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)

MinM

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

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)

MinM

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

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.

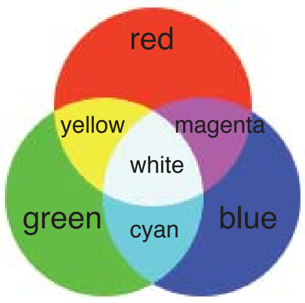
4.3 Color Models

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

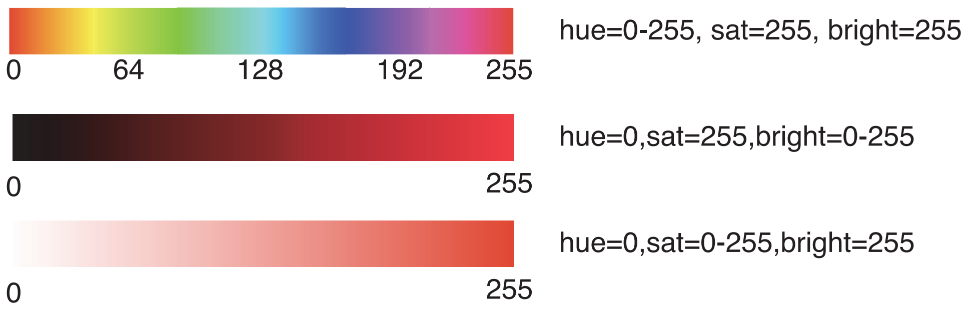


4.3.2 About the HSB Color Model

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


hue=0-255, sat=255, bright=255
0
64
128
192
255
hue=0,sat=255,bright=0-255
0
255
hue=0,sat=0-255,bright=255
0
255

This is equivalent to going around the edge of the color wheel but instead of ranging from 0° to 360°, 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

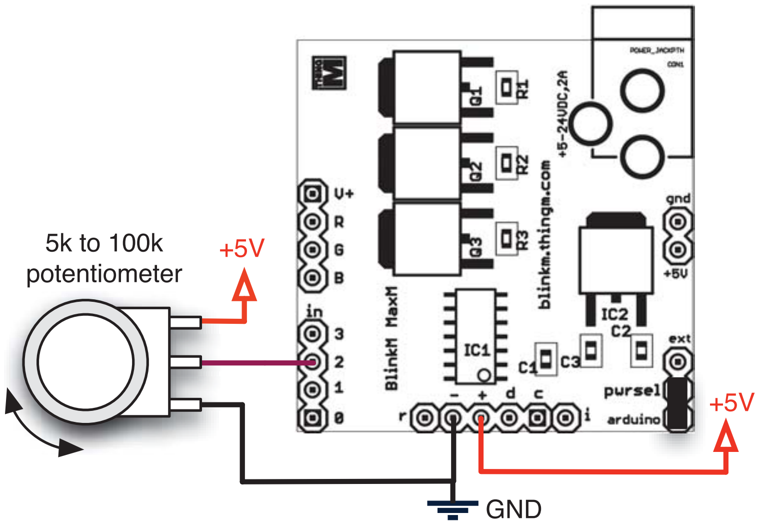
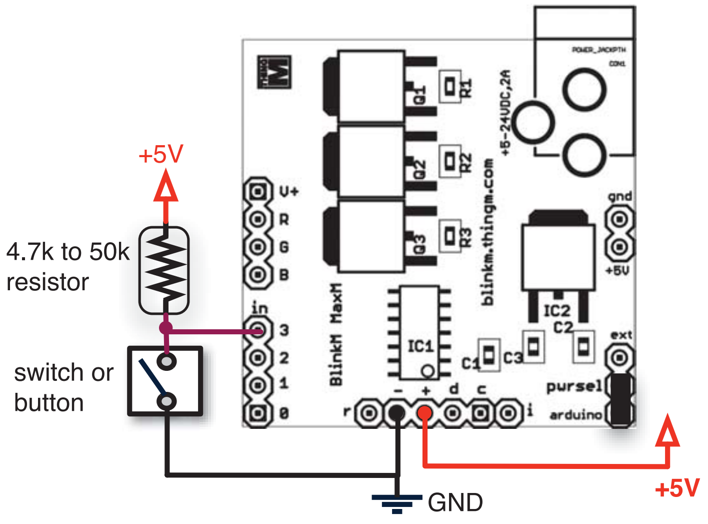


Figure 4.5.1 shows how to add a button to create a digital control. The resistor is used to set the default (non-pressed) value to be the highest possible voltage. Reading the input without pressing the button would give a value around 255 (0xff). The button is also wired so that when it is pushed, it 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

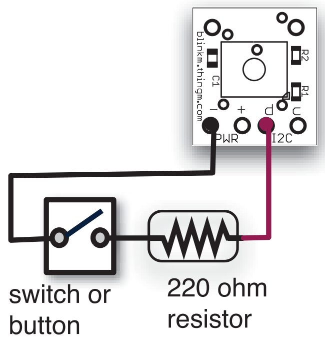


4.6 Reading Inputs with MinM

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.

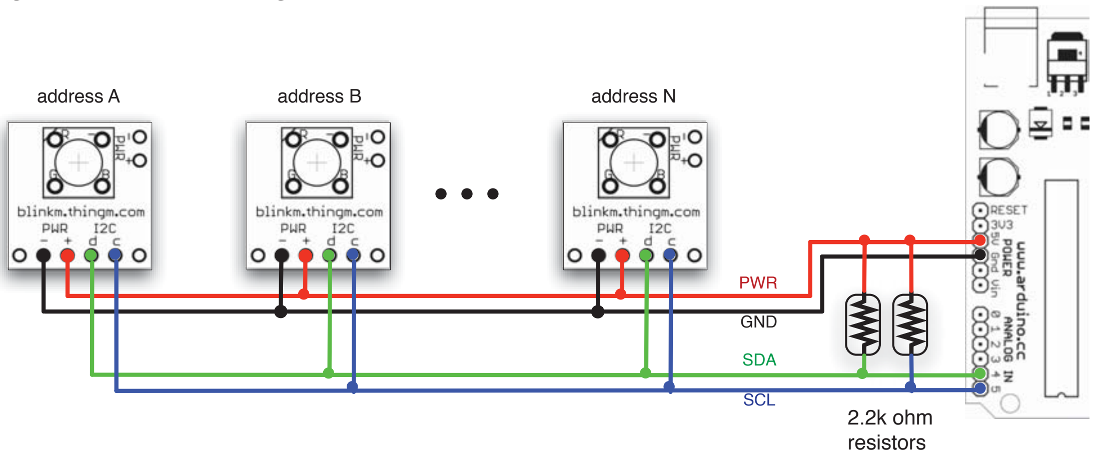
5.1 Connecting Multiple BlinkMs

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

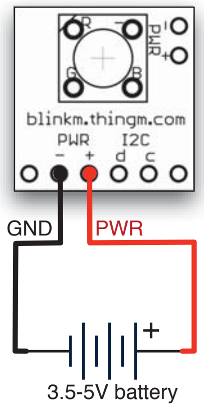
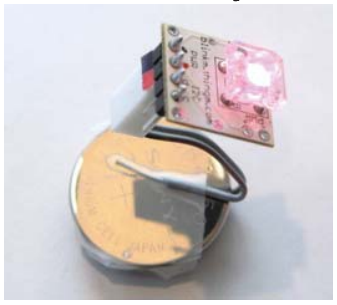


Photo 5.3: Battery Powered BlinkM



5.3.1 Battery Powered MaxM

MaxM

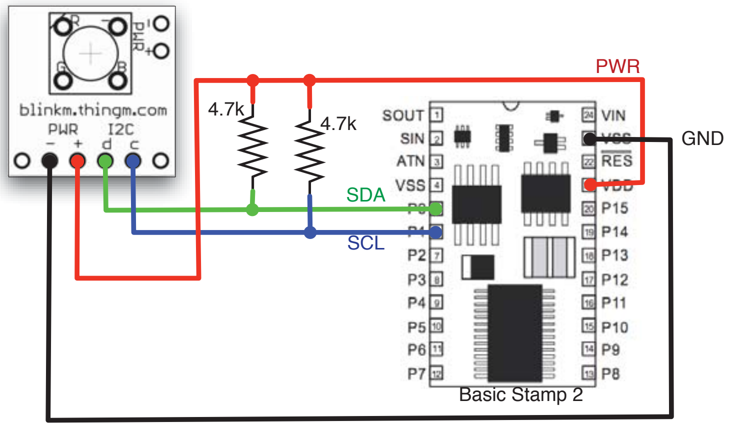
When using a MaxM in Master+Blaster configuration, it is so bright and draws so much power that only somewhat large battery packs should be used. A good portable configuration is a 4xAA battery pack which should give several hours of operation.

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.

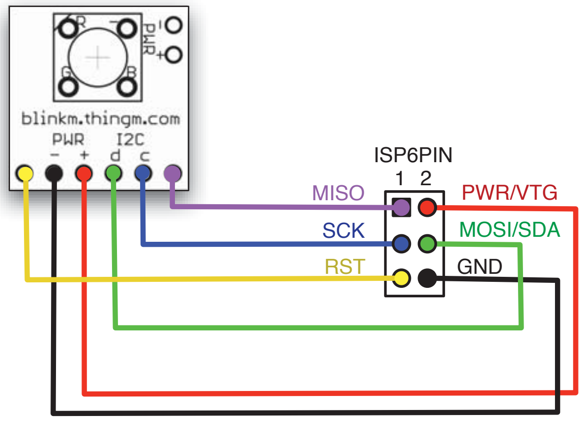
Figure 5.3: Connecting BlinkM to a Basic Stamp 2



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		24	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

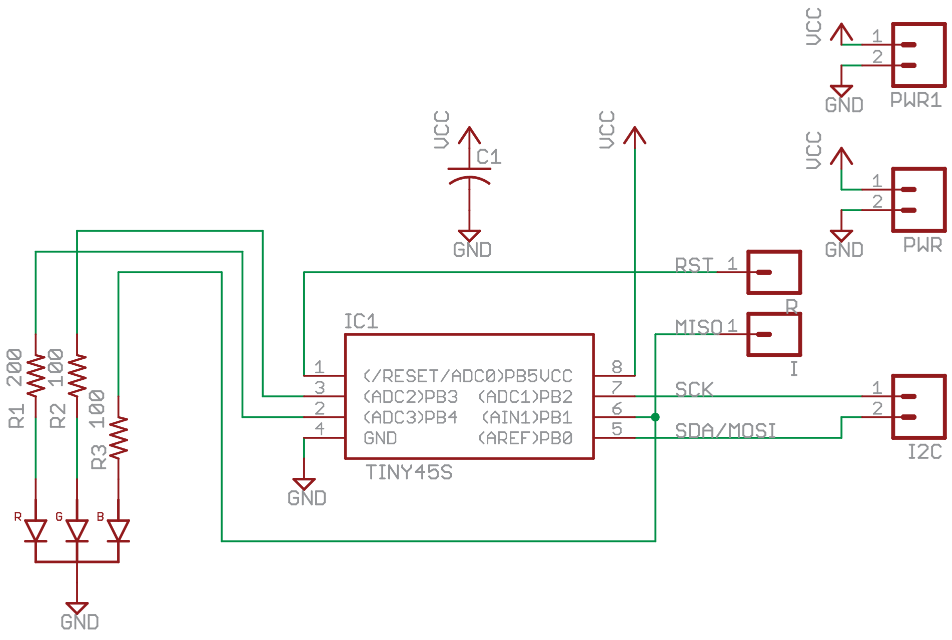


Figure 8.2: BlinkM MaxM “Master” driver board

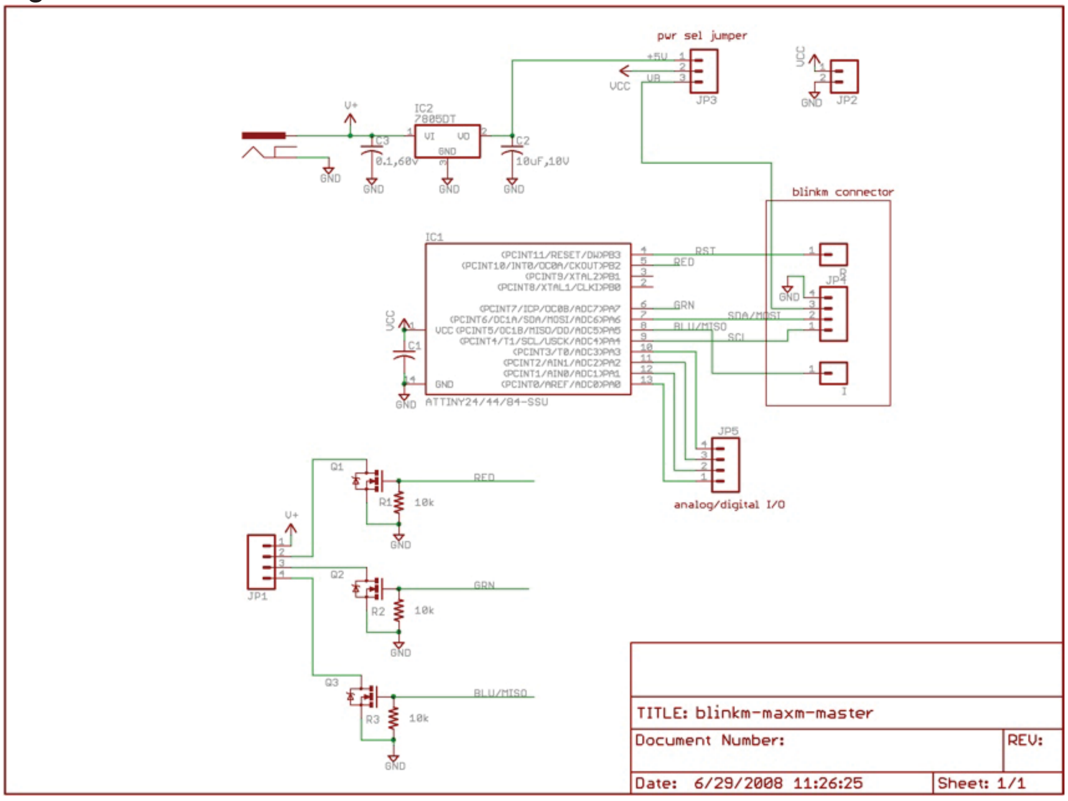
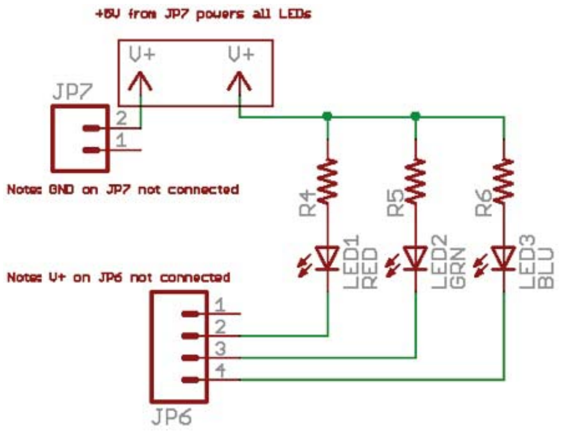


Figure 8.3: BlinkM MaxM “Blaster” LED board


+5V from JP7 powers all LEDs
V+
V+
JP7
Note: GND on JP7 not connected
R4
R5
R6
LED1 RED
LED2 GRN
LED3 BLU
Note: V+ on JP6 not connected
JP6

9. Packaging Information

All units in inches.

Figure 9.1: BlinkM Packaging Information

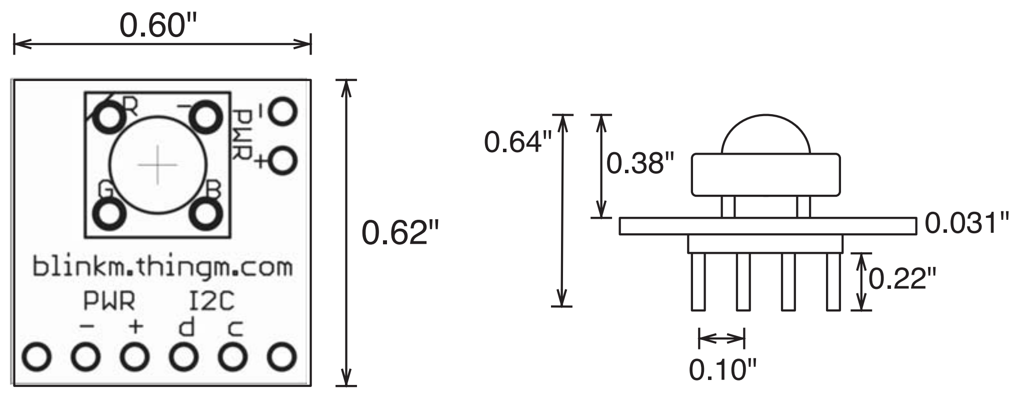


Figure 9.2: MinM Packaging Information

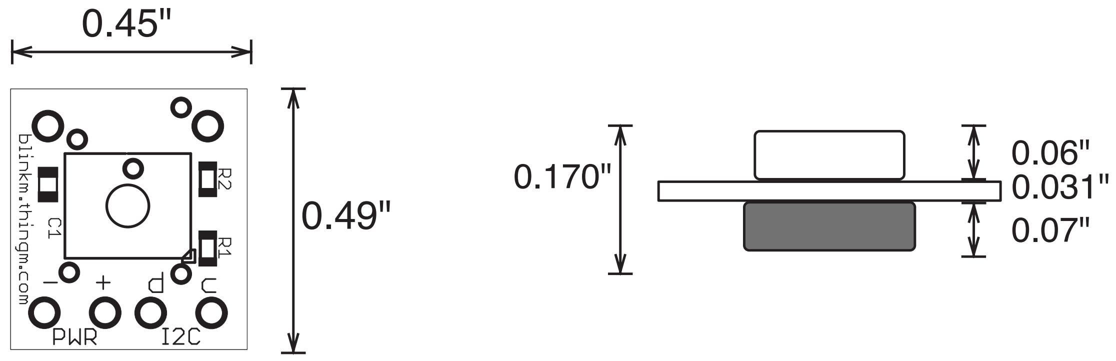


Figure 9.3: BlinkM MaxM "Master" Packaging Information

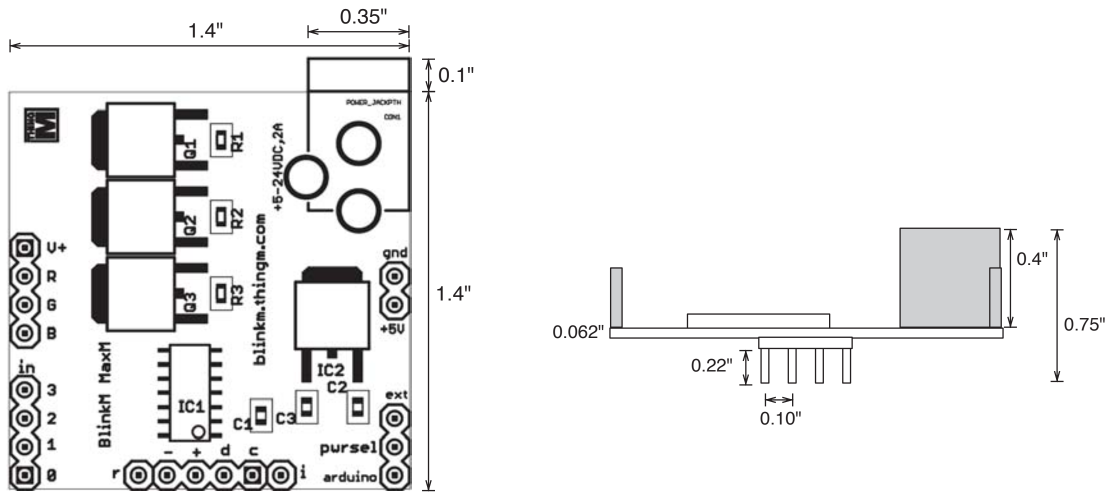


Figure 9.4: BlinkM MaxM "Blaster" Packaging Information

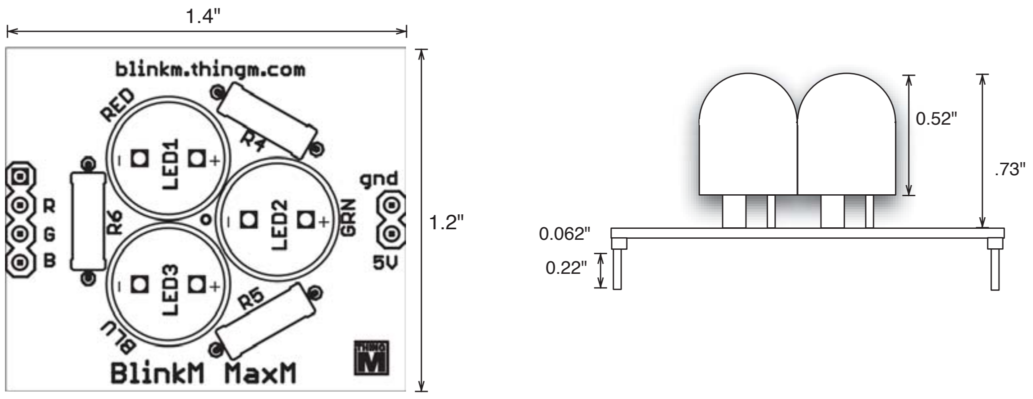
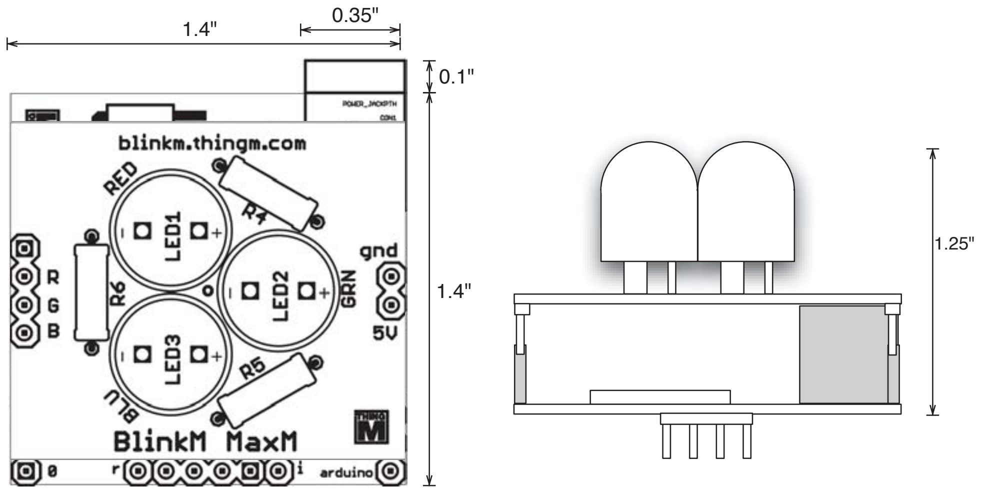


Figure 9.3: BlinkM MaxM "Master+Blaster" Packaging Information



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

THINGM LABS

http://thingm.com/

1126 Palm Terrace
Pasadena, CA 91104