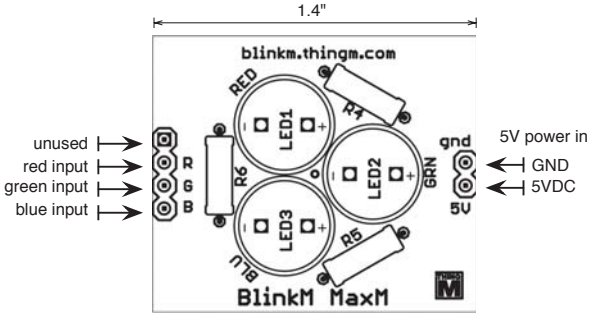


Photo 1.3.2: BlinkM MaxM "Blaster"

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 (5-24 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 +24 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

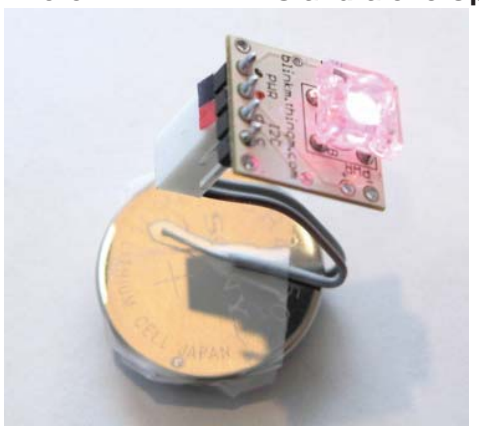
After a light script is programmed into a BlinkM or MaxM, it can be removed from its programmer and used without an external controller. All BlinkMs, MinMs, and MaxMs ship with a default startup light script so one can immediately see how it works.

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

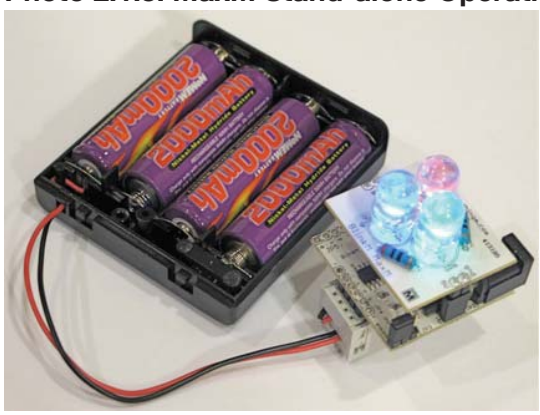


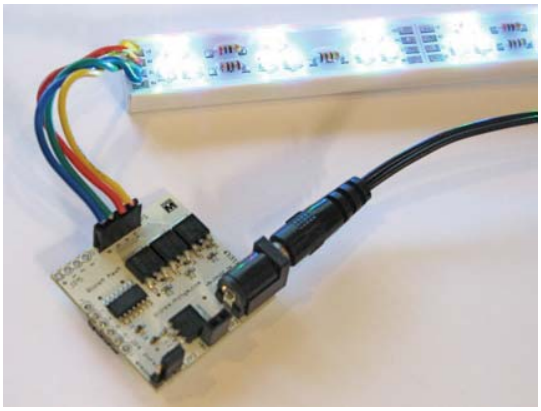
2.1.3 Stand-alone Operation for MaxM

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

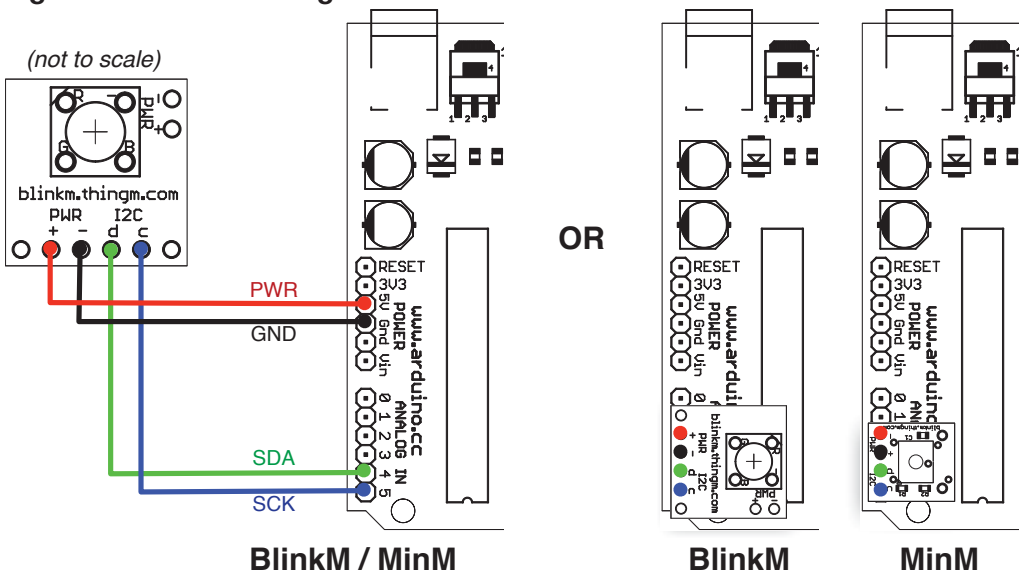
BlinkM needs two wires for power and two for data. Figure 2.2.1a shows the connections needed when hooked up to an Arduino. The Arduino “analog in” pins 4 & 5 also double as the I2C data signal (“SDA”) and clock signal (“SCL”), respectively.

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

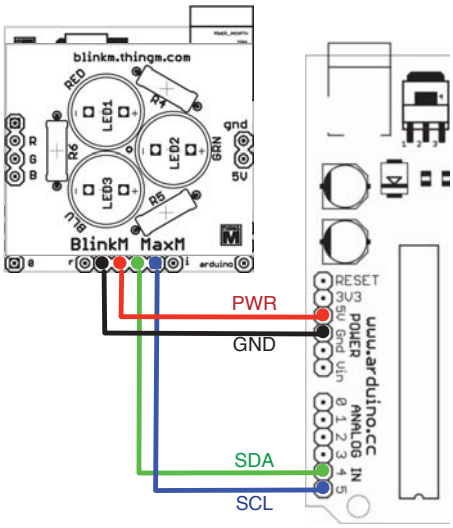


Photo 2.2.2: Connecting MaxM to Arduino

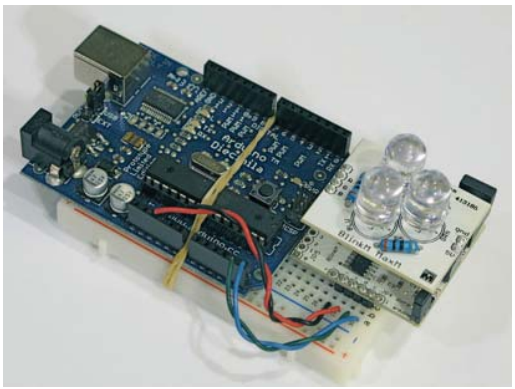
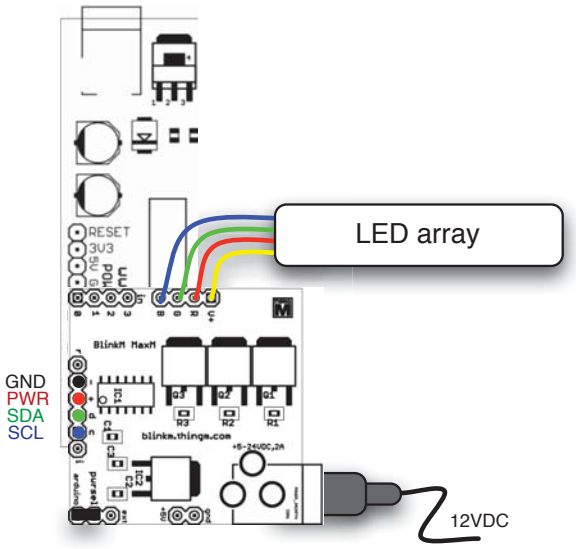


Photo 2.2.2 shows an example method of how to connect a MaxM to an Arduino. Note that unlike a BlinkM, a MaxM cannot be plugged directly into Arduino's Analog In pins 2,3,4,5 because the MaxM draws too much power.

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

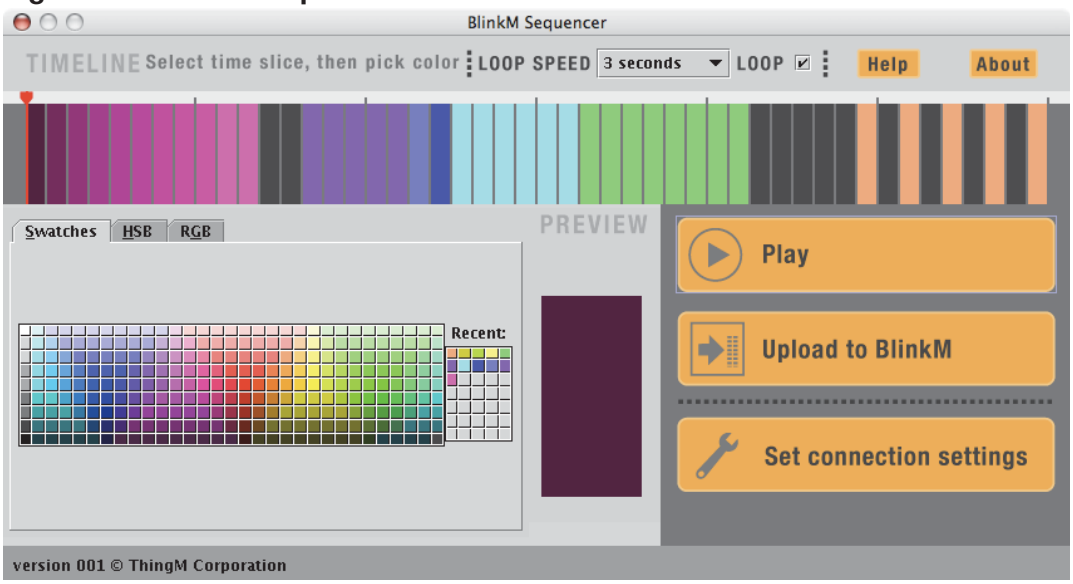
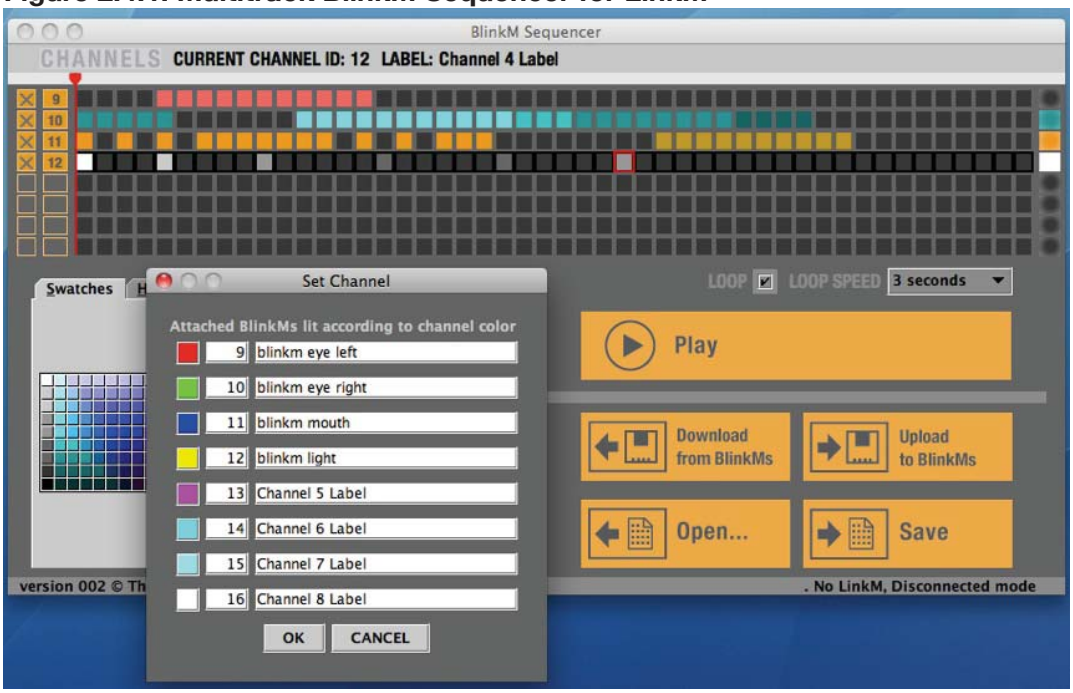


Figure 2.4.1: Multitrack BlinkM Sequencer for LinkM



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}

* 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.

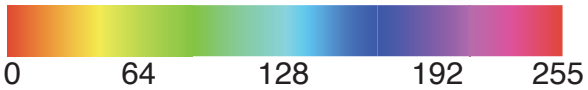
Examples

```
{'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