



Beginning Maximite Programming and Interfacing

with the

CircuitGizmos CGMMSTICK and CGCOLORMAX

Applies to:

CGMMSTICK1, CGCOLORMAX1, CGCOLORMAX2, MMBasic 4.3+

Acknowledgments:

Beginning Maximite Programming and Interfacing with the CircuitGizmos
CGMMSTICK and CGCOLORMAX Copyright 2013 CircuitGizmos, LLC
<http://circuitgizmos.com> with language reference parts provided with permission by
Geoff Graham <http://geoffg.net>

License: MMBasic is Copyright 2011, 2012, 2013 Geoff Graham -
<http://mmbasic.com>.

Thanks to John Harding for the CAN documentation and MMBasic CAN support.

Thank you to Leslie for manuscript feedback and to Harry Strand
<http://www.homedomination.com/> for technical review.

A **huge** thank you to Geoff Graham <http://geoffg.net> for his phenomenal efforts.

Thanks to the good people on The Back Shed (http://www.thebackshed.com/forum/forum_topics.asp?FID=16&PN=0) for their feedback on this document and MMBasic support.

Updated versions of this document can be found at:
<http://www.circuitgizmos.com/files/begmax.pdf>

This book contains references to copyrighted products that are not explicitly indicated as such. The absence of the copyright symbol does not infer that a product is not protected. Registered trademarks are similarly not expressly indicated.

Dedication:

To Josette and Sterling. Thank you for being patient as I wrote this, and, well, patience for all sorts of other things.

In loving memory of my Mom. You touched my life indelibly.

Contents

Contents.....	4
Preface.....	11
Quick CGMMSTICK Hardware Interface Example.....	17
Maximite / CGMMSTICK / CGCOLORMAX Introduction.....	26
CGMMSTICK / CGCOLORMAX Setup and MMIDE.....	35
CGMMSTICK Setup.....	36
CGColorMAX Setup.....	43
MMIDE Setup.....	47
Maximite BASIC Features.....	59
Project #1	
CGMMSTICK LED Control Example.....	61
Project #2	
CGColorMAX Relay Shield Example.....	68
Project #3	
CGMMSTICK Sound Example.....	74
Note/Frequency Table for SOUND command.....	78
Project #4	
CGMMSTICK Input Example and Analog Input Example.....	81
Project #5	
CGMMSTICK LCD Interface Example.....	85
Project #6	
CGMMSTICK Vacuum Fluorescent Display Example.....	87
Project #7	
CGColorMAX1 Serial I/O Example.....	92
Serial Communications.....	98
The Serial OPEN Command.....	99
Serial Input/Output Pin Allocation.....	101
Reading and Writing the Serial Port.....	102
Opening a Serial Port as the Console.....	103
Project #8	
CGMMSTICK SPI Interface Example.....	104
Serial Peripheral Interface (SPI).....	108
Project #9	
CGMMSTICK I2C Interface Example.....	111
Inter Integrated Circuit (I2C) bus.....	115

Project #10	
CGMMSTICK 1-Wire Interface Example.....	117
1-Wire Interface.....	122
Project #11	
CGCOLORMAX LCD Shield Example.....	124
Project #12	
CGKEYCHIP1 Custom Keypad Example.....	137
CGMMSTICK1 Technical Information.....	144
SD card insertion (CGMMSTICK1).....	146
SD card removal (CGMMSTICK1).....	149
J1 Pinout (CGMMSTICK1).....	151
CGVGAKBD1 Schematic.....	152
Schematic (CGMMSTICK1).....	155
Firmware Update (CGMMSTICK1).....	156
Firmware Upgrade Steps (CGMMSTICK1).....	157
CGCOLORMAX1 Technical Information.....	162
Power Supply (CGCOLORMAX1).....	163
Rear Connector (CGCOLORMAX1).....	165
Shield Connections (CGCOLORMAX1).....	168
Interface Circuits (CGCOLORMAX1).....	170
Misc Connections (CGCOLORMAX1).....	172
Pin Outs and Function (CGCOLORMAX1).....	173
Firmware Update (CGCOLORMAX1).....	174
Firmware Upgrade Steps.....	175
CGCOLORMAX2 Technical Information.....	180
Power Supply (CGCOLORMAX2).....	181
Rear Connectors (CGCOLORMAX2).....	184
Shield Connections (CGCOLORMAX2).....	188
Interface Circuits (CGCOLORMAX2).....	190
Misc Connections (CGCOLORMAX2).....	194
Pin Outs and Function (CGCOLORMAX2).....	197
Firmware Update (CGCOLORMAX2).....	198
Firmware Upgrade Steps.....	199
CGMMSTICK and CGCOLORMAX I/O Characteristics.....	204
CGColorMAX Composite Output.....	205
Joystick Connections.....	206
Example Shields.....	208
MMBasic Reference.....	210
Language Constructs.....	211
Editing, Programming, and Debugging.....	213
Screen, Graphics, and Keyboard.....	214
Math Functions.....	216

<u>Character and String Functions</u>	217
<u>File System</u>	218
<u>Input and Output</u>	219
<u>Language Implementation Characteristics</u>	221
MMBasic language constructs	222
<u>Defined Subroutines and Functions</u>	223
<u>Subroutine Arguments</u>	224
<u>Local Variables</u>	225
<u>Defined Functions</u>	227
<u>Passing Arguments by Reference</u>	229
<u>Naming Conventions</u>	232
<u>Timing</u>	233
<u>Interrupts</u>	234
CLEAR	235
CONTINUE	236
DATA	237
DIM	238
DO LOOP	239
DO LOOP UNTIL	240
DO WHILE LOOP	241
ELSE	242
ELSEIF THEN	244
END	246
END FUNCTION	247
END SUB	248
ENDIF	249
ERASE	251
EXIT	252
EXIT FOR	253
EXIT FUNCTION	254
EXIT SUB	255
FOR TO STEP	256
FUNCTION	257
GOSUB	259
GOTO	260
IF THEN ELSE	261
IF THEN GOTO	263
IF THEN ELSE ELSEIF ENDIF	265
IRETURN	266
LET	267
LOCAL	268
LOOP UNTIL	269
NEXT	270
ON nbr GOTO GOSUB	272
OPTION BASE	273
OPTION Fnn	274
OPTION PROMPT	275
OPTION USB	276
PAUSE	277
RANDOMIZE	278
READ	279
REM or '	280
RESTORE	281
RETURN	282
SETTICK	283
SUB	284
TIMER (command/function)	285

<u>WHILE WEND</u>	286
<u>MMBasic editing, programming, and debugging</u>	287
Full Screen Editor	289
<u>AUTO</u>	291
<u>CHAIN</u>	292
<u>CONFIG CASE</u>	293
<u>CONFIG KEYBOARD</u>	294
<u>CONFIG TAB</u>	295
<u>COPYRIGHT</u>	296
<u>DATE\$ (command/function)</u>	297
<u>DELETE</u>	298
<u>EDIT</u>	299
<u>ERROR</u>	301
<u>LIST</u>	302
<u>LOAD</u>	303
<u>MEMORY</u>	304
<u>MERGE</u>	305
<u>MM.CMDLINES</u>	306
<u>MM.DEVICES</u>	307
<u>MM.FNAME\$</u>	308
<u>MM.VER</u>	309
<u>NEW</u>	310
<u>OPTION BREAK</u>	311
<u>RENUMBER</u>	312
<u>RUN</u>	313
<u>SAVE</u>	315
<u>TIME\$ (command/function)</u>	316
<u>TROFF</u>	317
<u>TRON</u>	318
<u>MMBasic Screen, Graphics, and Keyboard</u>	319
Graphics	320
Working with Color	328
Color Modes	330
Scan Line Color Override	332
Game Playing Features	333
Loadable Fonts	337
Sprite Definition Files	339
Special Keyboard Keys	341
<u>BLIT</u>	343
<u>CIRCLE</u>	347
<u>CLR\$</u>	352
<u>CLS</u>	354
<u>COLLISION</u>	355
<u>COLOR</u>	356
<u>CONFIG COMPOSITE</u>	359
<u>CONFIG VIDEO</u>	361
<u>FONT</u>	362
<u>FONT LOAD/UNLOAD</u>	365
<u>INKEY\$</u>	369
<u>INPUT (from keyboard)</u>	370
<u>KEYDOWN (keyboard)</u>	371
<u>LINE</u>	372
<u>LINE INPUT (from keyboard)</u>	379
<u>LOADBMP</u>	380
<u>LOCATE</u>	381
<u>MM.HPOS</u>	384

<u>MM.HRES</u>	385
<u>MM.VPOS</u>	387
<u>MM.VRES</u>	389
<u>MODE</u>	391
<u>OPTION VIDEO</u>	395
<u>PIXEL (command/function)</u>	396
<u>POS</u>	402
<u>PRESET</u>	403
<u>PRINT</u>	405
<u>PRINT @</u>	406
<u>PSET</u>	409
<u>SAVEBMP</u>	411
<u>SCANLINE</u>	412
<u>SPRITE</u>	414
<u>TAB</u>	417
<u>MMBasic math functions</u>	419
<u>ABS</u>	420
<u>ATN</u>	421
<u>CINT</u>	422
<u>COS</u>	423
<u>DEG</u>	424
<u>EXP</u>	425
<u>FIX</u>	426
<u>INT</u>	427
<u>LOG</u>	428
<u>PI</u>	429
<u>RAD</u>	430
<u>RND</u>	431
<u>SGN</u>	432
<u>SIN</u>	433
<u>SQR</u>	434
<u>TAN</u>	435
<u>MMBasic character and string functions</u>	436
<u>ASC</u>	437
<u>BINS</u>	438
<u>CHRS</u>	439
<u>FORMAT\$</u>	440
<u>HEX\$</u>	442
<u>INSTR</u>	443
<u>LCASE\$</u>	444
<u>LEFT\$</u>	445
<u>LEN</u>	446
<u>MIDS</u>	447
<u>OCTS</u>	448
<u>RIGHT\$</u>	449
<u>SPACE\$</u>	450
<u>SPC</u>	451
<u>STR\$</u>	452
<u>STRING\$</u>	453
<u>UCASE\$</u>	454
<u>VAL</u>	455
<u>MMBasic File System</u>	456
<u>Storage Commands and Functions</u>	457
<u>CHDIR</u>	458
<u>CLOSE (file)</u>	459
<u>COPY</u>	460

<u>CWD\$</u>	461
<u>DIR\$</u>	462
<u>DRIVE</u>	463
<u>EOF</u>	464
<u>FILES</u>	465
<u>INPUT #</u>	466
<u>INPUT\$ (from file)</u>	467
<u>KILL</u>	468
<u>LINE INPUT (file)</u>	469
<u>MKDIR</u>	470
<u>MM.DRIVES\$</u>	471
<u>MM.ERRNO</u>	472
<u>NAME</u>	473
<u>OPEN (file)</u>	474
<u>OPTION ERROR</u>	476
<u>PRINT</u>	477
<u>RMDIR</u>	478
<u>WRITE</u>	479
<u>XMODEM</u>	480
<u>MMBasic input and output</u>	481
Communications	482
Interrupts	483
CAN (Controller Area Network)	484
Audio and PWM Output	494
<u>BYTE2NUM</u>	496
<u>CAN</u>	497
<u>CLOSE (serial)</u>	500
<u>CLOSE CONSOLE (serial)</u>	501
<u>I2CDIS</u>	502
<u>I2CEN</u>	503
<u>I2CRCV</u>	504
<u>I2CSEND</u>	506
<u>I2CSDIS</u>	507
<u>I2CSEN</u>	508
<u>I2CSRCV</u>	509
<u>I2CSSEND</u>	510
<u>INPUT\$ (from serial)</u>	511
<u>LOC</u>	512
<u>LOF</u>	513
<u>MM.I2C</u>	514
<u>MM.OW</u>	515
<u>NUM2BYTE</u>	516
<u>OPEN (serial)</u>	517
<u>OPEN AS CONSOLE (serial)</u>	518
<u>OWCRC8</u>	519
<u>OWCRC16</u>	520
<u>OWREAD</u>	521
<u>OWRESET</u>	523
<u>OWSEARCH</u>	524
<u>OWWRITE</u>	525
<u>PEEK</u>	526
<u>PIN (command/function)</u>	527
<u>PLAYMOD</u>	528
<u>PLAYMOD STOP</u>	529
<u>POKE</u>	530
<u>PORT (command/function)</u>	531
<u>PULSE</u>	532

<u>PWM</u>	533
<u>PWM STOP</u>	534
<u>SETPIN</u>	535
<u>SOUND</u>	536
<u>SPI</u>	537
<u>TONE</u>	538
<u>TONE STOP</u>	539
<u>FAQ – Frequently Asked Questions</u>	540
<u>ASCII Tables</u>	542
<u>Index</u>	546

Preface

Hi, and welcome to Beginning Maximite Programming and Interfacing with CircuitGizmos CGMMSTICK and CGCOLORMAX!

If you are new to Maximites, or specifically new to the CGMMSTICK / CGCOLORMAX made by CircuitGizmos (<http://www.circuitgizmos.com>), then this document is for you.

So what is the Maximite? It is a small retro-computer that really DOES FUN STUFF!

If your first computer is a multi-core, multi-gigabyte PC like the new ones sold today, or if your first computer was a "Home Computer" back in the 1980s, you might be aching to have your computer do something "real" around you.

The desktop PC that I'm typing this document on certainly does a lot - word processing, surfing the internet, and video editing. In fact compared to the Sinclair ZX81, TRS-80, Tandy COCO (Color Computer), Apple II, and Commodore 64 of the 1980s a modern computer is really a phenomenal machine.

Ever since I had Sinclair, TI 99/4A, TRS-80, Tandy COCO, and Apple II computer back in the 1980s I have wanted to use the machines to interface to the real world and write a BASIC program to control my hardware. That could be done if you found some unique interface hardware to do the job. But even if you did that, you were converting the only personal computer that you had at that time into a very expensive controller.

Wouldn't it have been nice to have several multi-thousand dollar computer setups to play with? Several Sinclair ZX81 or Tandy COCO computers set up that you could conveniently wire your circuits to? I couldn't afford to do that in the 1980s. Not on my budget.

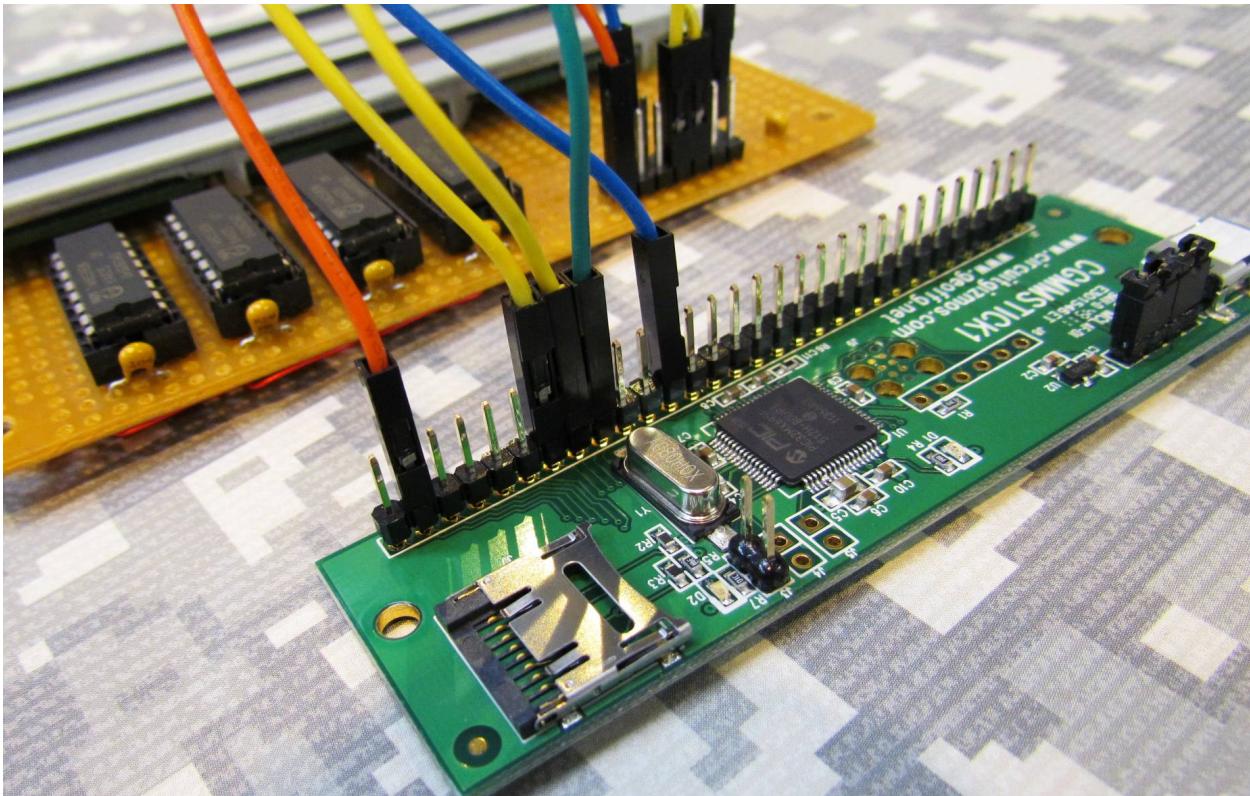


Figure 1: CGMMSTICK1 controlling LCD displays.

That is exactly what the Maximite can do for you. The Maximite has more power than an Apple II or Sinclair ZX81 on a single small circuit board WITH the ability to interface to your own hardware easily. The original Maximite was the clever invention of Geoff Graham (<http://geoffg.net>) and on a small printed circuit board he included power that rivals the 1980s Home Computer.

The Maximite that Geoff designed runs BASIC like the 1980s Home Computers did, but has a keyboard interface, a VGA connection, and an SD card. The SD card and internal (FLASH) drives take the place of the “floppy drives” of the 1980s – they are significantly better. The Maximite runs the BASIC code from these drives and can also store data on them. The Maximite also has a USB port that connects to your PC and (because of the PC driver) appears on the PC as a serial port.

There are 20 input/output lines on the CGMMSTICK Maximite that let you control LEDs, LCDs, motors, relays, and other output devices. There are even more I/O lines (an additional 20) on the CGCOLORMAX1. You can measure voltages, connect to switches and buttons, and interface to sensors. These lines let you connect to serial devices, I2C chips, and SPI circuits. Don't know how to do that? That is exactly what this document will help you learn.

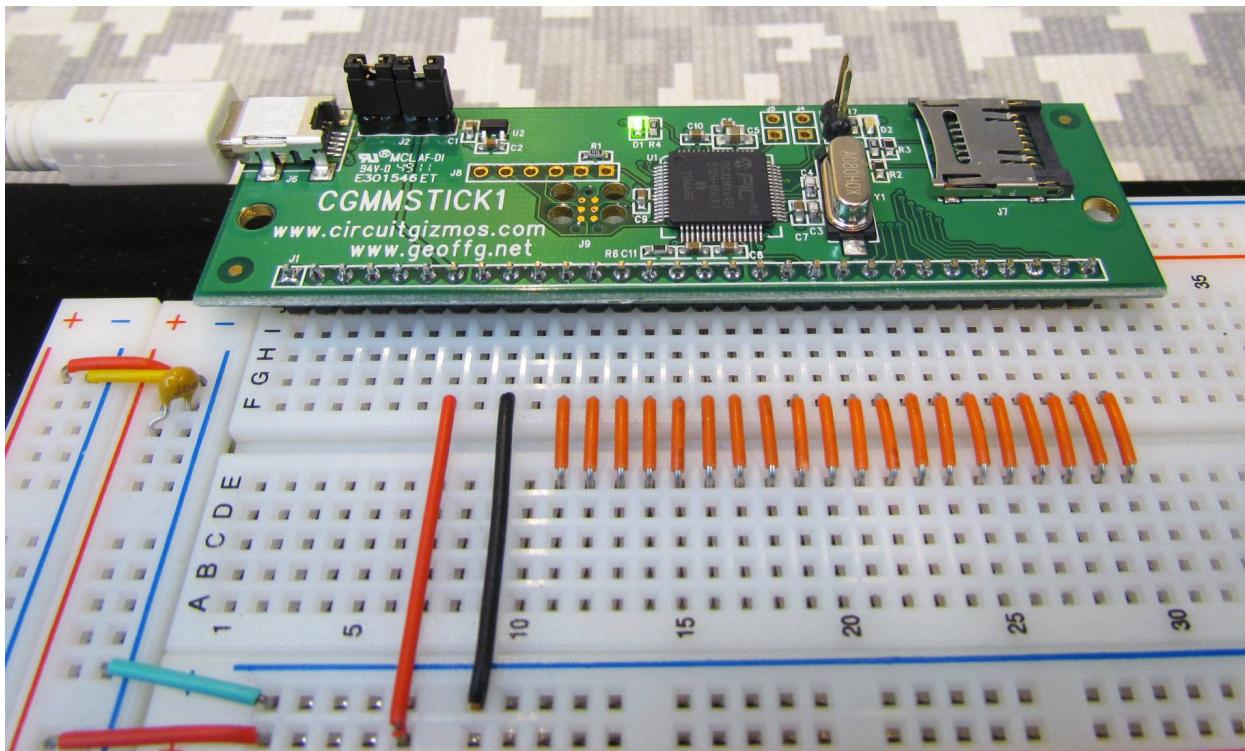


Figure 2: CGMMSTICK1 in a solderless breadboard.

The Maximites covered in this document are the products made by CircuitGizmos, LLC <http://circuitgizmos.com> the CGMMSTICK and the CGCOLORMAX.

As of this writing there are two versions of the CGCOLORMAX. The CGCOLORMAX1 was the first version and is being replaced by the CGCOLORMAX2. In this book they are referred to as CGCOLORMAX when differentiating them isn't necessary. When the differences need to be pointed out, then the revision number is used: CGCOLORMAX1 or CGCOLORMAX2.

The unique feature of the CGMMSTICK is that it can plug directly into a solderless breadboard for easy circuit prototyping. The CGMMSTICK is also easy to use in projects that are more permanent than solderless breadboard projects.

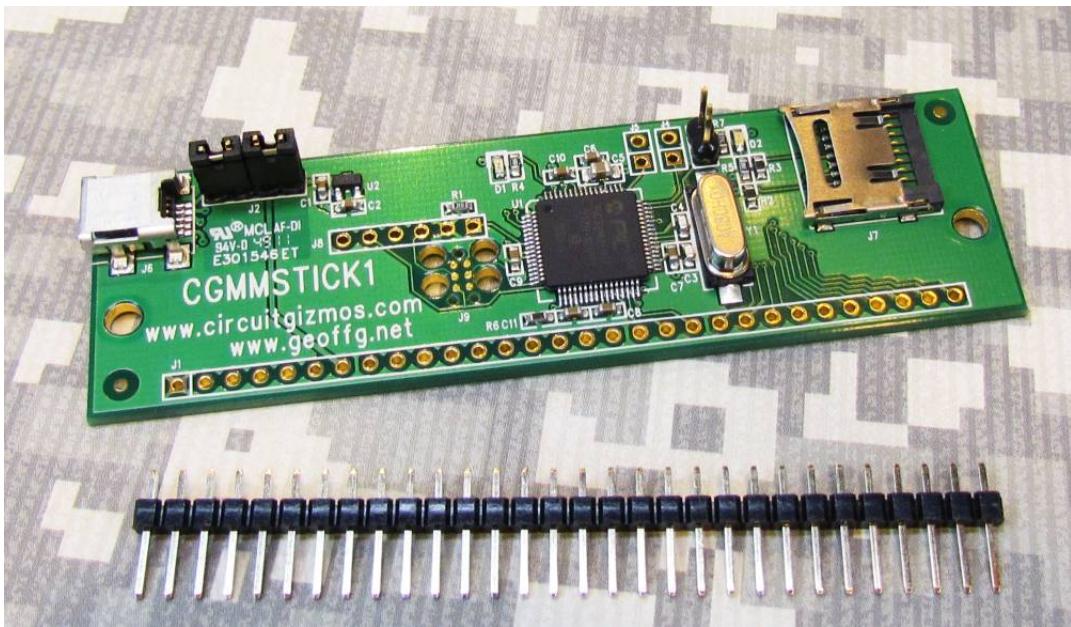


Figure 3: CGMMSTICK1 and a 30 pin header. A straight header or a right-angle header lets you use the 'stick' in a solderless breadboard.

On a printed circuit board that is only 3.5 inches long and 1 inch wide, the CGMMSTICK gives you **much** more than the power of a Sinclair ZX81 to run your hobby and engineering projects.



Figure 4: The CGCOLORMAX1 nicknamed "ColorMax". Female headers for Arduino shields and male headers have been added for experimentation.

The CGCOLORMAX has 8 color (black, red, green, blue, cyan, yellow, purple and white) graphic output and 20 more I/O lines. There is a footprint on the PCB that can support adding Arduino Shields to the CGCOLORMAX.

The CGCOLORMAX also has a “sea-of-holes” area for hardware prototyping and circuits for RS485 and RS232 interfacing. There is a populated battery-backed clock on board.

The CGCOLORMAX can be powered from 5V on USB or 8 to 18V on the input power connector.

The CGCOLORMAX comes on a 5.7 inch by 4 inch board and supports color graphics output on a VGA connector. Included is a keyboard connector and full-size SD card.

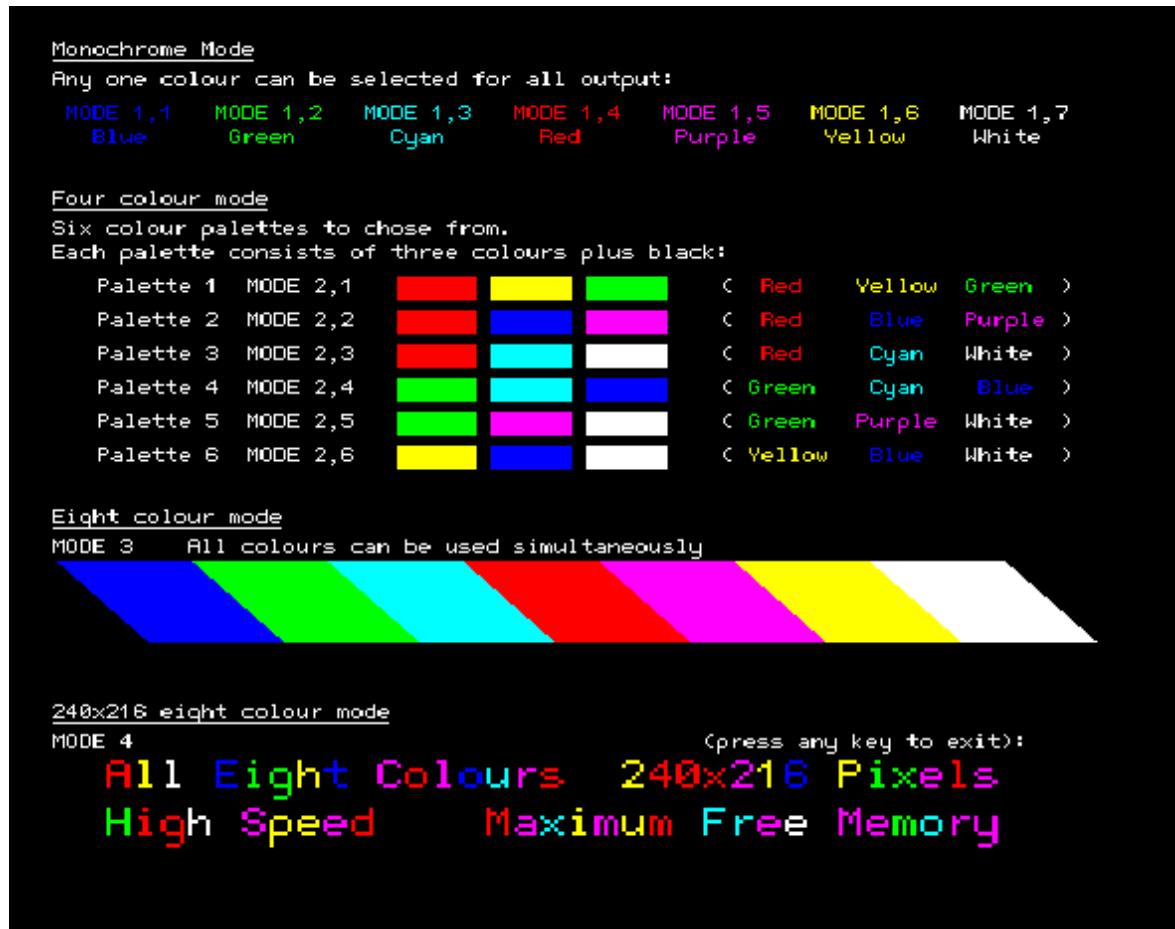


Figure 5: Example CGCOLORMAX VGA output.

MMBasic supports several video modes that allow you to balance color features against available program memory. You can even use the CGCOLORMAX in black and white mode to get the most program memory possible.

Quick CGMMSTICK Hardware Interface Example

If you are like me you really *don't want to wait* a few chapters to learn about what the Maximite CGMMSTICK and CGCOLORMAX can do. This chapter gives you a quick look at a CGMMSTICK example.

Using the CGMMSTICK or CGCOLORMAX might be very familiar or easy to understand for you, or you might be very new to all of this. Don't worry about the first examples here if you are new to all of this. This document does assume that you have some hobbyist-level familiarity with electronics and also familiarity with the BASIC language.

By the end of this document, you should be familiar enough with the workings of the CGMMSTICK/CGCOLORMAX to feel confident that you will be able to program and utilize the CGMMSTICK or CGCOLORMAX in your projects.

Before getting into greater detail on the Maximite hardware, software, and the workings of the CGMMSTICK or CGCOLORMAX, I'd like to whet your appetite for the Maximite with a simple CGMMSTICK example.

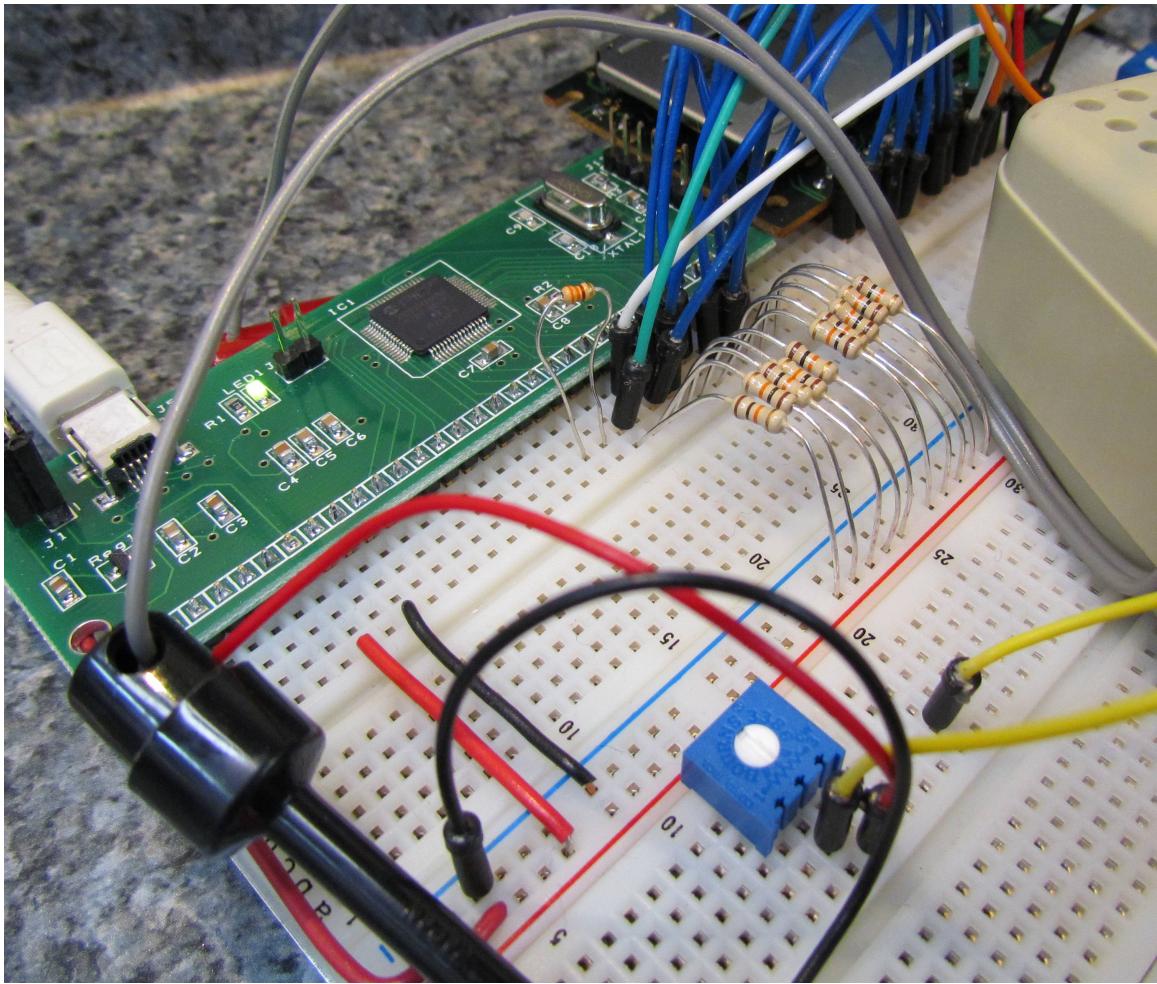


Figure 6: Miscellaneous interfacing using the prototype for the CGMMSTICK1. The CGMMSTICK development prototype pictured here was used for a lot of the initial CGMMSTICK1 testing.

It is easy to plug the CGMMSTICK into a white solderless breadboard and connect a USB cable to a PC to control the Maximite. This example demonstrates connecting an LED to one of the interface lines of the CGMMSTICK.

This introductory example uses a program that runs on PC/Linux/Mac as an aid to CGMMSTICK project development. The MMIDE program uses the serial port that is created when you connect the CGMMSTICK to your computer. The MMIDE program can be used as a terminal, as a way to transfer files, and to assist in hardware/software development.

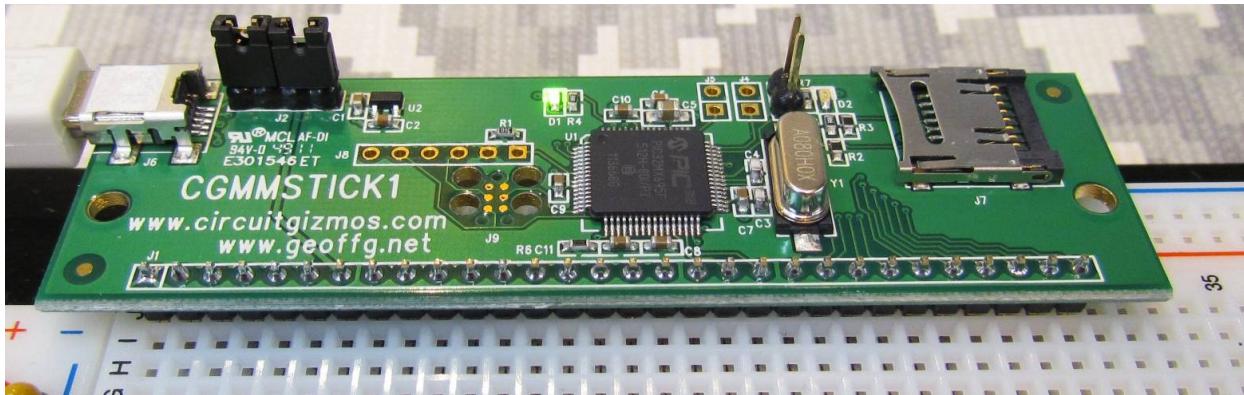


Figure 7: CGMMSTICK1 mounted in positions 1-30 of the author's solderless breadboard.

The picture shows a solderless breadboard that is set up with the CGMMSTICK. The CGMMSTICK is plugged in to the solderless breadboard so that the numbered pins correspond to the numbers on the long connector. The 20 input/output lines of the CGMMSTICK are the connections on the right side. Ground, 3.3 volt, and 5 volt connections are some of the remaining pins on the right.

A chapter at the end of this document lists the pin out of the CGMMSTICK.

The electrical hardware used for this very simple example is a single LED and 330 ohm resistor. One CGMMSTICK output line is connected to the LED, and the resistor connects the other end of the LED to ground.

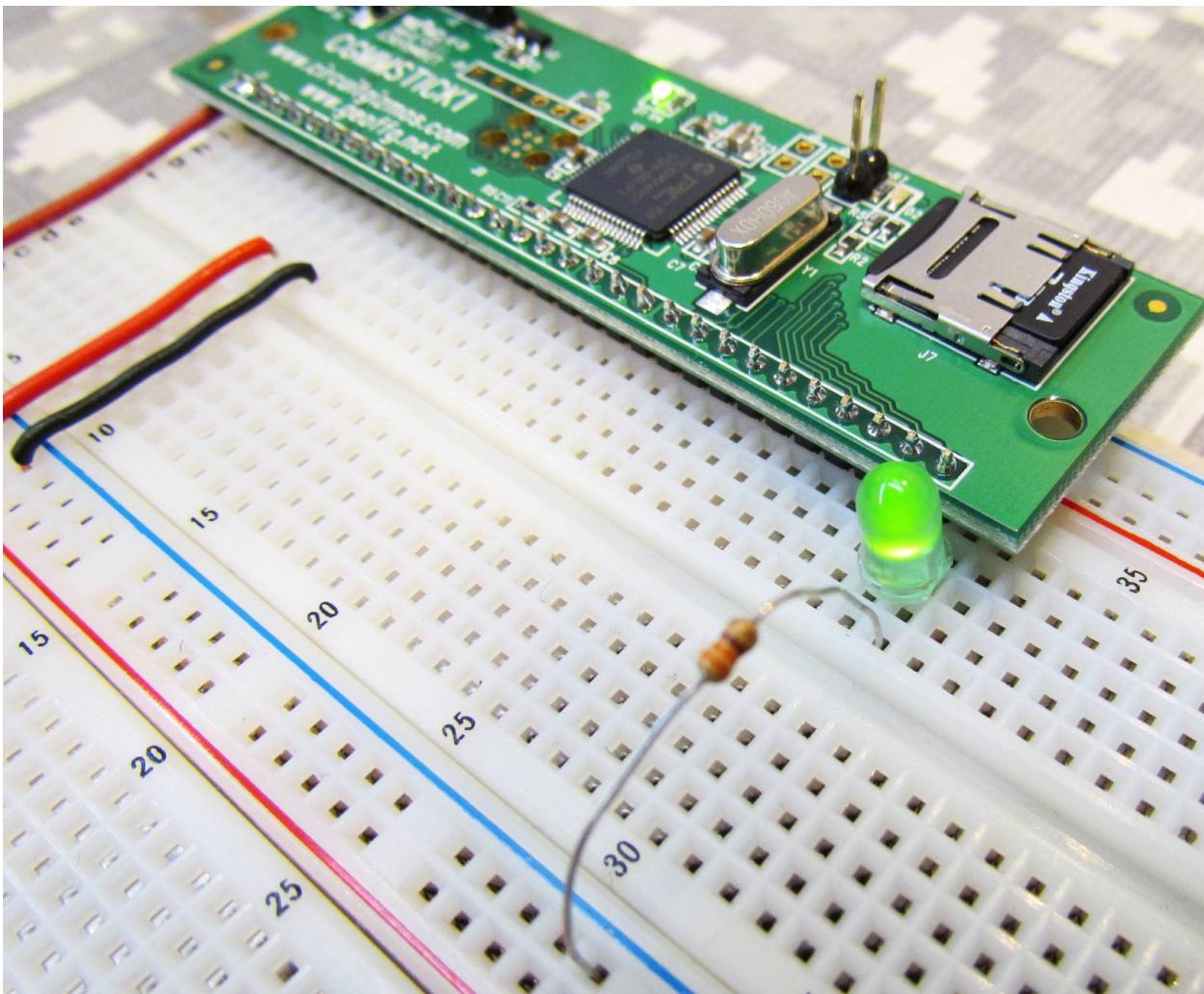


Figure 8: LED connected between I/O line #20 and a resistor (330 ohm) that goes to ground.

The picture shows the solderless breadboard with the LED example circuit. The I/O pin chosen was pin 20 (from software perspective which is pin 30 on the header), the connection farthest to the right on the CGMMSTICK. The breadboard is set up so that ground runs along the blue bus bar.

Pin 9 on JP1 (the CGMMSTICK header) is ground and attaches to the ground bus on the white solderless breadboard.

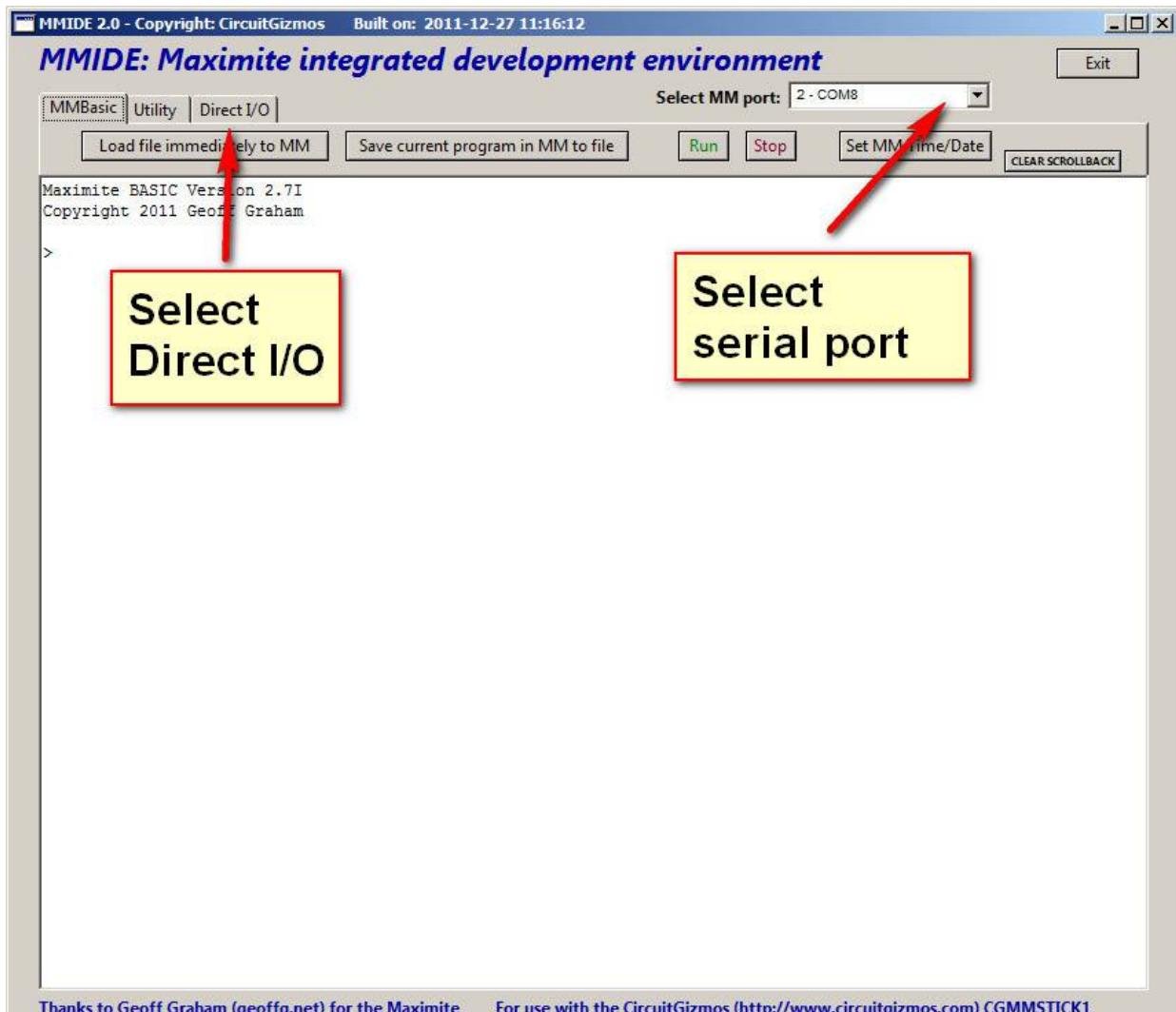


Figure 9: MMIDE - Select serial port and then click the Direct I/O page.

Normally your developed BASIC code would be run to turn this LED on and off. But before doing that, the MMIDE development tool can be used to verify the LED circuit. Run MMIDE and select the appropriate serial port for the CGMMSTICK. Select the page tab to select the Direct I/O page.

You can get MMIDE from the CircuitGizmos web site: <http://www.circuitgizmos.com>

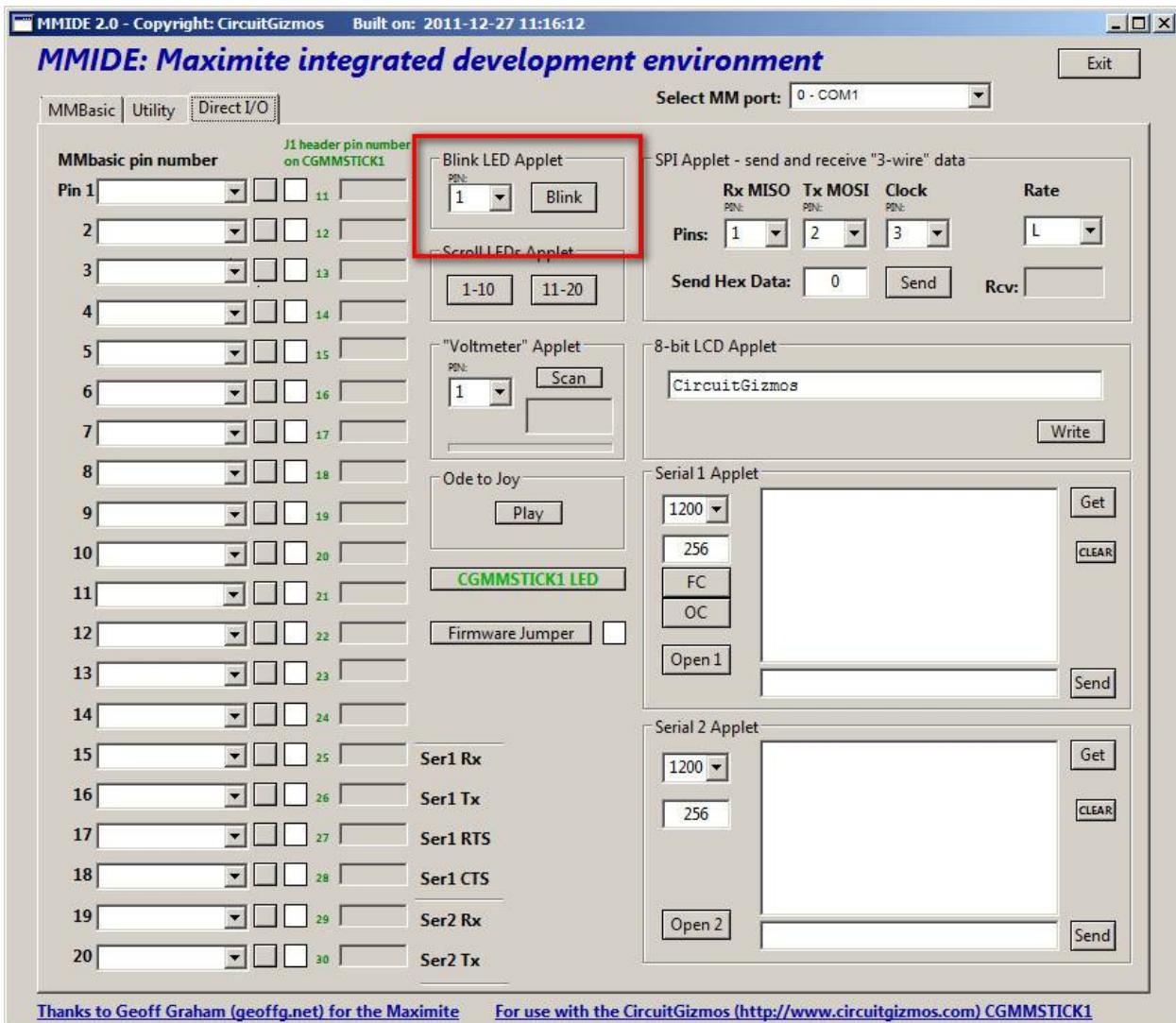


Figure 10: Use the Blink LED Applet to test the LED that you have connected.

On the Direct I/O page you can see a little application called the “Blink LED Applet”. There are only two things to do to run this applet. First you select the I/O port connected to the LED. The little applet defaults to I/O pin #1, but can be changed to any of the 20 I/O pins.

In this example the I/O port used for the LED is port #20. Port #1 is the default when you first run the program, just select “20” from the drop-down menu.

After you have selected the port, press the “Blink” button. Your LED should start blinking. If it doesn't, recheck your circuit. One common problem is a reversed LED – just flip it around.

If you press the “Blink” button a second time, the LED blinking will cease. MMIDE sends commands directly to the CGMMSTICK in order to make this little applet work. The applet sets the port line to be an output port, and then sends commands

to turn that line on and off over and over. When done, make sure that you turn the blinking LED off.

I'll mention again here that you might have wired up the LED circuit and could be clicking along with these instructions if you already have the CGMMSTICK and drivers installed, or you might just be reading along to learn what this version of a Maximite can do. The setup/installation process will be talked about in more detail later.

So you've just tethered the CGMMSTICK to your computer, added a simple LED circuit to it, and controlled that LED with MMIDE. But now what? Isn't the little Maximite supposed to be the brains, not a big old computer running MMIDE? It is. To make it the brains, all you have to add is some BASIC code that you write and then run on the CGMMSTICK.

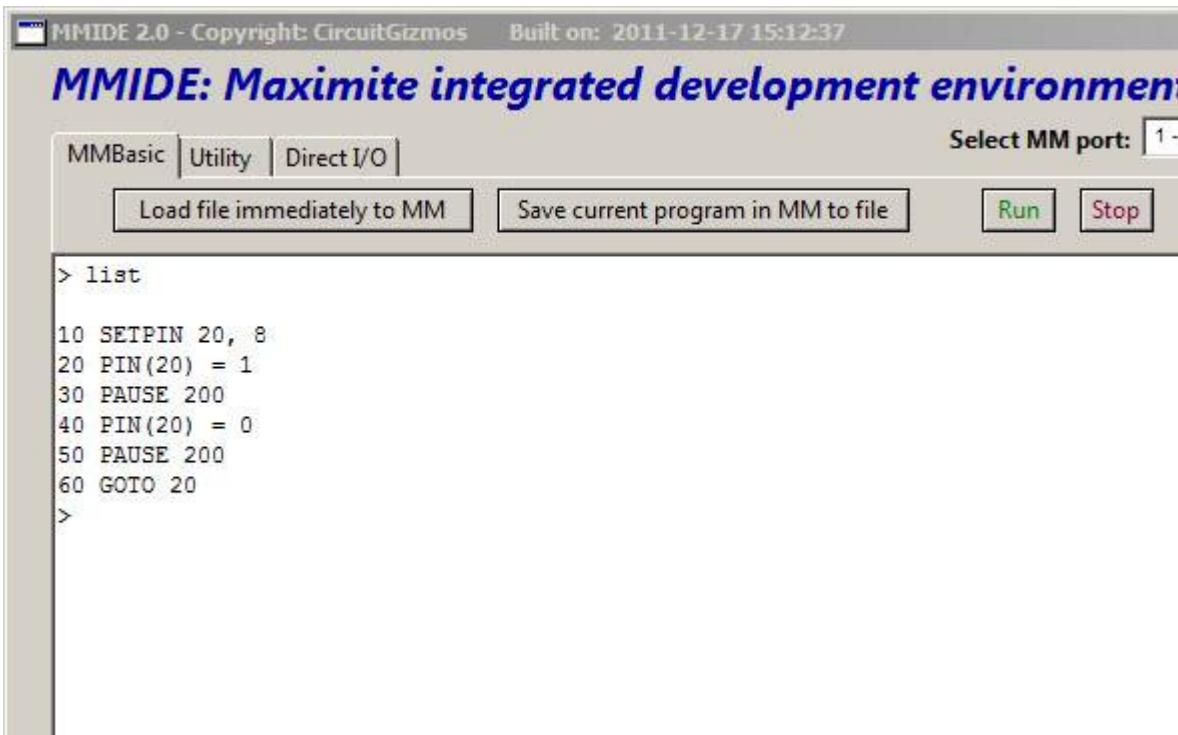


Figure 11: LED blinking BASIC program

If you return to the MMIDE page tab that says "MMBasic" you will be connected to the Maximite in a mode where you can type in your BASIC programs. This BASIC program in this example will do essentially the same thing as the MMIDE Applet did. Here is the program for you to enter:

```
10 SETPIN 20, 8
20 PIN(20) = 1
30 PAUSE 200
40 PIN(20) = 0
50 PAUSE 200
60 GOTO 20
```

This can also be done without line numbers in a text editor and downloaded to the CGMMSTICK. Downloading is explained later. Here is what code *without* line numbers may look like:

```
' Set pin direction
SETPIN 20, 8

' Repeat forever
DO
    ' Set pin high and pause
    PIN(20) = 1
    PAUSE 200

    ' Set pin low and pause
    PIN(20) = 0
    PAUSE 200
LOOP
```

You can run the program by entering the command "RUN" or you can click the green Run button. If you have entered the program correctly, the LED should be blinking while the program is running.

You can stop the blinking by typing Control-C or by clicking the red Stop button in MMIDE.

You can look up the commands in the program to see in detail what the program does at each step. In short, the program does about the same thing that the MMIDE "Blink LED Applet" does. The BASIC program sets the line (20) to be an output line, turns the line on, waits a little, turns the line off, waits a little, and then repeats.

This program can be saved to memory on the CGMMSTICK to be run again later. It can even be made to run automatically when the CGMMSTICK is powered up, so you would then have a fancy and overly sophisticated LED blinder. This fancy and overly sophisticated LED blinder has given you a good taste of what the CGMMSTICK Maximite can do.

Maxomite / CGMMSTICK / CGCOLORMAX

Introduction

As I mentioned in the preface, the Maximite is a little computer that does fun stuff! It is the computer that I have wanted for years. It is a computer meant for interfacing with electronic projects.

It is a “retro 1980s” computer with modern features such as SD card program / data storage instead of floppies and a modern version of BASIC with commands specifically for interfacing to circuits.

Geoff Graham (<http://geoffg.net>) created this little gem. I'll describe his original Maximite a little bit before the CGMMSTICK and CGCOLORMAX are described. From Geoff's web page on the Maximite:

The Maximite is a small and versatile computer running a full featured BASIC interpreter with 128K of working memory.

It will work with a standard VGA monitor and PC compatible keyboard and because the Maximite has its own built in SD memory card and BASIC language you need nothing more to start writing and running BASIC programs.

The Maximite also has 20 input/output lines which can be independently configured as analog inputs, digital inputs or digital outputs. You can measure voltage, frequencies, detect switch closure, etc and respond by turning on lights, closing relays, etc - all under control of your BASIC program.

The full story of the Maximite is here: http://geoffg.net/Maxomite_Story.html

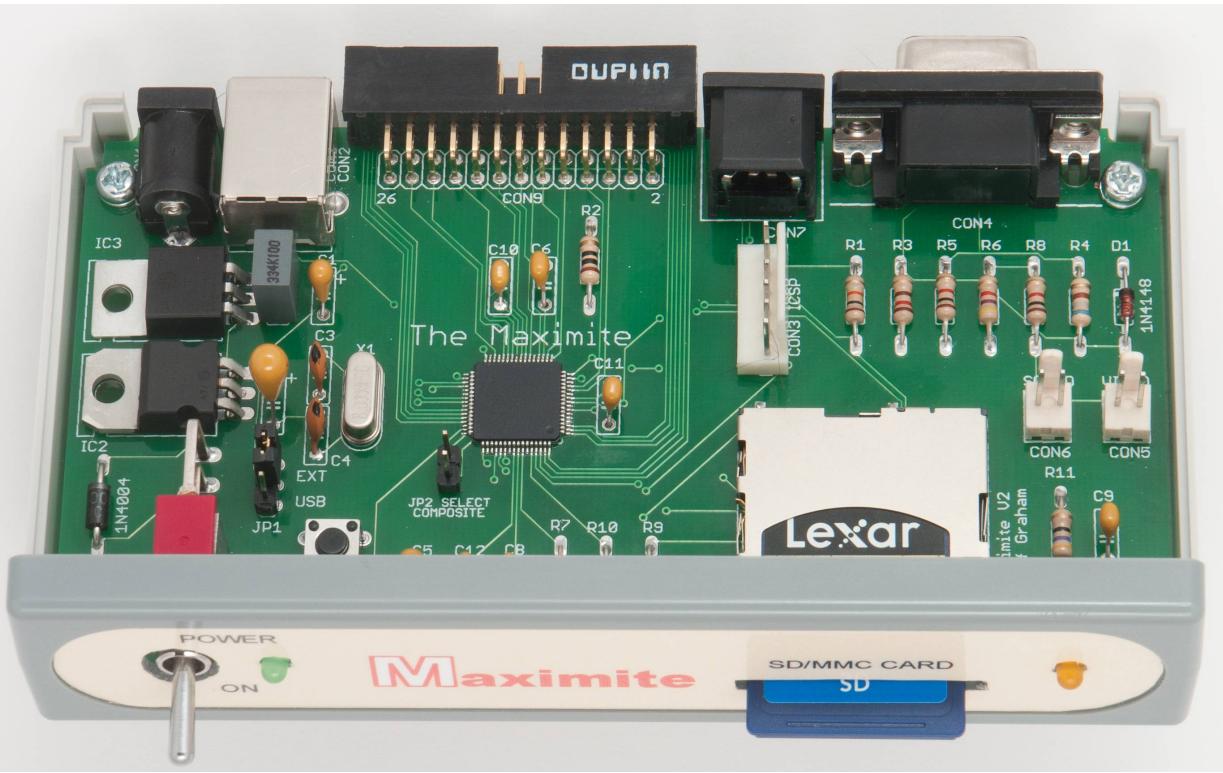


Figure 12: The original Maximite by Geoff Graham

Pictured above you can see the original Maximite in an enclosure. The picture has the cover off and shows some of the features of the design. Sticking out the front of the case is a full-size SD card for program and data loading and storage. That reminds me a great deal of early 1980s home computers with floppy drives sticking out the front, but the SD card can hold considerably more than the old floppy disks could. The original Maximite is approximately 6.1" by 3.6".

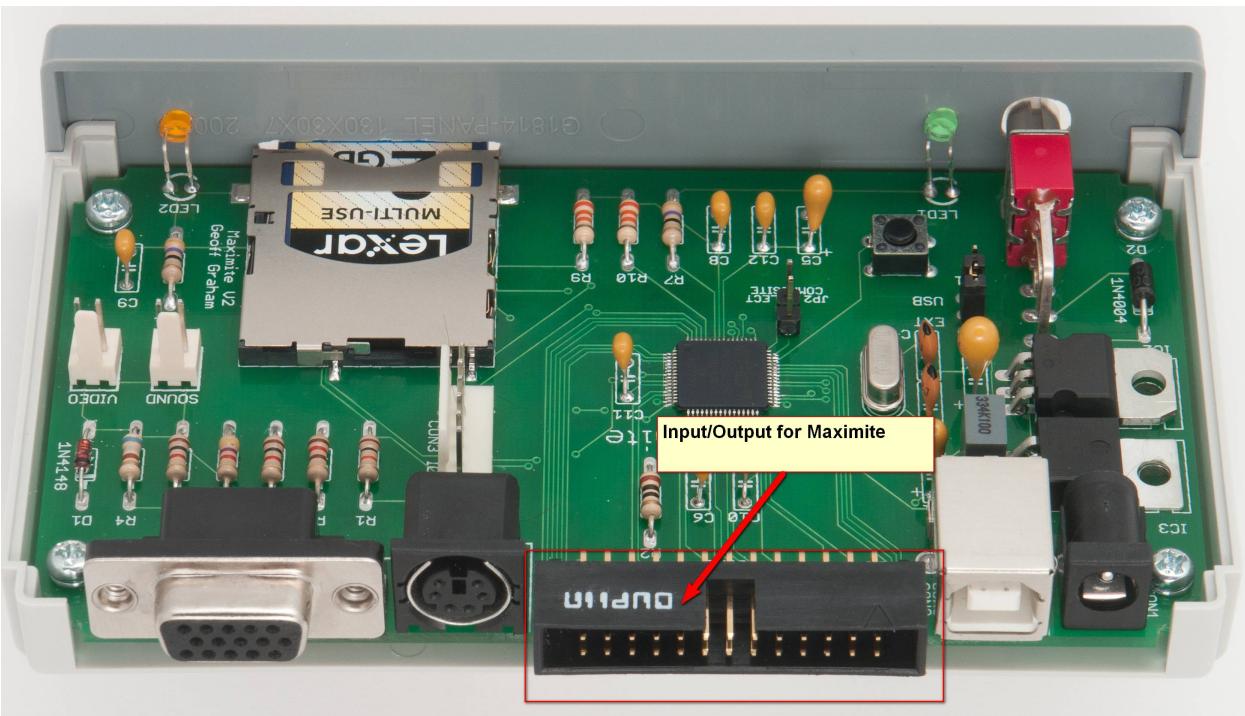


Figure 13: Input/Output lines on the original Maximite.

A rear view of the original Maximite shows where you can connect (from left to right) a VGA monitor, a PS/2 keyboard, I/O devices, USB (to a PC), and power.

When the Maximite is connected to a PC, the driver that is provided with the Maximite makes the hardware appear as a serial port. You can open a PC terminal program to connect to the Maximite through your PC keyboard and display.

You can also use the VGA port and keyboard port as the display and keyboard for this computer. Connect power, a PS/2 keyboard, and a VGA monitor and you have a stand-alone computer.

The row of pins on the back of the Maximite that I mentioned above as useful for "I/O devices" is where you can attach your own circuits to the Maximite to control.

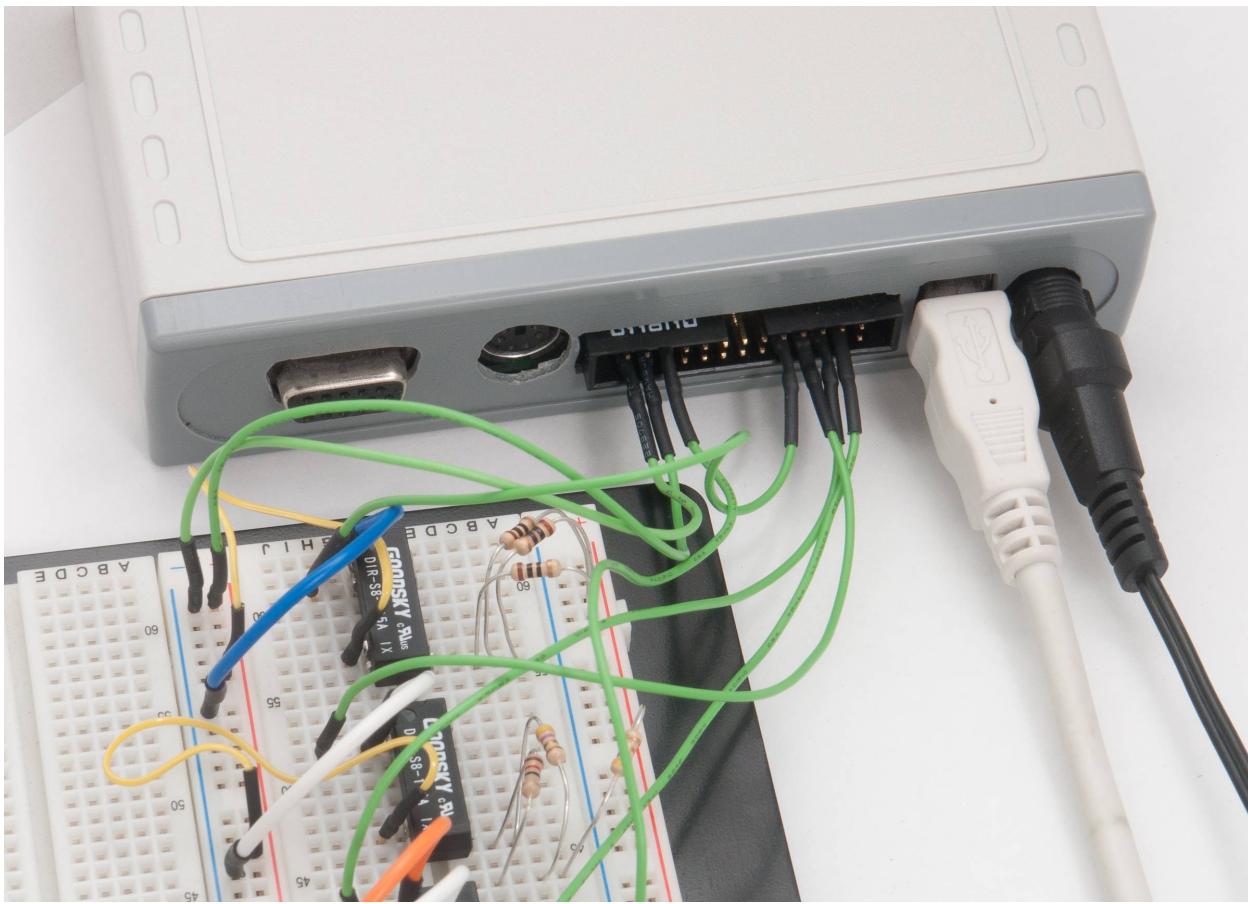


Figure 14: Prototyping with the Maximite. Jumper wires connect signals between the Maximite and the circuit being tested.

You can see in the picture above a Maximite that uses jumper wires to attach to a solderless breadboard. The jumper wires provide a signal pathway between the Maximite I/O lines (and the BASIC language that controls those lines) and a breadboard.

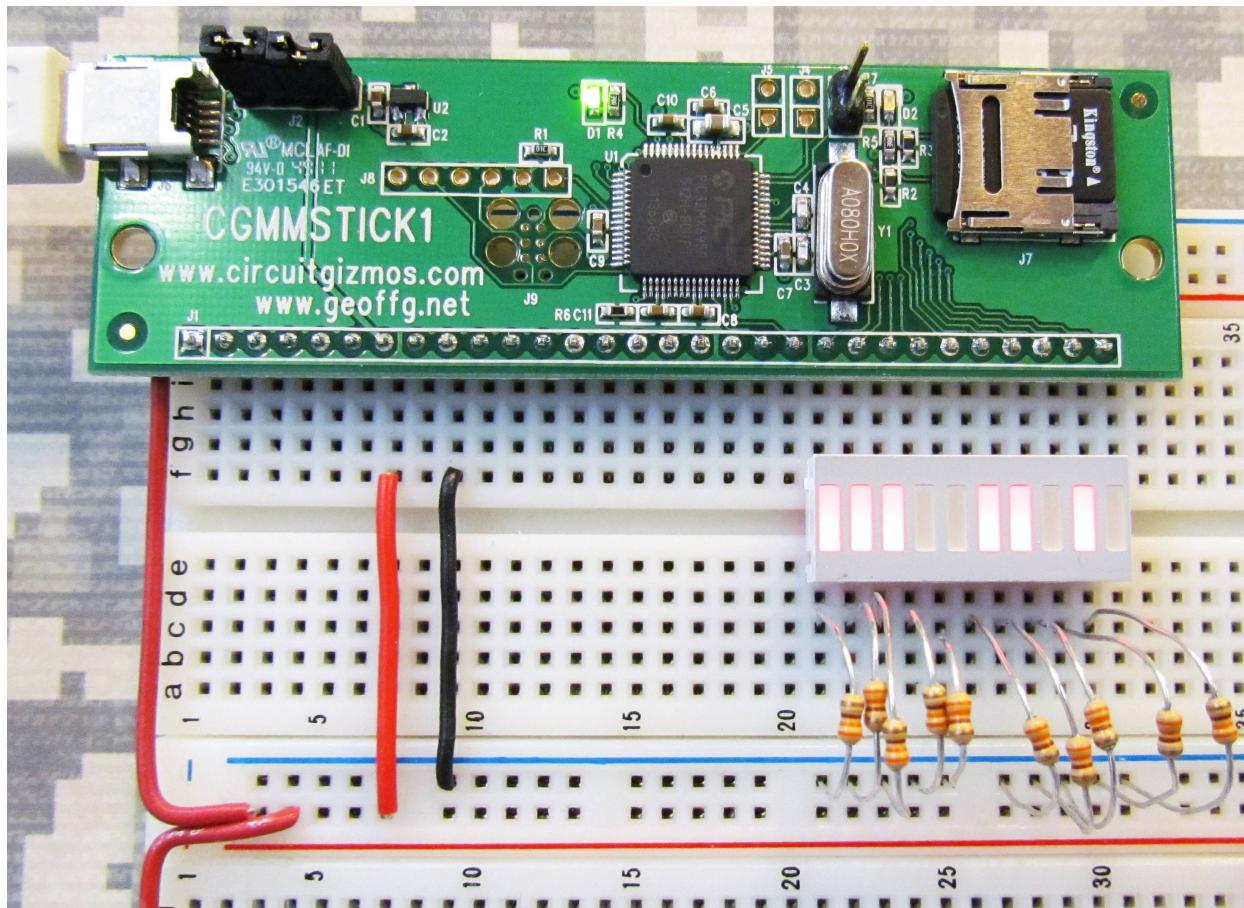


Figure 15: Prototyping with the CGMMSTICK1. The 'stick' plugs directly into a breadboard for convenience. The USB connection supplies 5V to the solderless breadboard.

The CGMMSTICK is a monochrome Maximite design intended for direct insertion into a solderless breadboard. This makes it great for circuit experimentation and control. The CGMMSTICK can be controlled directly from a PC, with a program (such as MMIDE) controlling all of the 20 Maximite I/O lines, or with 5V power provided to the CGMMSTICK through the solderless breadboard – the CGMMSTICK would run independent of the PC.

The CGMMSTICK can be embedded in a design as the control processor, automatically running a BASIC program.

The VGA and keyboard signals are also present on the 30 pin connector. The CGVGAKBD1 board brings these signals out to appropriate connectors.

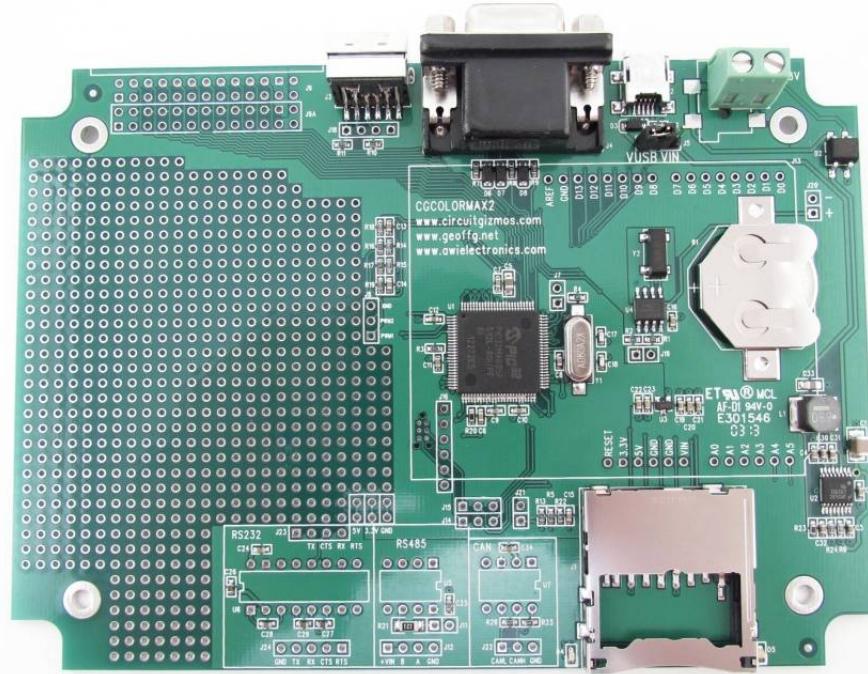


Illustration 16: CGCOLORMAX2 has VGA and keyboard connections, a real-time clock, full-size SD card, and support for interface circuitry.

The CGCOLORMAX is a color Maximite design that has VGA color and supports Arduino Shields. The CGCOLORMAX has all of the capabilities of the original Maximite and a similar form factor, but also includes an area for prototyping circuits as well as RS232 and RS485 circuits. This design has an on-board battery-backed clock and a full-size SD card.

The COLORMAX1 has surface-mount RS485 and RS232 circuits. The COLORMAX2 has optional DIP-package RS232, RS485, and CAN devices.

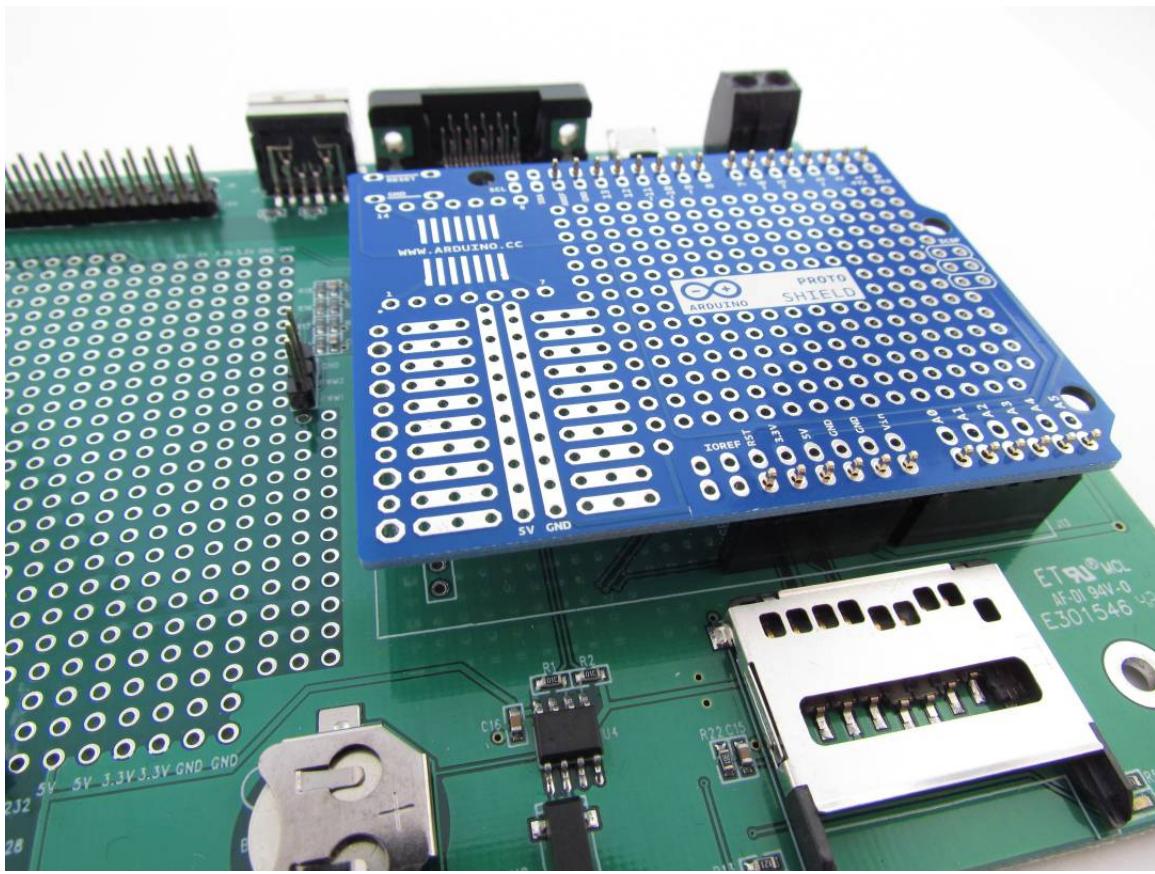


Figure 17: The CGCOLORMAX has room for shields mounted on the PCB. Here is a prototype shield mounted on the CGCOLORMAX1.

If you wish to add additional circuitry, it is easy to expand the hardware capabilities of the CGCOLORMAX by adding commonly available Shields.

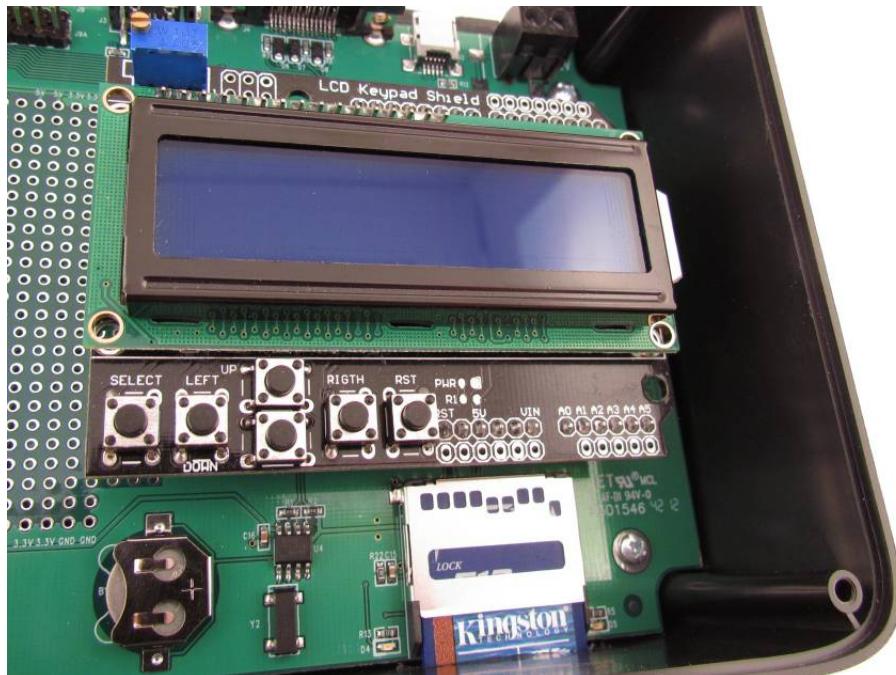


Figure 18: CGCOLORMAX1 mounted in an enclosure with a Shield in place. This is an LCD and button shield.

There are a large number of Shields available for Arduino devices. With appropriate MMBasic software they can all be used with the CGCOLORMAX.

Shields allow you to expand the hardware capabilities of the CGCOLORMAX simply by inserting a shield with the hardware that you need onto the Shield footprint.

Need an LCD display and buttons? The LCD Shield example in a later chapter shows example code.

Need relays? I/O expansion? Touch-screen graphic color LCD? Motor control? GPS interface? Backup battery? Just snap a shield in place.



Figure 19: All of the color video modes demonstrated by a CGCOLORMAX1 displayed on a VGA monitor. This picture is a CGCOLORMAX with an LCD shield attached and connected to a keyboard.

The CGCOLORMAX is design-compatible with the Colour Maximite. The CGCOLORMAX hardware offers more features.

CGMMSTICK / CGCOLORMAX Setup and MMIDE

Setup of a CGMMSTICK or a CGCOLORMAX is pretty simple. The two products are used in two general ways: either stand-alone, or tethered to a computer.

Stand-alone means that you don't need a PC to run the CGMMSTICK or CGCOLORMAX. Provide an appropriate source of power and the board will run. Add a display and keyboard to the CGCOLORMAX, or display, keyboard, and CGVGAKBD adapter to a CGMMSTICK and you have a standalone computer to program.

Tethered to a computer through the USB connection and (with the right driver) the CGMMSTICK/CGColorMAX appears as a serial port. Open a terminal program (or MMIDE) and you have a window into the CGMMSTICK/CGColorMAX. The program acts as a text display and keyboard input to the board.

You can switch between these two ways of using the CGMMSTICK or CGCOLORMAX as you see fit.

CGMMSTICK Setup

The CGMMSTICK that you have purchased has a row of 30 connections running along one edge as solderable wire points.

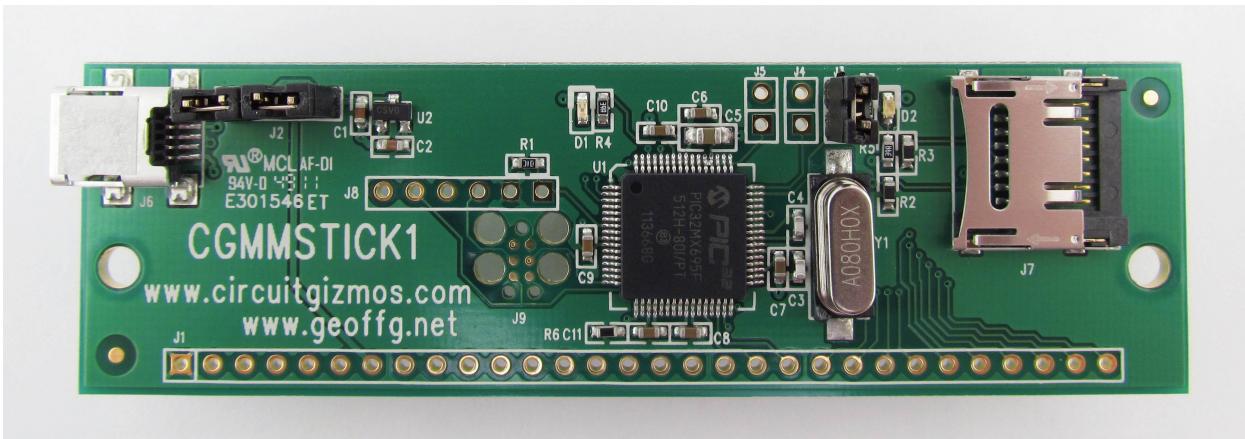


Figure 20: CGMMSTICK1 with the row of 30 connections used to interface to electrical circuits.

These connections are used for interfacing to your circuits. If you have a project where you would like to connect your circuits directly to the CGMMSTICK, the J1 row of 30 pins can have 20 gauge wire soldered directly to the gold-coated pins.

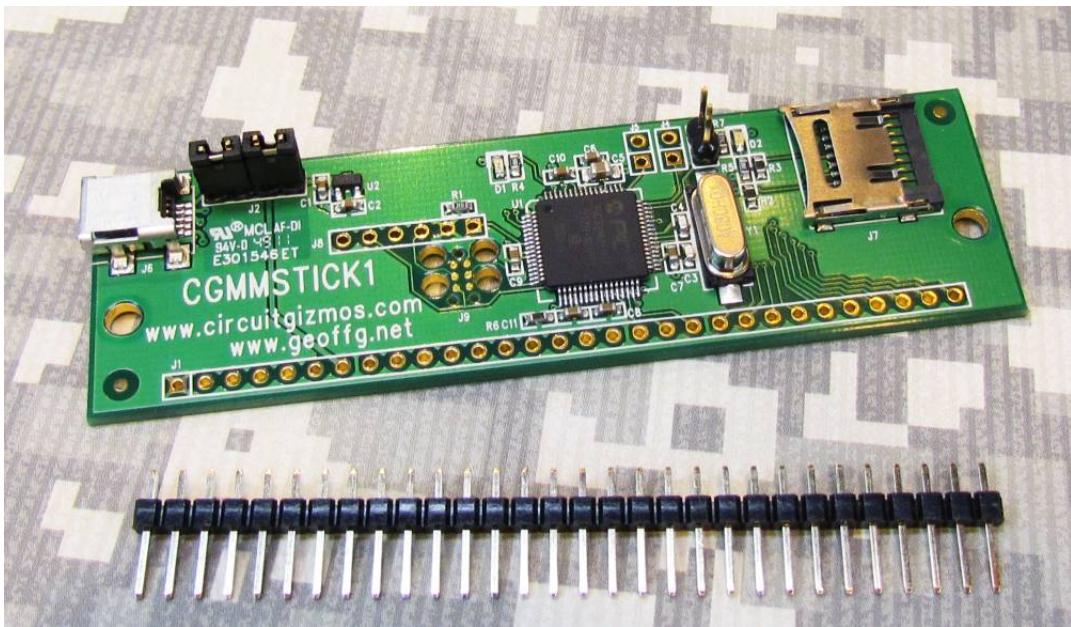


Figure 21: CGMMSTICK1 with a 30 pin header.

If you intend to use the CGMMSTICK with solderless breadboards, as the projects in this document demonstrate, then a 30 pin header is available from CircuitGizmos to solder in place.

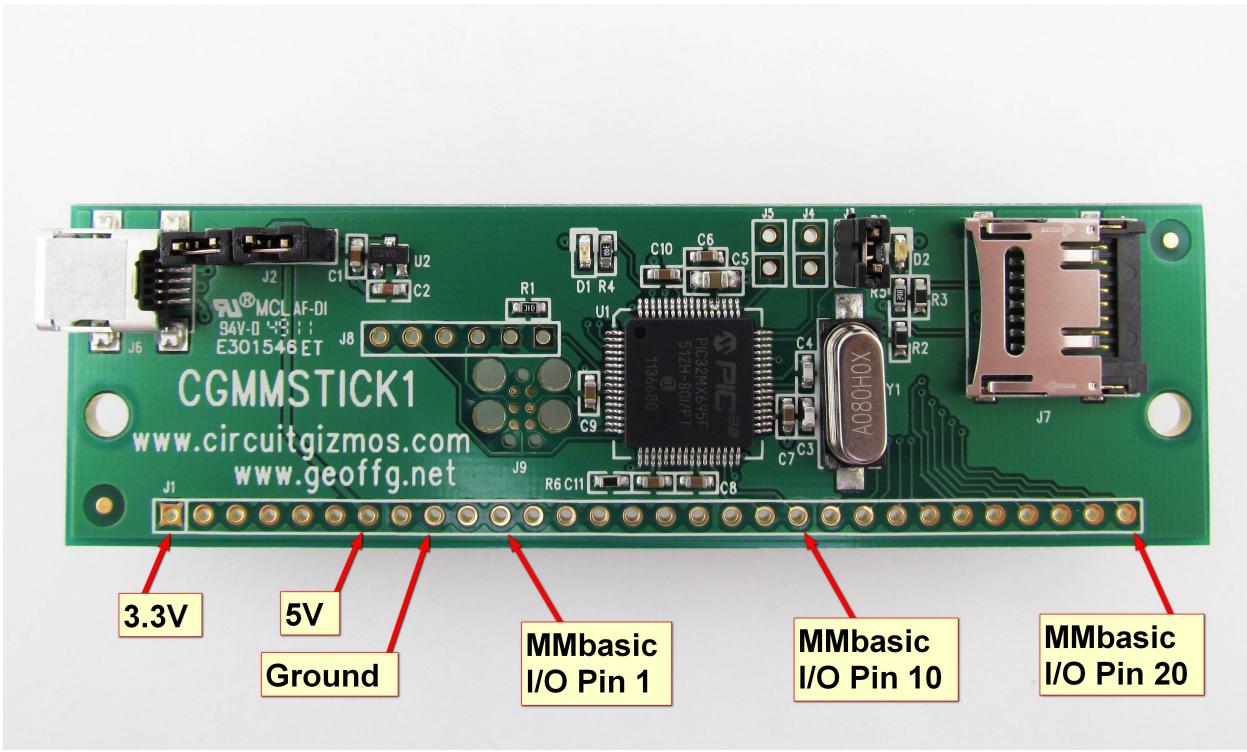


Figure 22: Some of the signals that you can connect to on the CGMMSTICK1.

Before you use the USB interface on the CGMMSTICK you will need to install the USB driver for the Maximite, so that the CGMMSTICK will appear to your PC as a serial port. The PC driver and instructions are included with the MMIDE download.

The MMIDE download is available for free from <http://www.circuitgizmos.com> More information about running MMIDE (there is no installation, just copied files) is described later.

Once you have the drivers installed for the CGMMSTICK , plugging in the USB connection will create a new serial port on your PC. You can use various serial terminal programs to connect to that new serial port and communicate with the CGMMSTICK. You can also use the Maximite Integrated Development Environment – MMIDE – to develop CGMMSTICK projects.

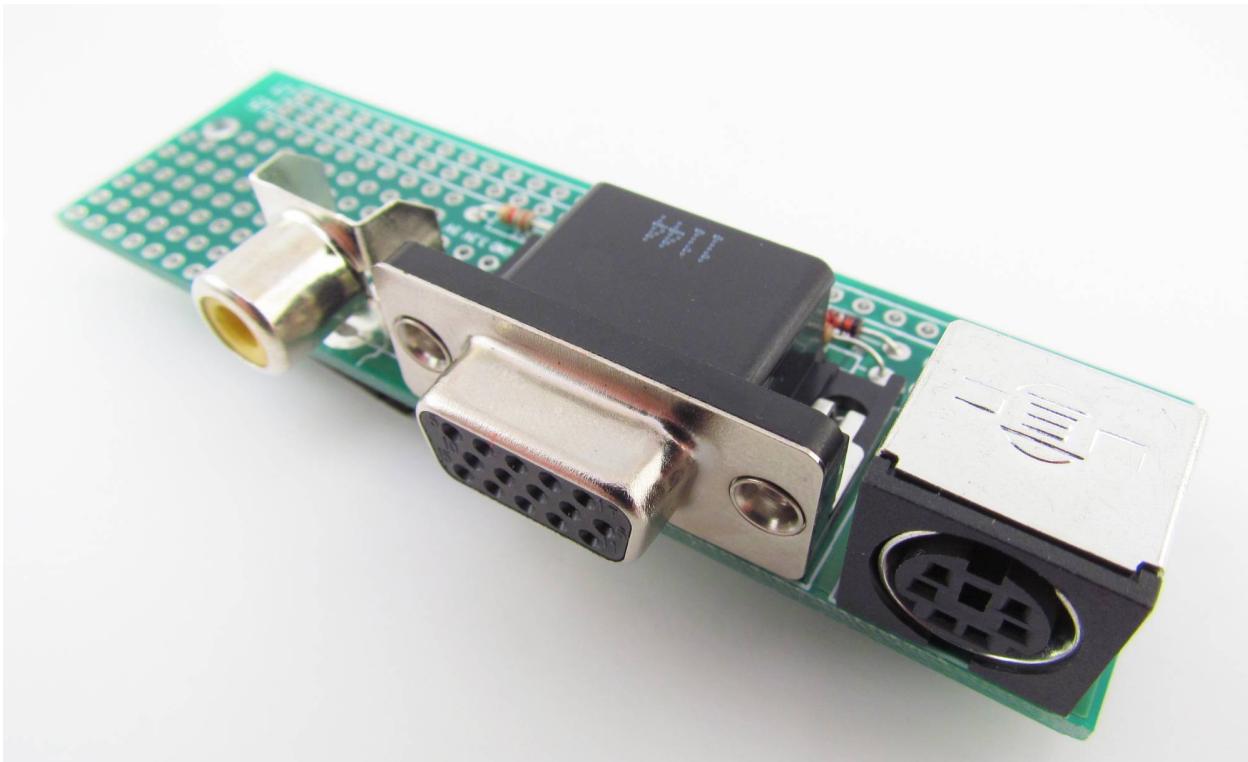


Illustration 23: The CGVGAKBD provides VGA and keyboard connectors for the CGMMSTICK.

If you don't want to use the CGMMSTICK tethered to your PC but want to use it as a stand-alone computer with a keyboard and VGA display, you can add the CGVGAKBD board to your CGMMSTICK to make a stand-alone computer.

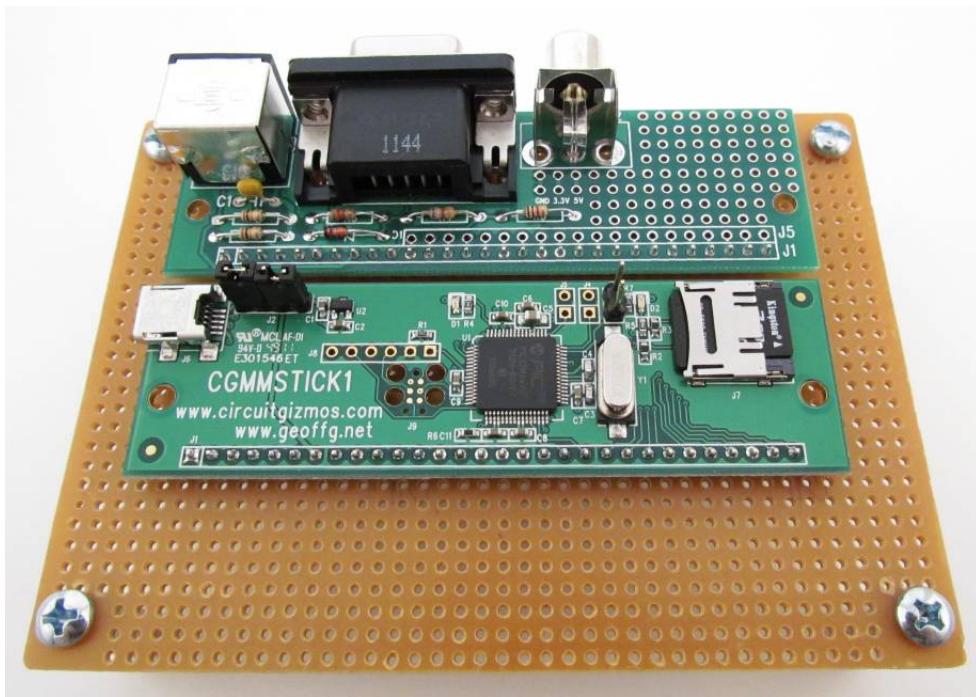


Illustration 24: A CGVGAKBD used with a CGMMSTICK. Both boards are mounted on a protoboard. Underneath the boards are wired together pin 1 to pin 1, 2 to 2, etc.

Some of the lines on the 30 pins of the CGMMSTICK are meant as signal lines for VGA and keyboard connections.

You can get away with wiring pins 1-10 on both devices together (A connection from pin 1 on the CGMMSTICK to pin 1 on the CGVGAKBD. Another from pin 2 to pin 2, etc.) Or you can just wire all 30 lines from one board to the next. Whatever is convenient.

You need to provide 5V DC at at least 160 mA to power this setup. 5V from an external supply (5V regulated wall wort will work) connects to the supply pins of the 30 pin J1.

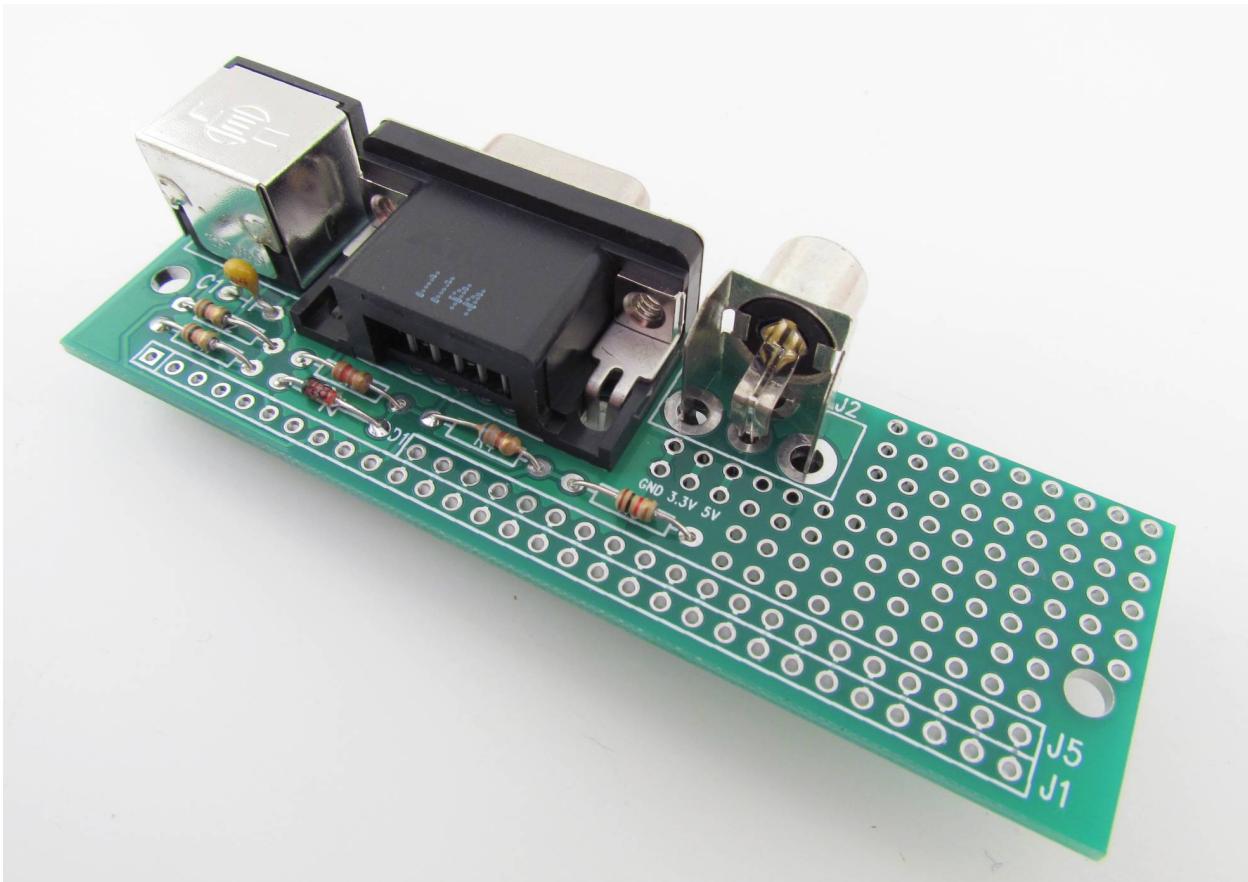


Illustration 25: The back side of a CGVGAKBD. Note pin 1 of J1 on the far left.

If you look at the back side of the CGVGAKBD you will see the J1 header which is meant to be a one-to-one duplicate of the J1 header on the CGMMSTICK.

Also note J5 which duplicates 20 of the J1 lines. These are the I/O lines that MMBasic can control. There is a small “sea-of-holes” on the board – a small prototyping area where you can add your own circuit. Perhaps something like a D/A converter, a relay, or small buttons could be wired from that area to some I/O lines on J5.



Illustration 26: A CGMMSTICK/CGMMVGAKBD combination as a stand-alone computer.

Once wired to a CGMMSTICK, this board lets you attach a VGA display for monochrome video. As an alternate display, you could also attach a composite monitor to the yellow RCA connector (requires a jumper on the CGMMSTICK board).

A keyboard can be connected to the PS/2 keyboard jack for input to the CGMMSTICK.

In the photo above 5V needed to power this stand-alone setup is provided through the USB connection. There is no “data” USB connection to the PC in this case – it was convenient for the photo to use a USB cell phone charger that provides 5V at 500 mA.

CGCOLORMAX Setup

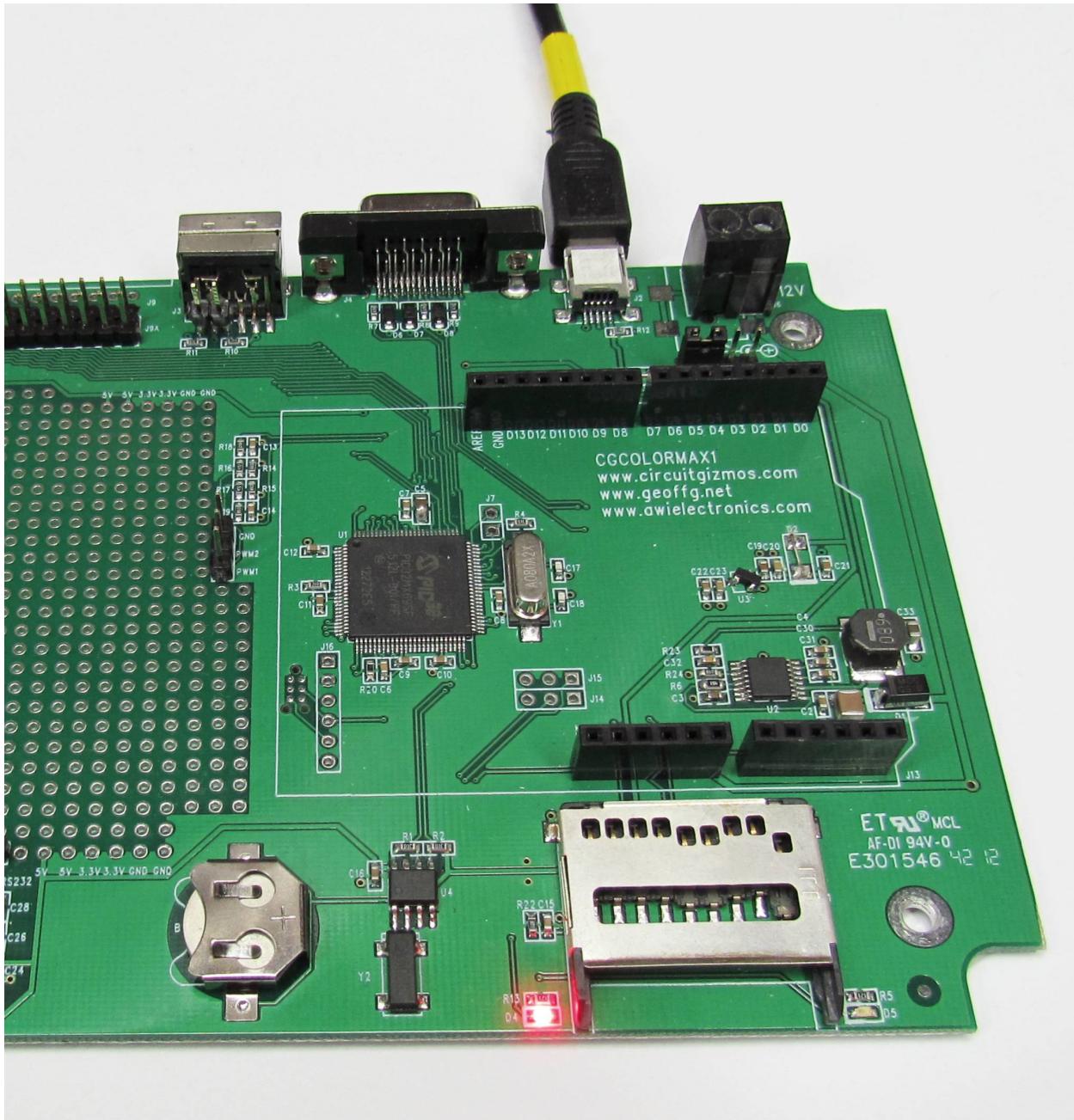


Illustration 27: CGCOLORMAX powered through USB. Tethered to the PC through the USB port. CGCOLORMAX1 shown here, but the same thing applies to the CGCOLORMAX2.

Before you use the USB interface on the CGCOLORMAX you will need to install the USB driver for the Maximite, so that the CGMMSTICK/CGCOLORMAX will appear to your PC as a serial port. The PC driver and instructions are included with the MMIDE download.

The driver is needed if you use the USB connection to your PC. The driver makes the board appear to the PC as a serial port.

Once you have the drivers installed for the CGCOLORMAX, plugging in the USB connection will create a new serial port on your PC. You can use various serial terminal programs to connect to that new serial port and communicate with the CGCOLORMAX. You can also use the Maximite Integrated Development Environment – MMIDE – to develop CGCOLORMAX projects.

On the CGCOLORMAX1 the power selection header should have a jumper in place on the left two pins. That is to say that the jumper in J5 would be in the "VUSB" position. The 5V from USB would then power the board.

On the CGCOLORMAX1 the power selection header has a total of 4 pins. On the CGCOLORMAX2 the power selection header has a total of three pins. Middle pin and left pin would be VUSB.

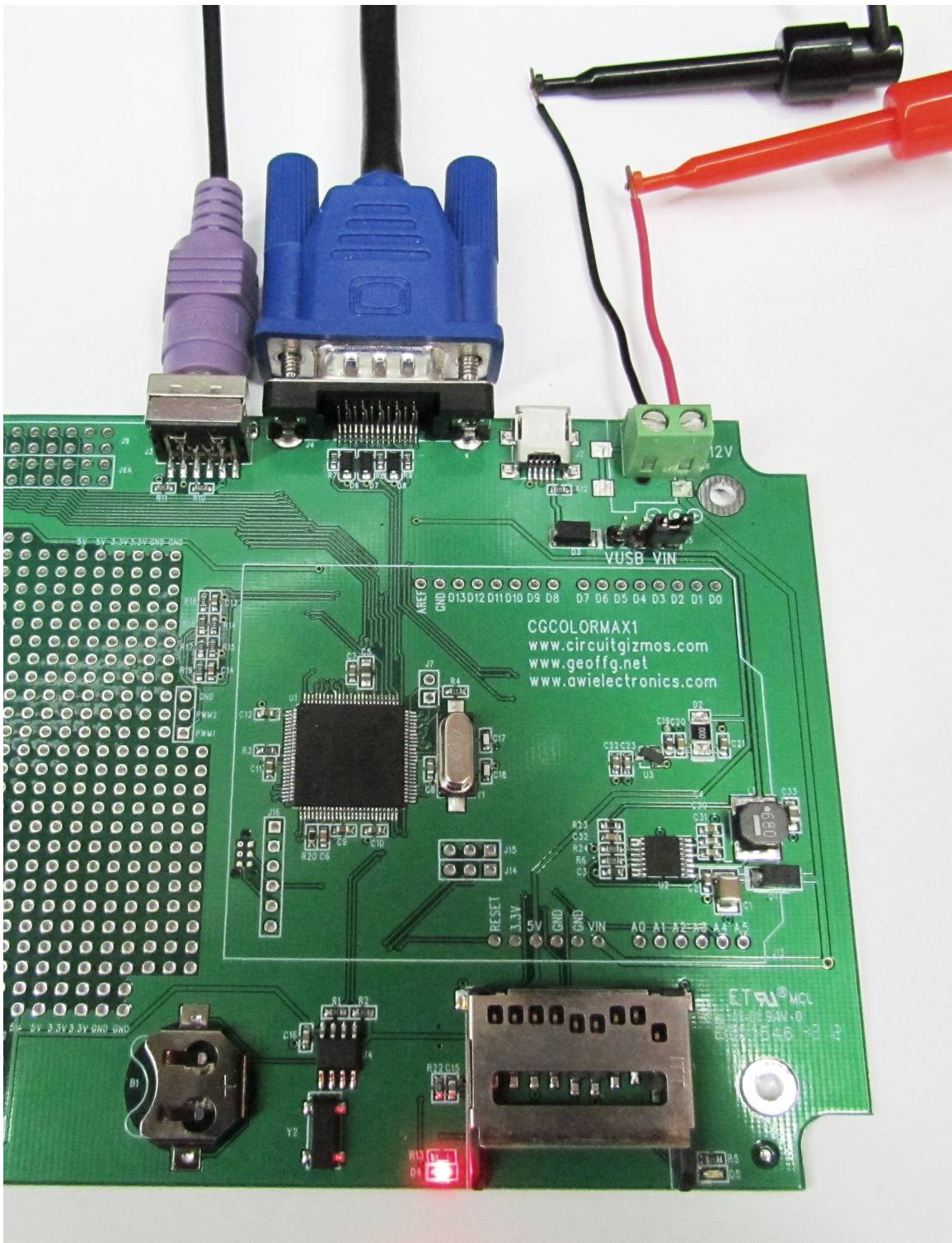


Illustration 28: CGCOLORMAX powered through the 12V power connections. Keyboard and VGA connections allow the board to be a stand-alone computer. CGCOLORMAX1 shown here, but the same thing applies to the CGCOLORMAX2.

If you don't want to use the CGCOLORMAX tethered to your PC but want to use it as a stand-alone computer with a keyboard and VGA display, you can power the board through the 12V power input connection and connect a VGA display and PS/2 keyboard.

On the CGCOLORMAX1 the power selection header should have a jumper in place on the right two pins (of the 4 total). That is to say that the jumper in J5 would be in the "VIN" position. The 8V – 18V DC from an external supply would then power the board.

On the CGCOLORMAX1 the power selection header has a total of 4 pins. On the CGCOLORMAX2 the power selection header has a total of three pins. Middle pin and right pin would be VIN.

MMIDE Setup

MMIDE is a great way to test out some of the functions of the CGMMSTICK or CGCOLORMAX. MMIDE runs without an installation program, as long as the "MMIDE Libs" directory and its contents are located in the same directory on your PC as the "MMIDE.exe" file.

When you run MMIDE it looks for all of the serial ports on the PC and lists them as available ports in the dropdown box "Select MM port." On my machine this happens to be the second port that is found and it is called "COM8." On your PC it is likely to be different.

Note: MMIDE is an evolving program. Features are added. Since the initial release the program has been improved to support the CGCOLORMAX. Many of the screen captures here look different from the current version of MMIDE which at the time of this writing is MMIDE 3.0d. The newest MMIDE is available for free from <http://www.circuitgizmos.com>

MMIDE 2.0+ works with the CGMMSTICK just fine. That version can still be used with the CGMMSTICK. MMIDE 3.0+ was updated for the CGCOLORMAX. The CGMMSTICK can also be used with the 3.0+ version - the extra features supporting the CGCOLORMAX will not function on the CGMMSTICK.

The CGCOLORMAX has twice as many I/O lines as the CGMMSTICK and the Direct I/O page in the program was updated to support that.

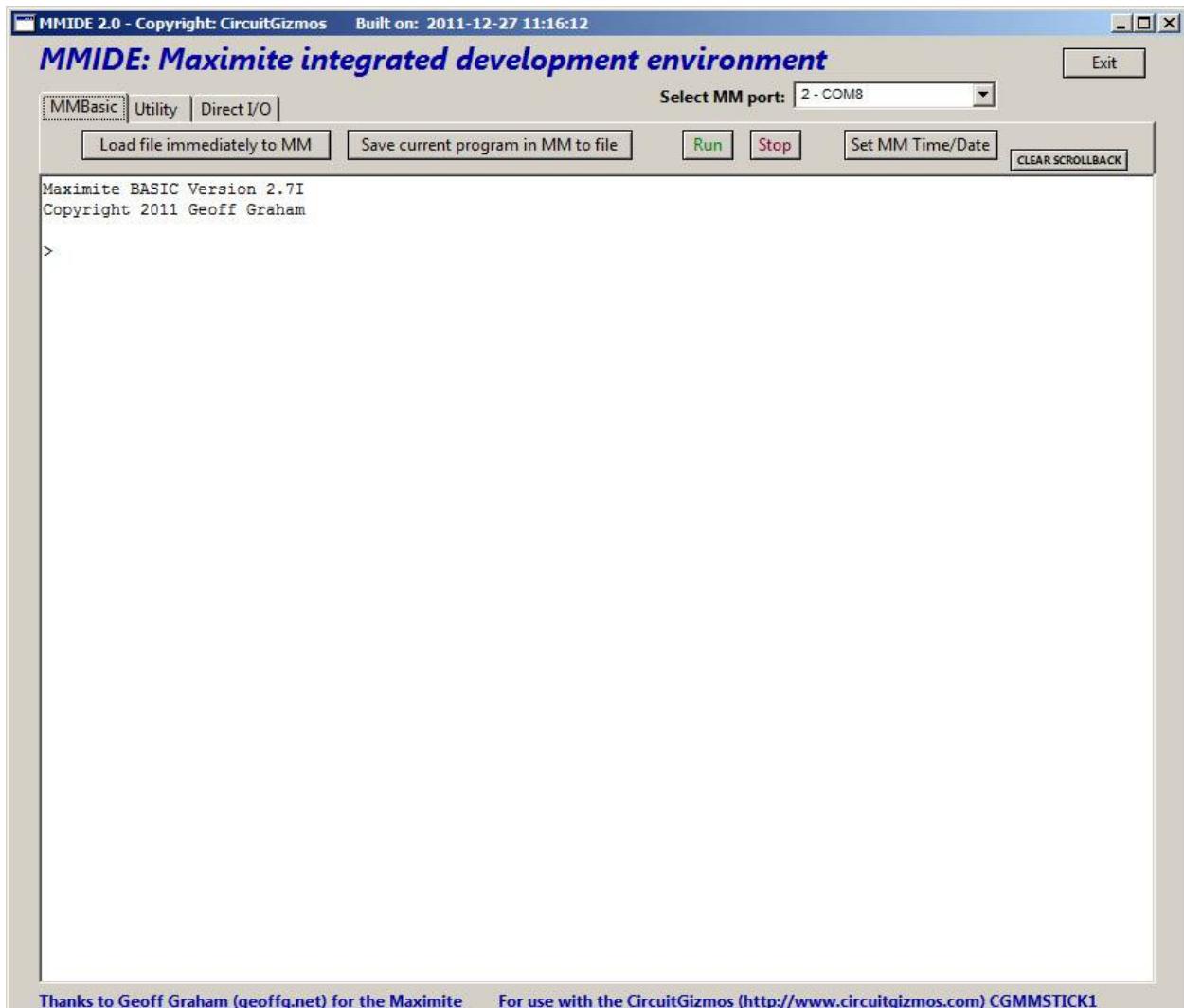


Figure 29: MMIDE interactive "MMBasic" page.

If you select the port associated with the CGMMSTICK/CGCOLORMAX, then the text area on the MMIDE MMBasic page will show the message that MMBasic prints when it connects. This is the MMBasic version and the copyright message from Geoff Graham.

The MMIDE MMBasic page acts as a serial terminal to the CGMMSTICK/CGCOLORMAX. You can type in and run a BASIC program in the text area as well as view the PRINTed messages from MMBasic.

There are six buttons on the MMIDE MMBasic page that assist in development of MMBasic programs.

The “Run” button runs whatever program is currently in memory on the CGMMSTICK or CGCOLORMAX. This button simply automatically types “run” followed by a carriage return (enter key) into the text window.

The "Stop" button stops whatever program is currently running on the CGMMSTICK/CGCOLORMAX. This button simply automatically types Control-C into the text window.

The "CLEAR SCROLLBACK" button erases the text area. It does not issue a CLS to MMBasic.

The "Load file immediately to MM" button lets you select a BASIC file from your PC and send it to the CGMMSTICK/CGCOLORMAX. It essentially types the program into the text window as if you were typing it. MMBasic has to be ready for it – in other words connected and NOT running a program.

The "Save current program in MM to file" copies whatever program is currently in the CGMMSTICK or CGCOLORMAX by running the LIST command, capturing that listing, and saving that to a file.

The MMBasic window will let you type directly into the command interpreter. You can type something like:

```
PRINT 2 + 2
```

Then after the last '2' on the line when you hit the enter key the interpreter immediately returns the result of that command. Note that the '>' is placed at the beginning of the line when the interpreter is ready for input. The examples in this book show input and output without that prompt character.

```
> Print 2 + 2
```

```
4
```

```
>
```

Illustration 30: MMBasic with prompts.

This is the result of the immediate command to print 2+2. The example in this text does not show the '>' prompt.

```
Print 2 + 2
```

```
4
```

Apparently 2+2 is still 4.

You can do the same thing by making the command to print the results of 2+2 into a short program with line numbers and then running the program. This is what that would look like in the window:

```
10 REM Is 2+2 = 4?  
20 Print 2+2
```

```
RUN
```

```
4
```

Since line numbers are not necessary (only a few examples in this book use them) you can also enter a program without line numbers using 'auto':

```
Auto
```

```
rem 2+2 once again  
print 2+2
```

```
run
```

```
4
```

```
list
```

```
Rem 2+2 once again  
Print 2+2
```

The short program is run and the result is still 4. The program was then listed to see what it looks like without line numbers.

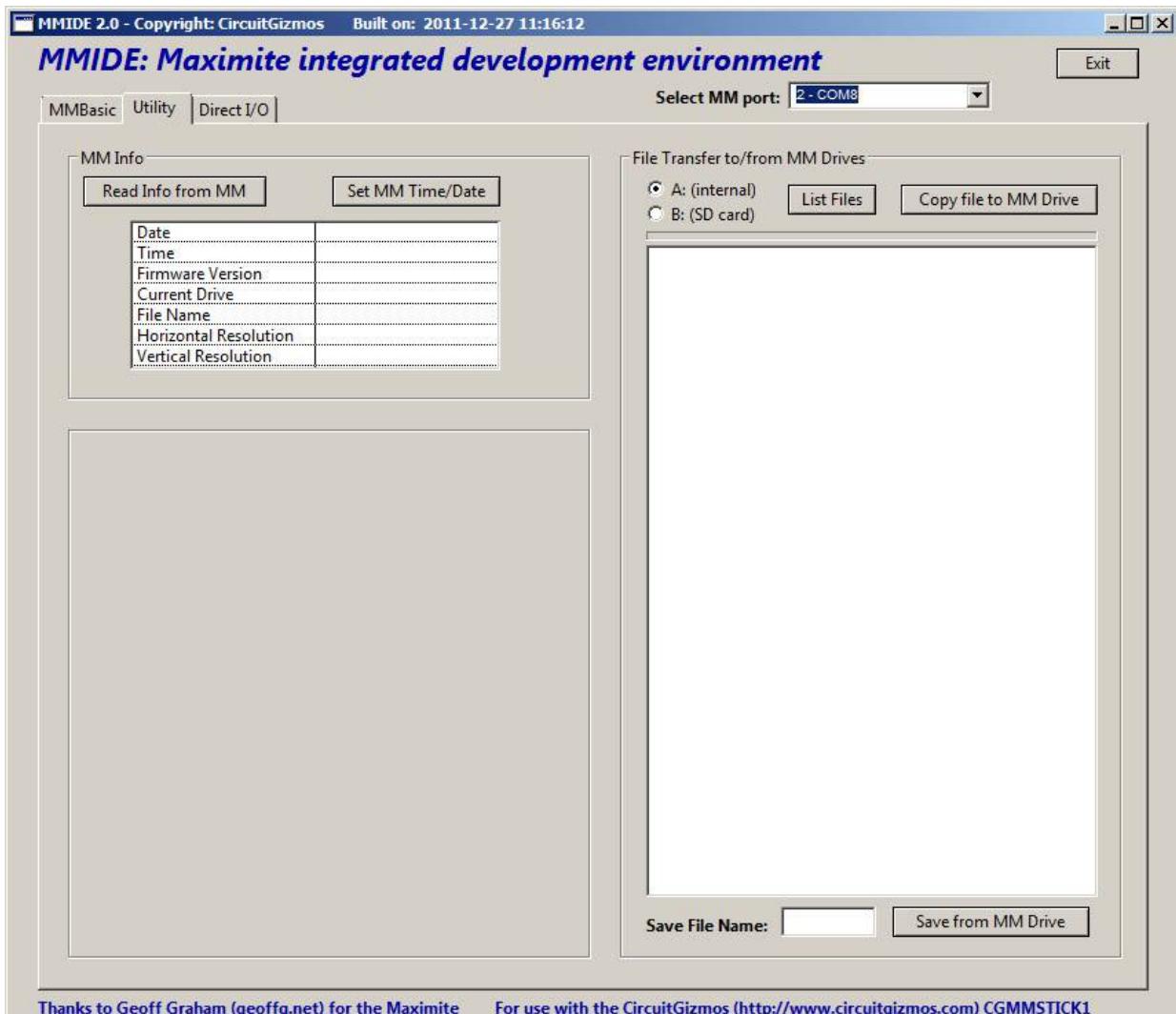


Figure 31: MMIDE Utility page. Read information from MMBasic and transfer files.

The MMIDE Utility page has a “Set MM Time/Date” button that gets the current time and date from the PC, and uses specific MMBasic commands to set the time and date on the CGMMSTICK. This works when MMBasic is ready to accept commands – not when a program is running.

The info area also has a button that grabs some information from the CGMMSTICK/CGCOLORMAX. It gets the date, time, firmware version, current drive, current file name (if there is one set), the horizontal resolution, and the vertical resolution.

The MMIDE Utility page also allows for the transfer of a file to and from the A: and B: drives. A radio button lets you select between the A: drive and the B: drive. The A: drive is internal to the microcontroller chip on the CGMMSTICK/CGCOLORMAX, and the B: drive is the micro SD card.

The “List Files” button will list the files that are on either the A: or B: drives.

The “Copy file to MM Drive” button will use the MMBasic XMODEM support to copy a file from your PC to the selected drive. This file can be an MMBasic program, or a data file.

The “Save from MM Drive” button will request the file that you've typed into the “Save file name” box from the CGMMSTICK or CGCOLORMAX using XMODEM and save that file to the PC.

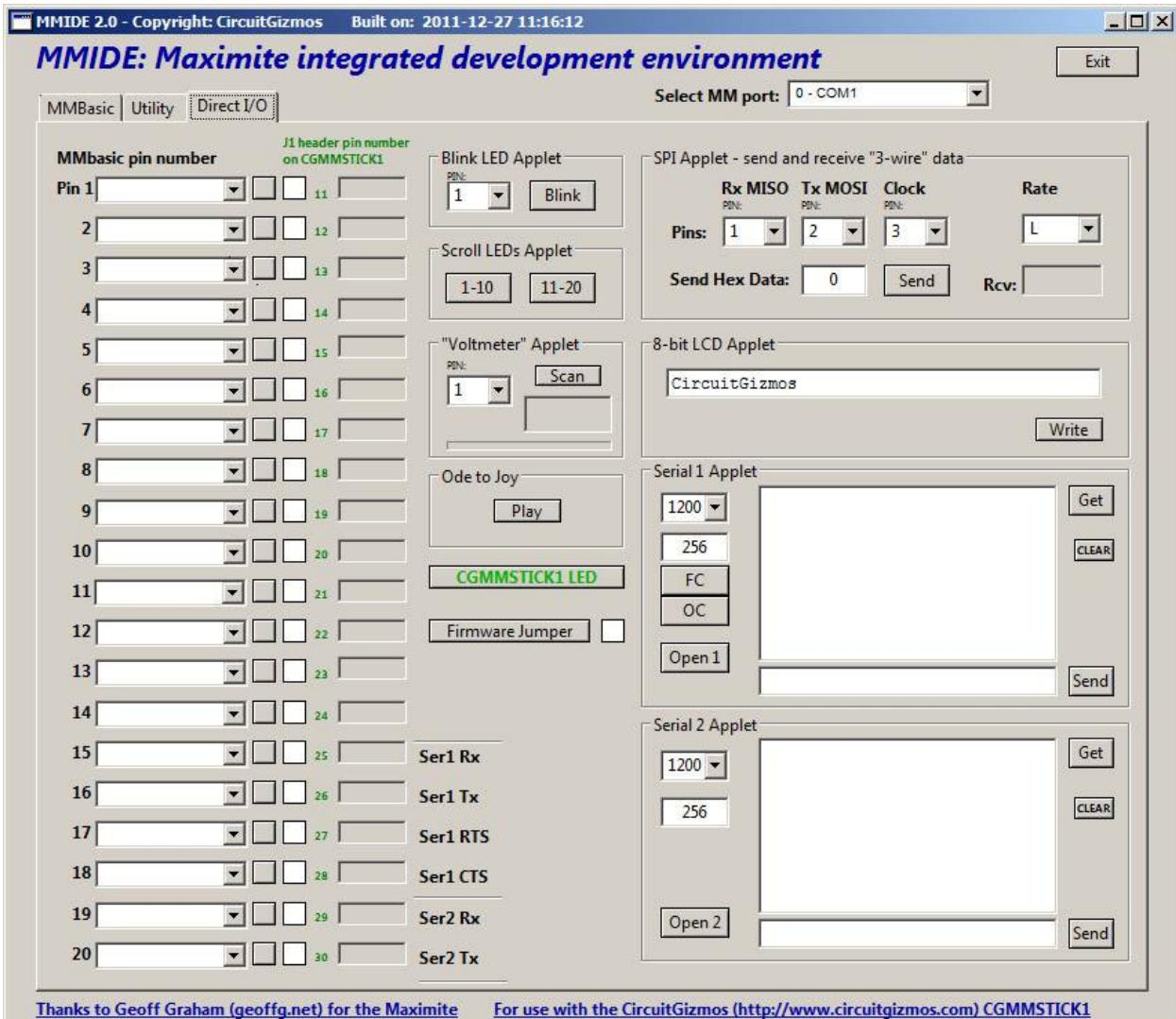


Figure 32: MMIDE's very helpful "Direct I/O" page.

The MMIDE Direct I/O page has a significant number of tools to assist in CGMMSTICK/CGCOLORMAX hardware and software development. These Applets send immediate commands to the hardware behind the scenes. Some of the routines transfer a small program that then runs on the CGMMSTICK/CGCOLORMAX hardware.

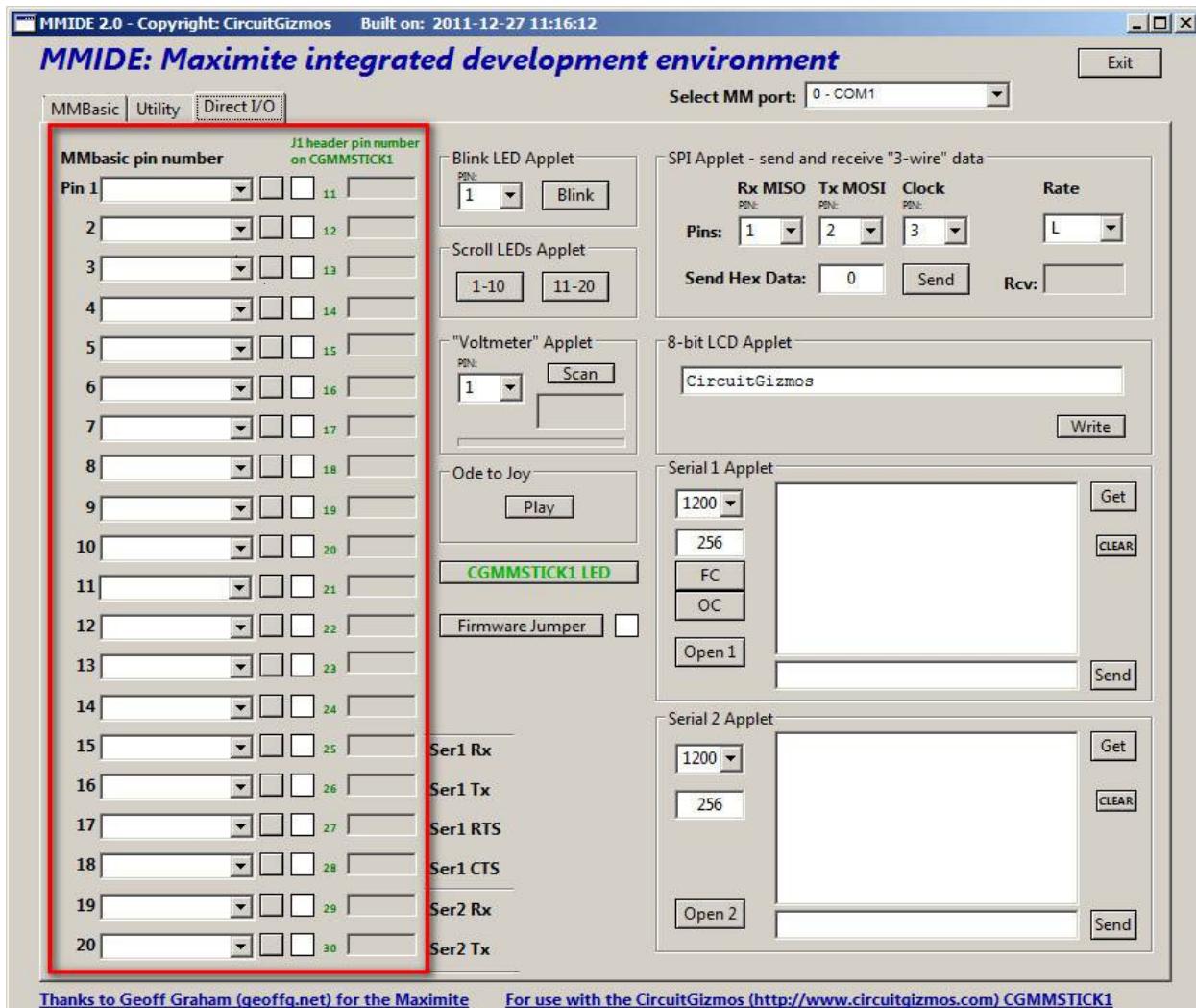


Illustration 33: MMIDE allows direct control of each I/O pin.

Highlighted above is the area of the page that lets all of the CGMMSTICK/CGCOLORMAX pins be manually read or written. Along the left edge in black text is the pin number as MMBasic would see them. The I/O pins number from 1-20 from the perspective of a MMBasic program. In green text a little to the right is the physical pin on the J1 header of the CGMMSTICK/CGCOLORMAX.

The drop-down box lets you select the pin function allowed for that pin. This follows the pin functions that the MMBasic SETPIN command uses. Some of the pins can be set differently than others. Some pins can be set to analog input, while other can not, for example.

When a pin is set to output, the square button will toggle the state of the pin. When the pin is set to a type of input (digital, analog, frequency) the square button reads that pin and displays the pin value in the text box next to that pin. There is also a

square color box that will change color to yellow for an analog input, red for a digital high, and black for a digital low.

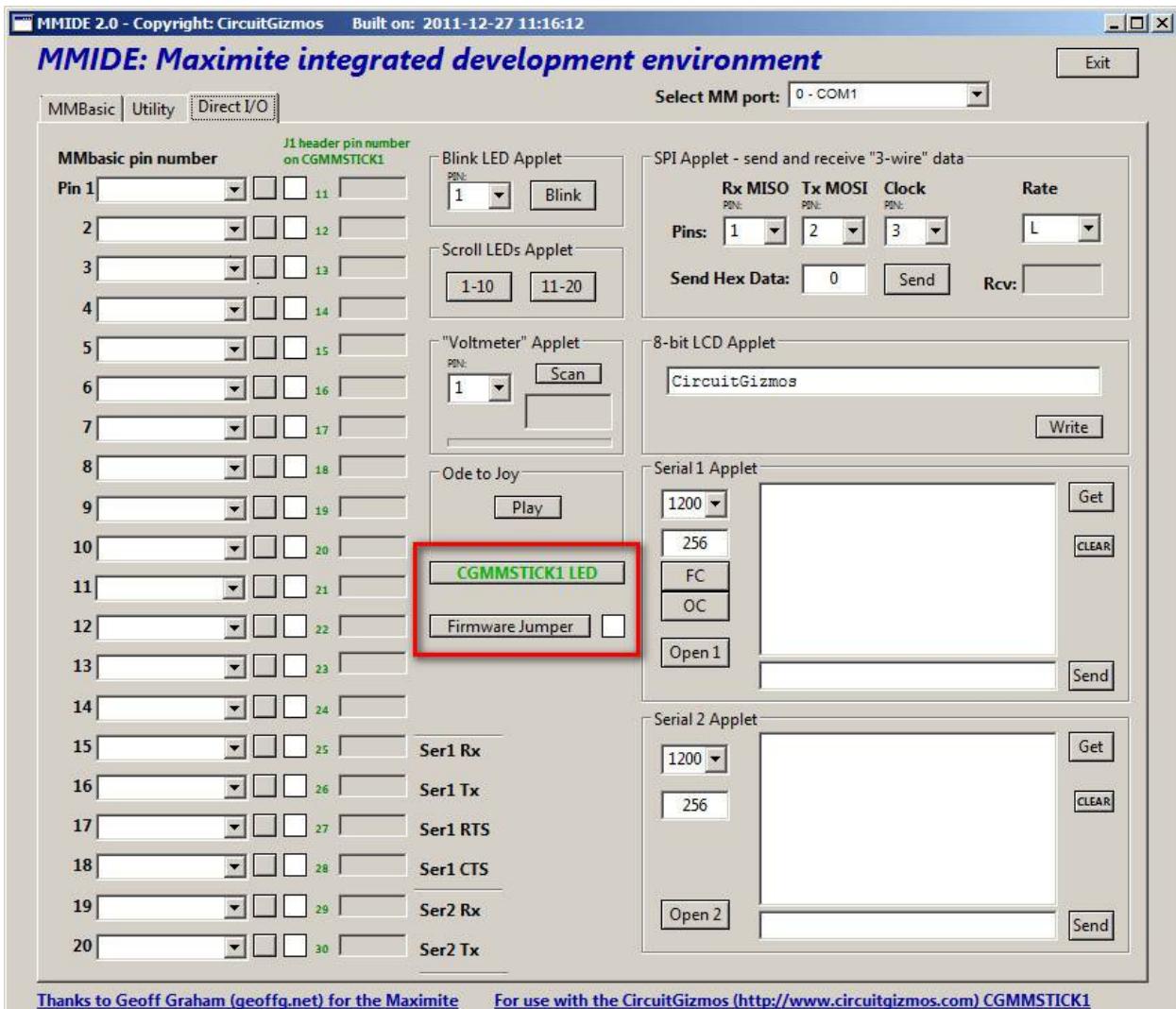


Illustration 34: MMIDE can read the firmware jumper and control the LED.

The MMIDE Direct I/O page has a button (in the highlight above) called “CGMMSTICK LED” that when pressed will toggle the green LED on the CGMMSTICK/CGCOLORMAX.

Also highlighted is a button that reads the state of the firmware jumper. The firmware jumper is J3 on the CGMMSTICK/CGCOLORMAX circuit board. When the jumper is in place, the color of the box next to the “Firmware Jumper” is red. When the jumper is not in place, reading the state by pressing the button will turn that box black.

The remainder of the page is broken up into areas called “Applets.” These can be useful for learning about the functions of the CGMMSTICK without writing any MMBasic code. They can also be used to test hardware before any code is written.

For example the "Ode to Joy" applet will play a short tune out the audio port of the CGMMSTICK/CGCOLORMAX. Just punch the "Play" button and you can verify your audio connection without writing any code.

There are applets for blinking LEDs, for connecting to an LCD, for SPI communication, for reading analog values, and for serial communications.

The applets are described in more detail later in this document.

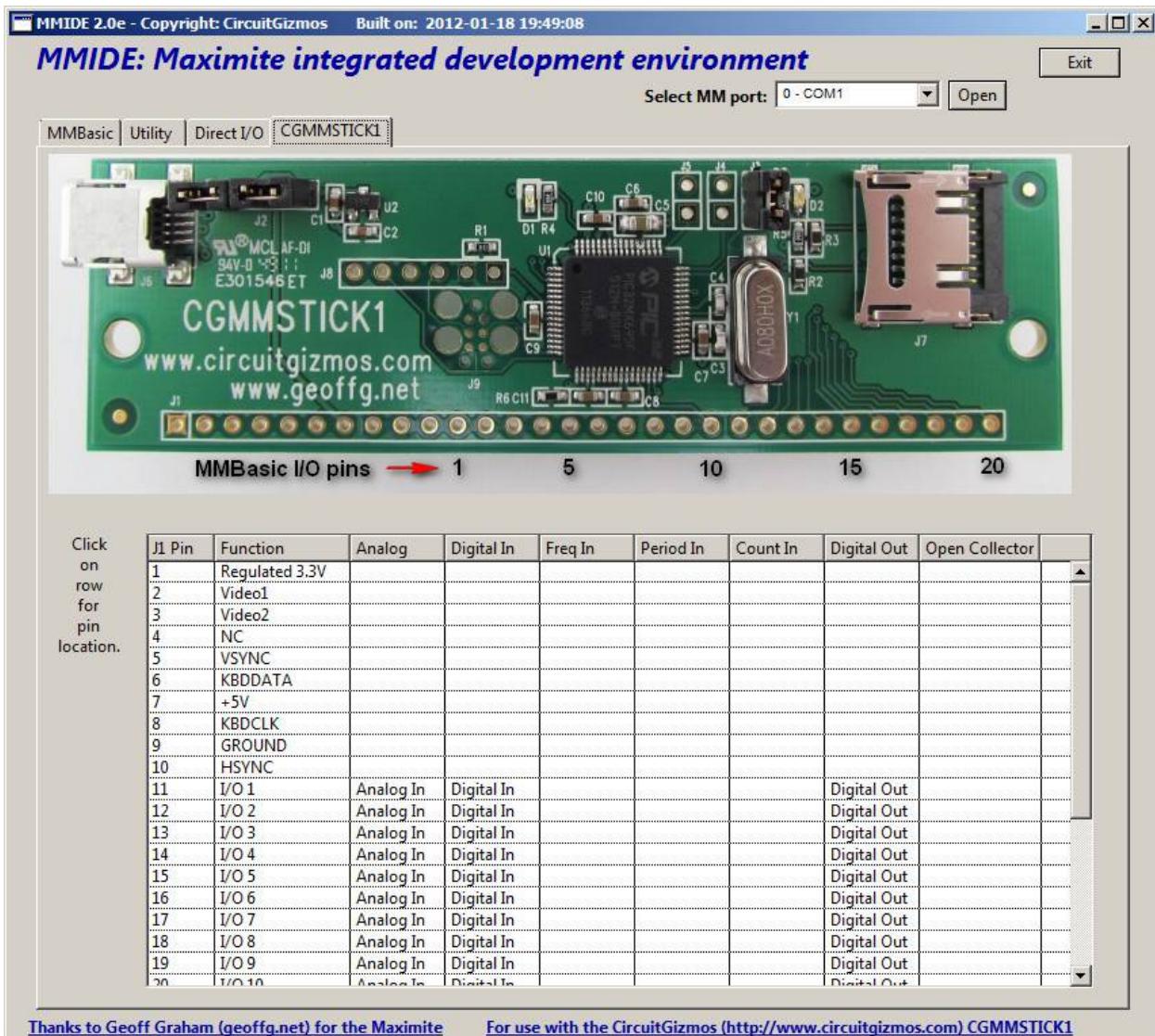


Illustration 35: MMIDE lists the CGMMSTICK I/O lines.

The MMIDE CGMMSTICK page shows an image of the CGMMSTICK to assist in identifying the pin connections to the 30 pins for J1. Clicking on a row in the table highlights the associated pin in the picture.

Maxomite BASIC Features

The CGMMSTICK and CGCOLORMAX have the Maximite BASIC (MMBasic) interpreter programmed into the PIC32 microcontroller. This powerful interpreter runs a version of BASIC based on the BASICs that ran on Home Computers and PCs, but has extensions suitable to interfacing to electronics.

MMBasic provides more than 200 language constructs (BASIC commands and functions). Features of the language like PRINT and GOTO are very familiar to people that program in BASIC.

What do you suppose the following code does?

```
10 Print "Hello World!"
```

Yes, when it is run it prints “Hello World!” to the display. (The display might be MMIDE, a PC terminal program, or a VGA display.)

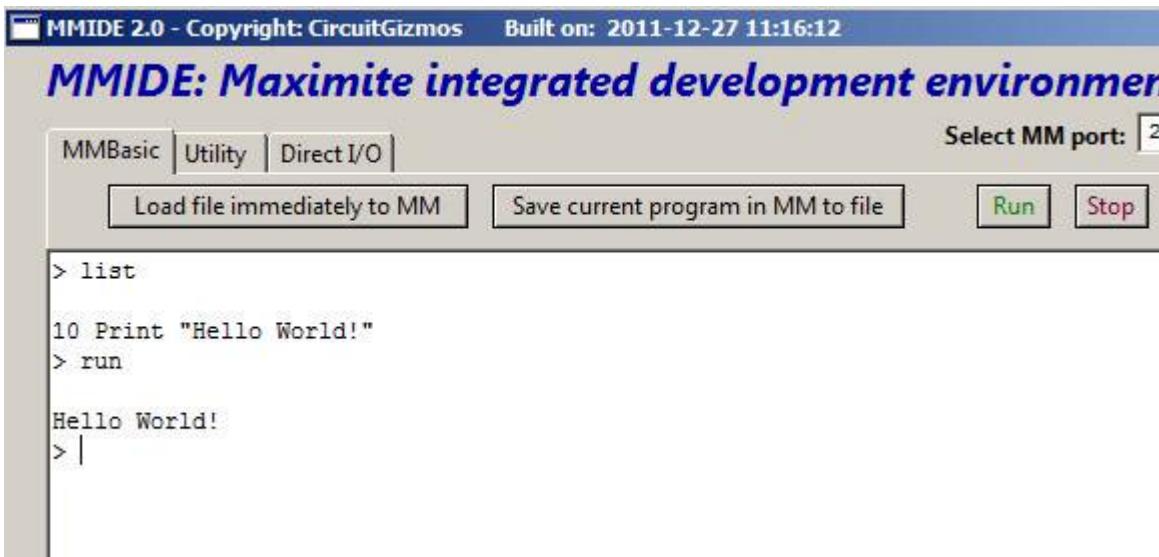


Figure 36: There it is! The "Hello World!" program done on the CGMMSTICK. In this case the display is MMIDE. You could also get the output on a VGA monitor if it is part of your setup.

If you explore the MMBasic section of this document you will notice all of the very familiar BASIC commands such as PRINT, FOR/NEXT, IF/THEN, and others. But it isn't these standard BASIC commands that make MMBasic great for interfacing to electronics, it is the additional powerful commands that deal with the input and output lines that are the real power for the CGMMSTICK and CGCOLORMAX.

These additional commands include SETPIN to control the input and output characteristics of the I/O pin, and PIN() to write to and read from the pin. These general purpose I/O commands have an assortment of digital, analog, and frequency measuring capabilities.

In addition to direct reading and writing, MMBasic has the ability to set some of the port pins to act as standard serial I/O, CAN, PWM, I2C bus interfacing, and SPI bus interfacing.

Project #1

CGMMSTICK LED Control Example

Using MMIDE it is pretty easy to set up a large group of pins to drive LEDs. The "Scroll LEDs Applet" on the Direct I/O page of MMIDE groups all 20 lines into two groups of 10. The groups can be scrolled separately.

Behind the scenes the applet code sets all of the lines to be outputs. It then lights the LEDs one by one from left to right. Then the applet turns the LEDs off one by one from right to left.

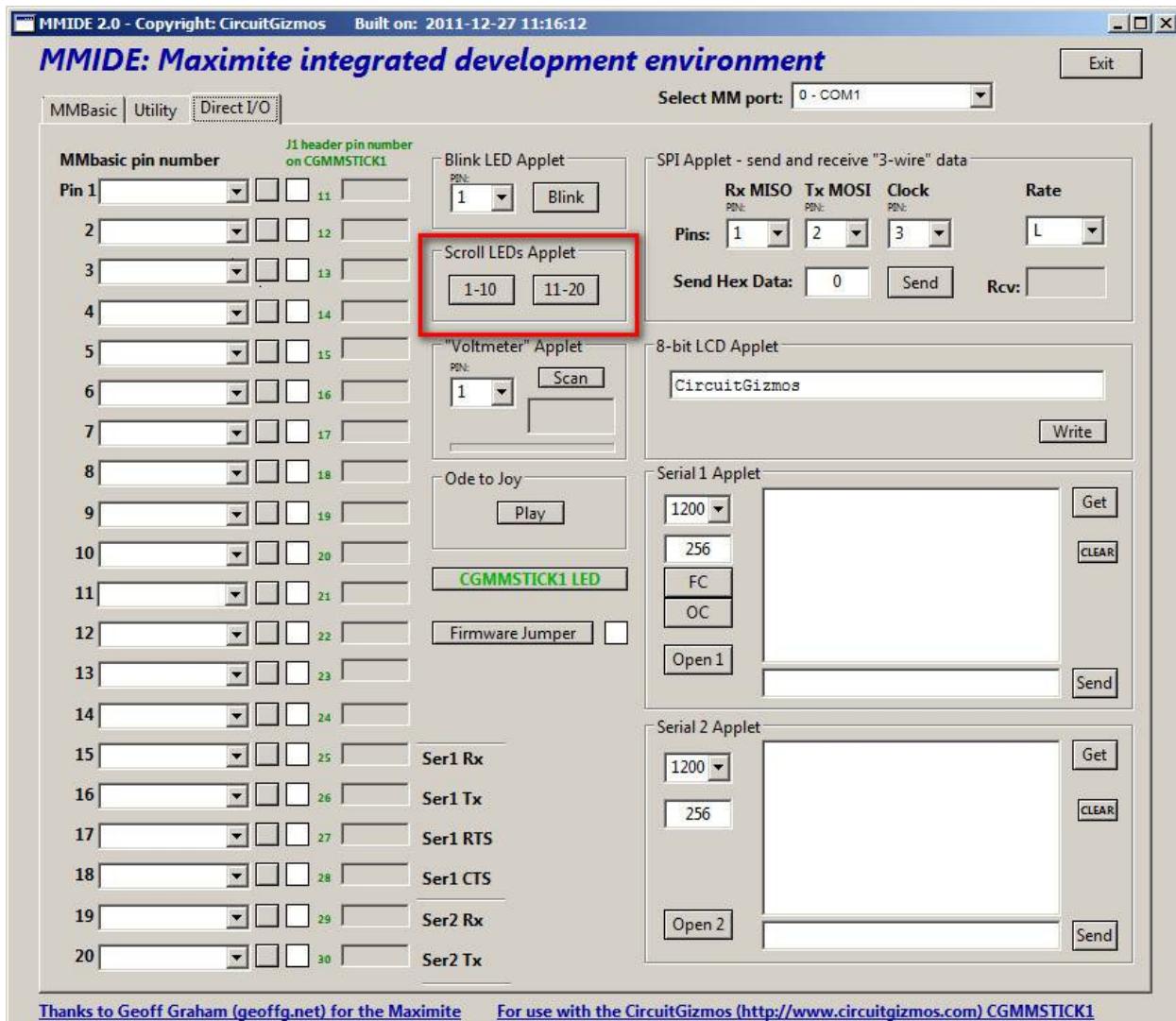


Figure 37: Using the MMIDE Scroll LEDs Applet to test a group of LEDs.

The MMIDE Scroll LEDs Applet lets you choose between two sets of ten pins that will light up when activated.

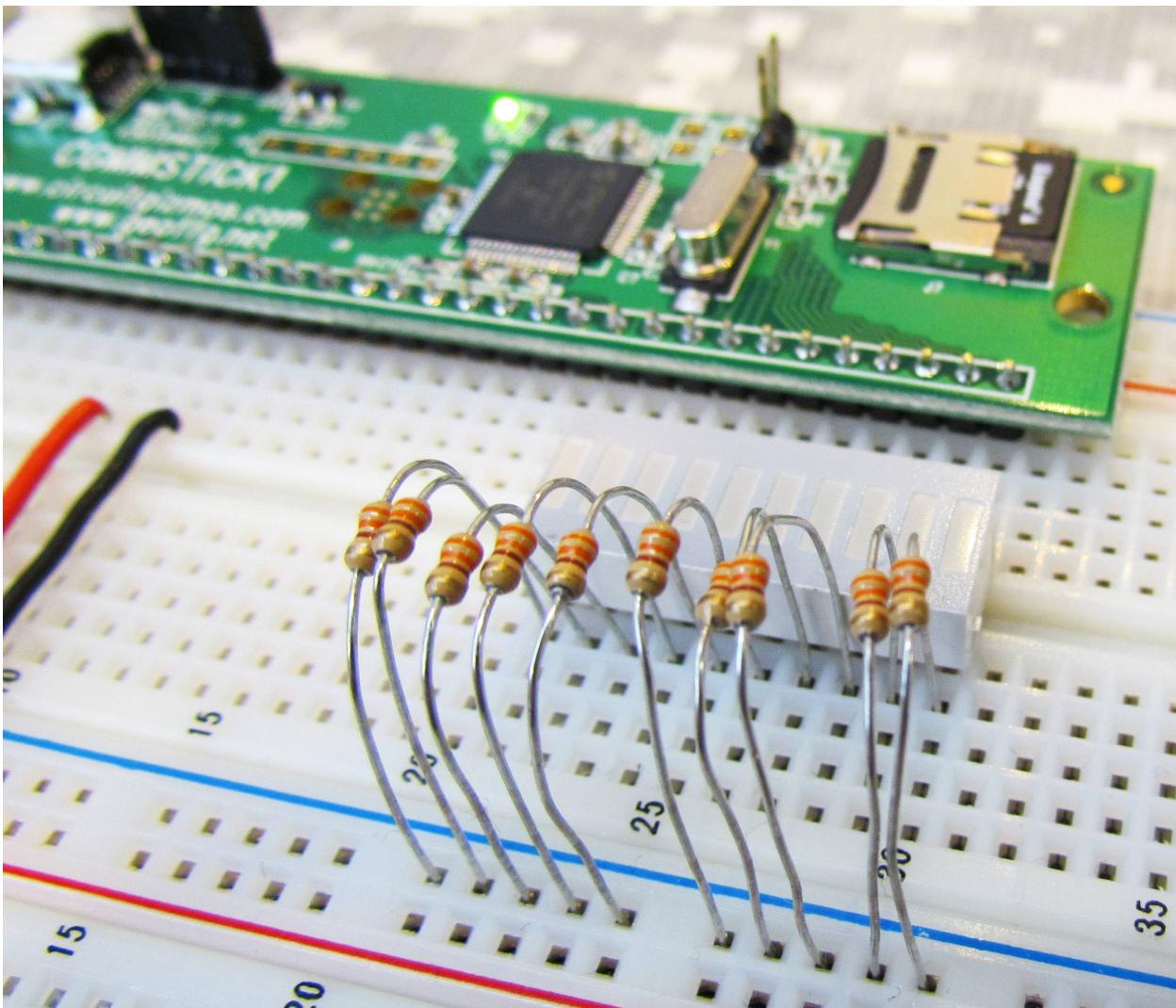


Figure 38: Solderless breadboard setup for demonstration.

The ten LEDs are contained in a single package. Plugged into the solderless breadboard you can see that they connect on one side to pins 21-30 of J1 which means that they are controlled by MMBasic commands as pins 11-20. Current-limiting resistors (330 ohm) tie the LEDs to ground.

Setting those 10 lines to output and then turning each line on and off will turn each LED on and off.

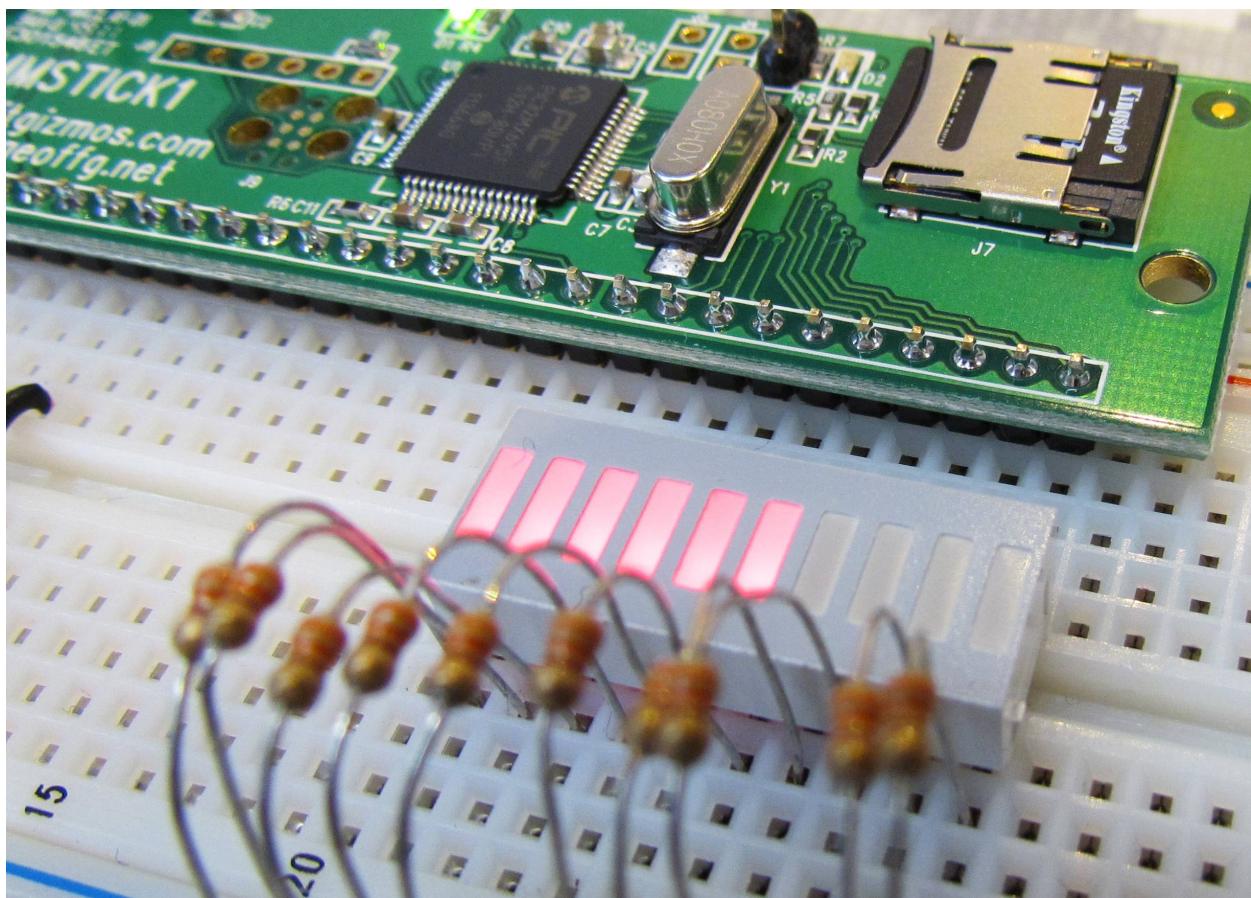


Figure 39: 60% power remaining on warp drive.

The Scroll LEDs Applet will light the LEDs from left to right when the "11-20" button is pressed.

Setting the applet aside, you can write a MMBasic program to control these LEDs, too. This example code will light and extinguish the LEDs randomly.

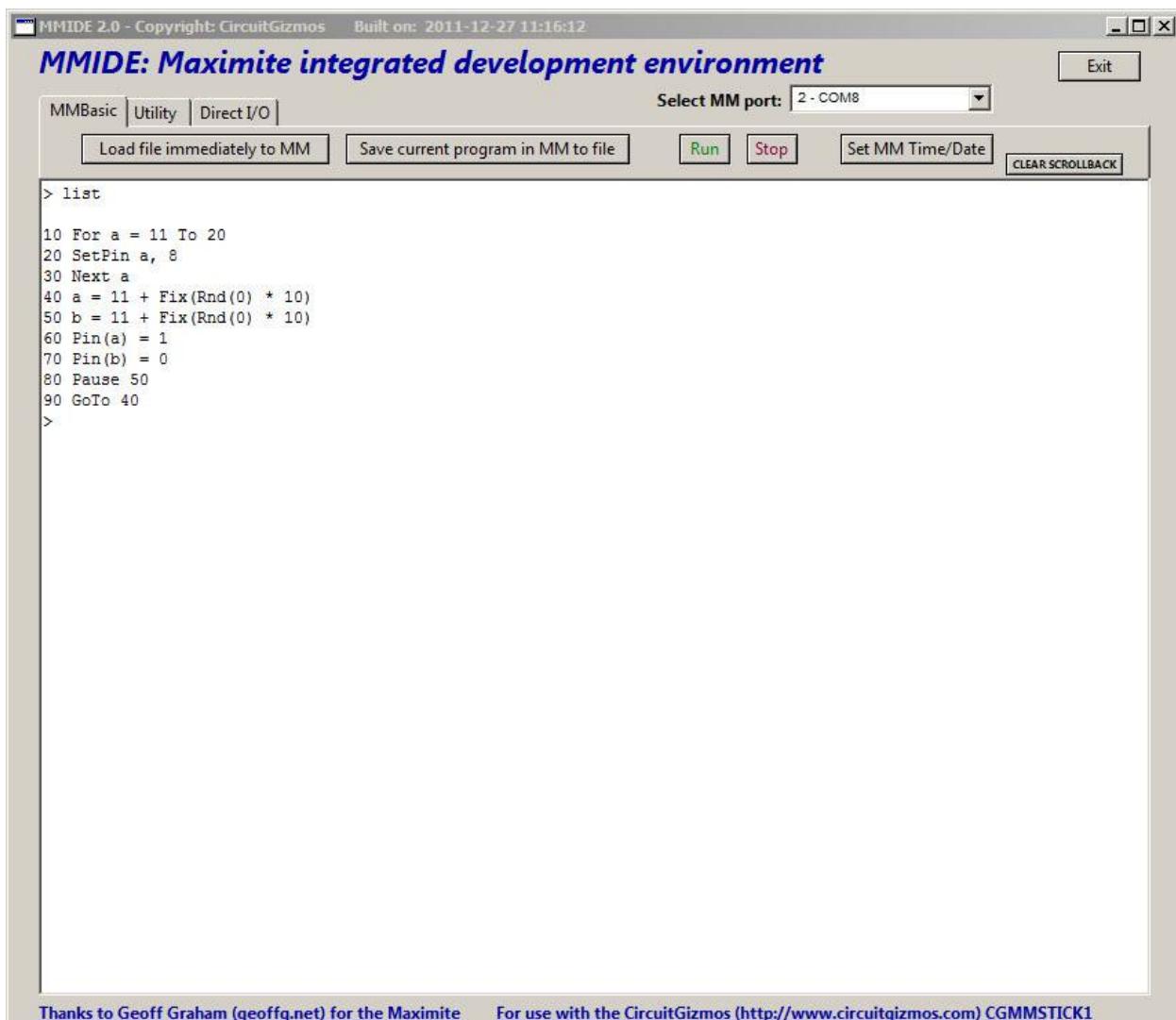


Figure 40: Using MMIDE to enter some code to control the LEDs

Code can be entered to the CGMMSTICK using the MMBasic page of MMIDE. The following code will flash the LEDs randomly.

```
10 For a = 11 To 20
20 SetPin a, 8
30 Next a
40 a = 11 + Fix(Rnd(0) * 10)
50 b = 11 + Fix(Rnd(0) * 10)
60 Pin(a) = 1
70 Pin(b) = 0
80 Pause 50
90 Goto 40
```

MMBasic lines 10, 20, and 30 loop to set the I/O pins 11-20 to output.

MMBasic line 40 establishes a random pin to turn on. Rnd(0) generates a random value between 0 and 1 that is multiplied by 10 to get a random number between 1 and 10. "Fix" cleans up the number by removing the fraction. The number is offset by 11 to finally be a random number between 11 and 20 for pins 11-20.

MMBasic line 50 does the same thing for a random line to turn off.

60 and 70 turn off and on these lines respectively. MMBasic line 80 pauses for a twentieth of a second before line 90 forces the whole thing to repeat.

When run, the program turns the 10 LEDs on and off randomly.

The code can also be done without line numbers:

```
' Set pins to output
For a = 11 To 20
    SetPin a, 8
Next a

Do
    ' Select two random pins
    a = 11 + Fix(Rnd(0) * 10)
    b = 11 + Fix(Rnd(0) * 10)

    ' Turn one on, one off
    Pin(a) = 1
    Pin(b) = 0

    ' Pause
    Pause 50
Loop
```

Project #2

CGCOLORMAX Relay Shield Example

For this project a CGCOLORMAX was paired up with a relay shield. By using relays, you can control high voltages and currents. The four relays on this board are rated to handle up to 2 amps at up to 35 Volts DC.

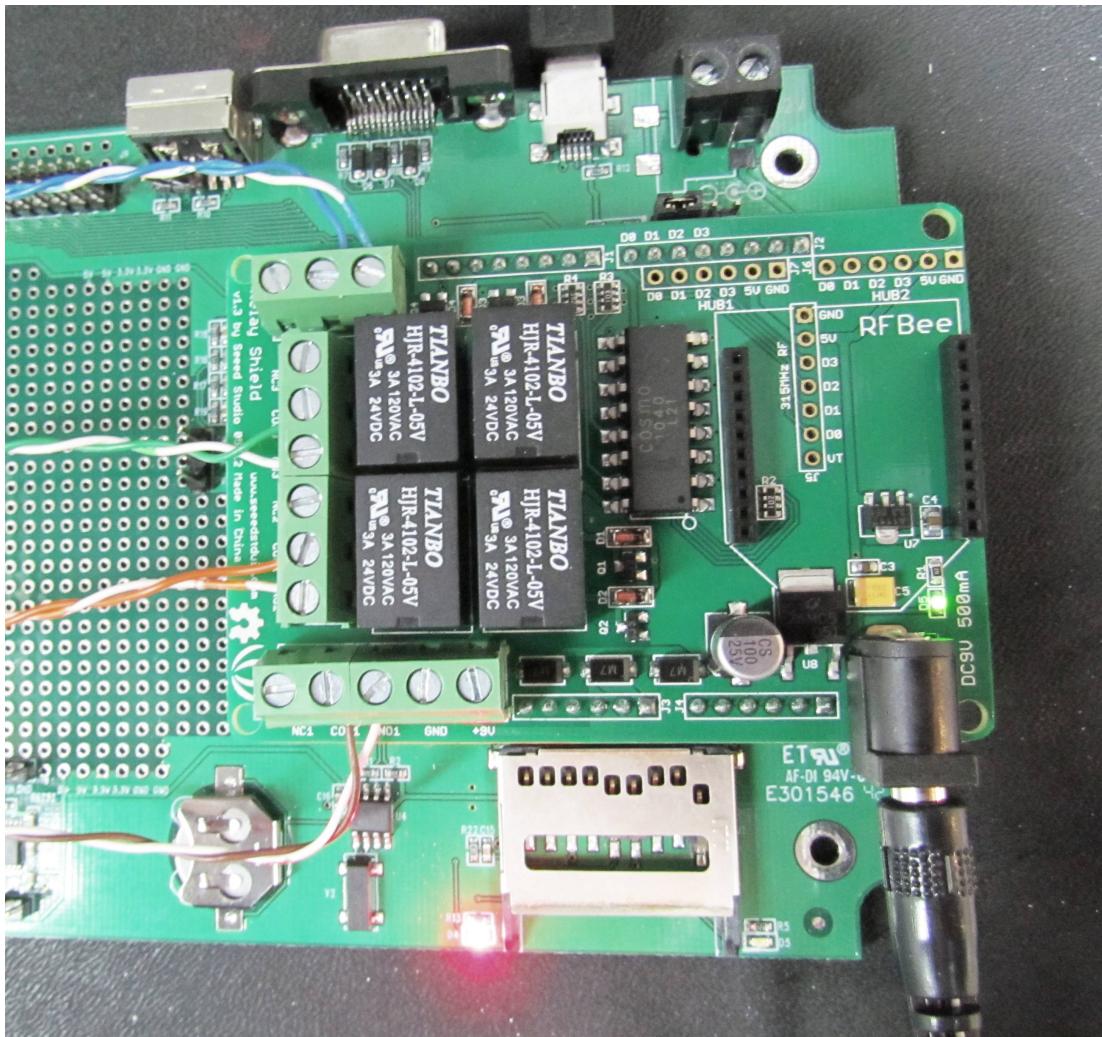


Illustration 41: A CGCOLORMAX1 board with a relay shield installed. The newer revision (the CGCOLORMAX2) accepts shields in the same way.

You can turn on/off lights, motors, and even larger relays by using MMBasic to control the relays on a relay shield. Connections to COM and N.O. are like a switch that closes when the relay is energized.

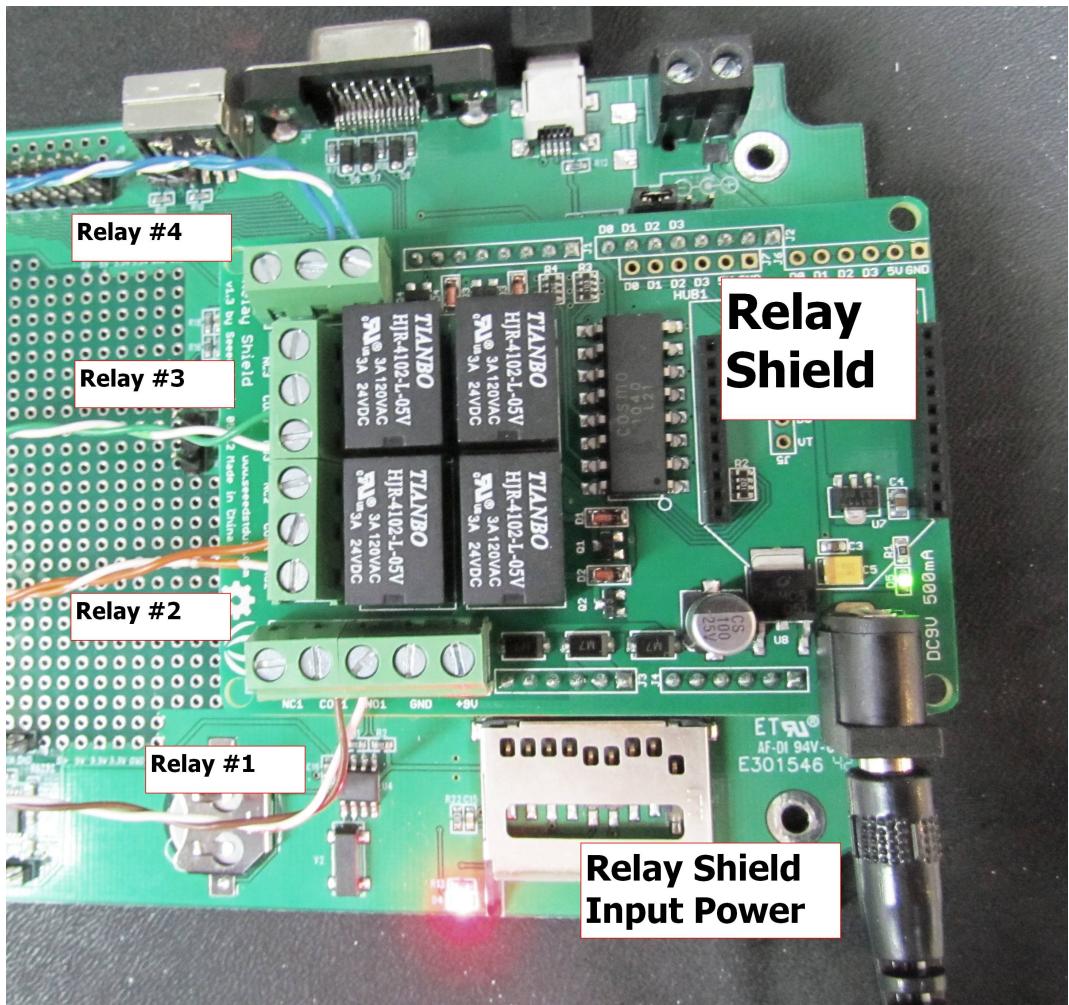


Illustration 42: Important parts of the relay shield annotated. Four sets of wires are connected to COM/N.O. and run off to the left to turn on some lights (not in picture).

The relay shield needs to be powered by an external DC supply of 7-12V DC through either the screw terminals or the 5.5/2.1 mm barrel jack. The external supply is regulated to 5V to power the relays.

The four relays are "Form C" which means that they each have a common connection that connects to the N.C. (normally closed) terminal when unpowered, and then to the N.O. (normally open) terminal when the relay is powered on.

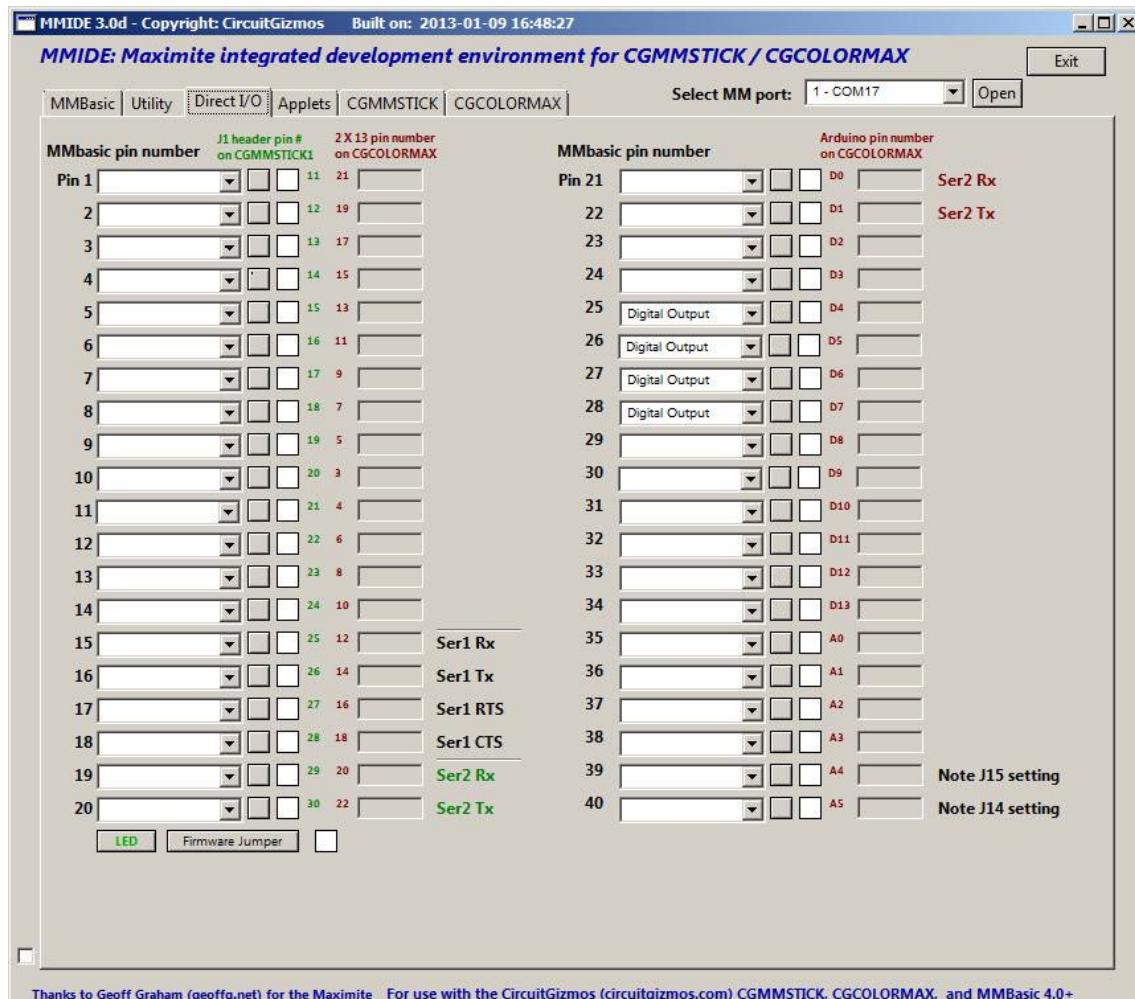


Illustration 43: The Direct I/O tab on MMIDE 3 can directly control the I/O of the shield connector. Here the appropriate lines are set to output.

Using MMIDE 3 it is pretty easy to set up the pins that are needed to operate the relays. D7 on the shield is relay #1. D6, D5, and D4 are relay #2, #3, and #4 respectively. These need to be set to output to control the relays.

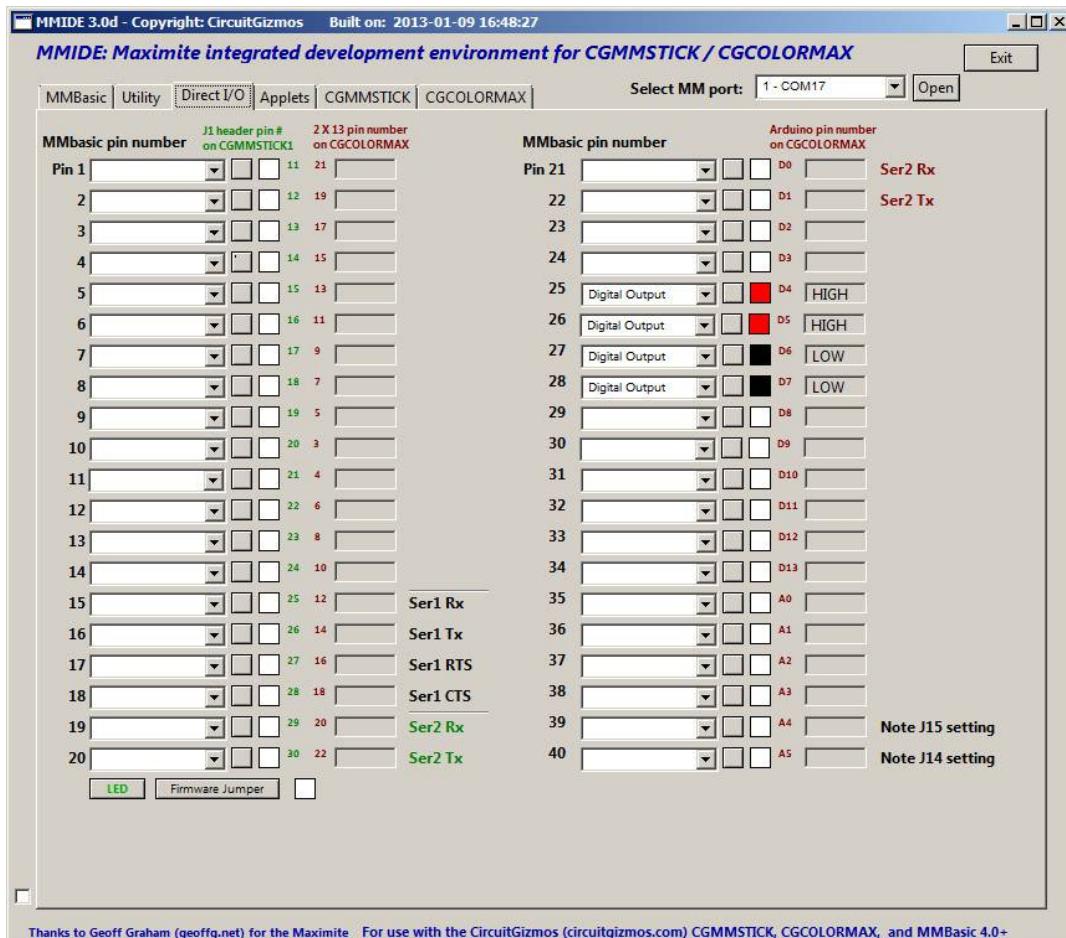


Illustration 44: MMIDE3 is again used for this relay control example. The on/off buttons associated with the lines that control the relay can be clicked to turn the relays on and off.

Relay #1 can be turned on/off in MMBasic by setting Pin 28 (same as D7 on the shield) to either 1 or 0.

```
' Set relay #1 line to output
SetPin 28, 8

' Turn relay #1 on
Pin(28) = 1

' Turn relay #1 off
Pin(28) = 0

' Set relay #2 line to output
SetPin 27, 8
```

```
' Turn relay #2 on
Pin(27) = 1

' Turn relay #2 off
Pin(27) = 0
```

If you happen to be powering the CGCOLORMAX with 8 to 12V DC, the same supply can also be wired to the relay shield to power it. Make sure that the power supply can provide the extra current needed to operate the relay shield.

Project #3

CGMMSTICK Sound Example

The Maximite can make sound using the 'SOUND' command. The SOUND command accepts a frequency to play for the sound and a duration to play that sound.

MMIDE has an Applet that plays Ode to Joy.

On the CGMMSTICK the audio connection is NOT on J1 along with all of the port lines. Instead it is at the top of the board on an unpopulated header named J5.

On J5 the top connection is ground, the bottom (square) hole is the audio line.

If you have a small amplifier, you can connect the amp to these two lines and run the Ode to Joy Applet that is in MMIDE.

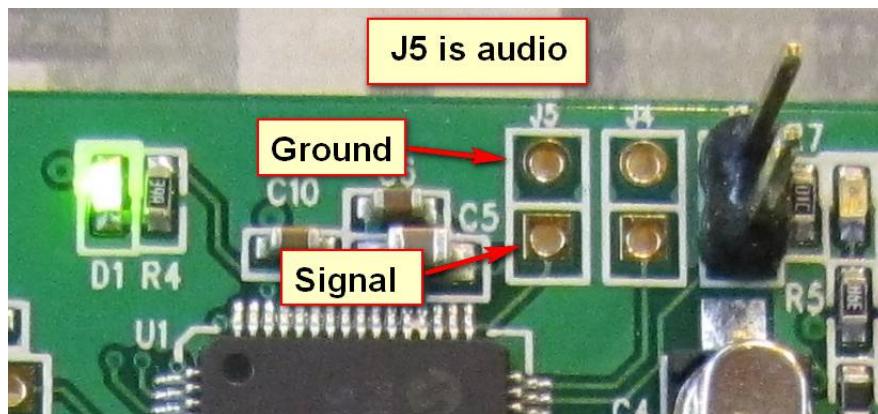


Figure 45: CGMMSTICK1 audio connection.

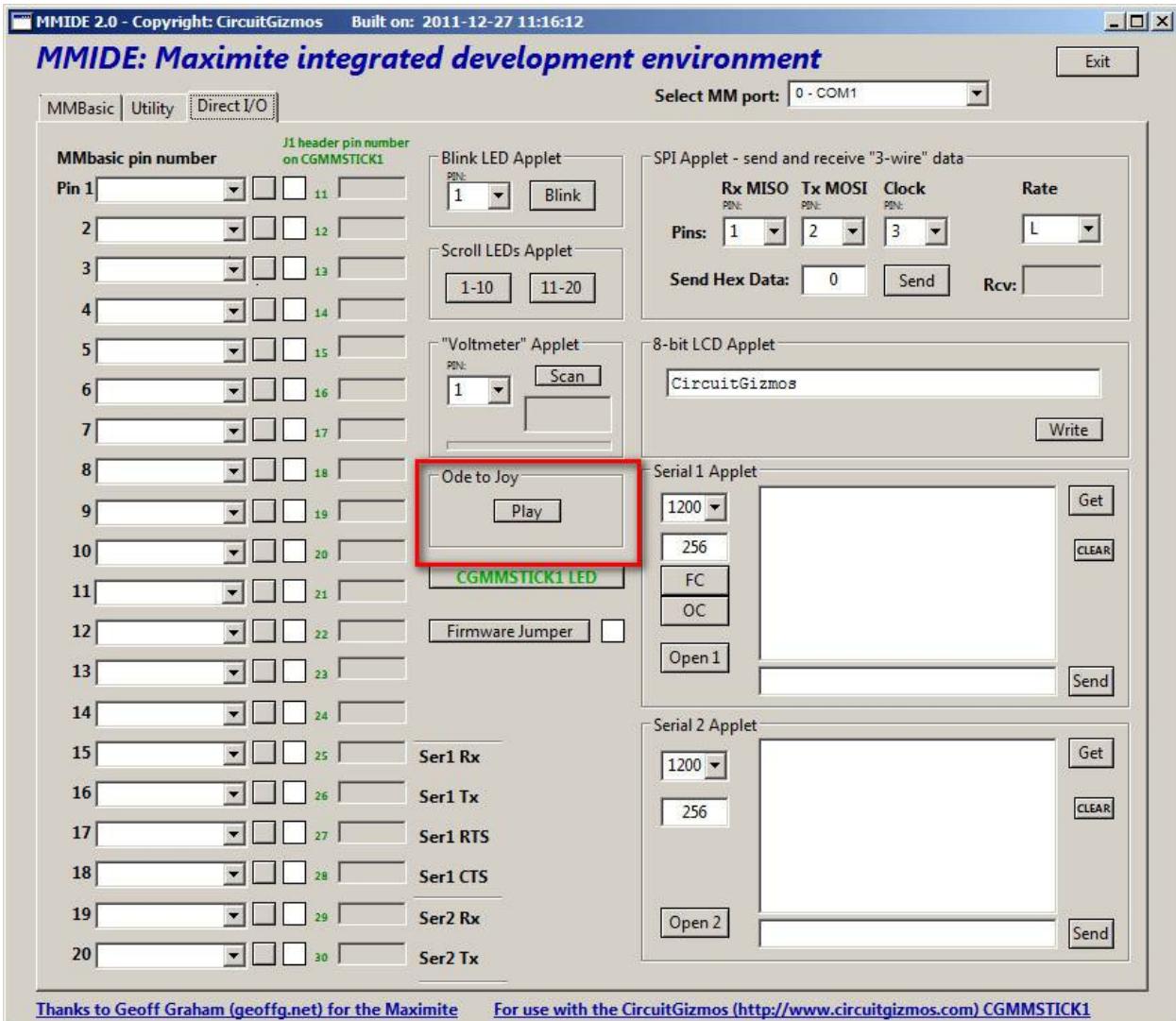


Figure 46: Using the Ode to Joy Applet in MMIDE to play a tune.

The Ode to Joy Applet downloads a MMBasic program to the CGMMSTICK and then runs that program. Below is the program that is sent. You might notice that there is a test for data where the frequency could be 0. If the frequency is 0 no note is played, just a delay.

I asked my 15 year old son to write this program. He used a frequency/note chart to get the frequencies right, but the notes for Ode to Joy were in his head – he didn't have to look them up.

His frequency table was a little bit different than the table that I reproduced after the program below.

```
10 FOR a=1 TO 65
20 READ fr,du
30 IF fr=0 THEN
40 PAUSE du
50 ELSE
60 SOUND fr,du
70 ENDIF
80 PAUSE du
90 PAUSE 30
100 NEXT a
110 DATA 493.88,300,493.88,300
120 DATA 523.25,300,587.33,300
130 DATA 587.33,300,523.25,300
140 DATA 493.88,300,440.00,300
150 DATA 392.00,300,392.00,300
160 DATA 440.00,300,493.88,300
170 DATA 493.88,450,440.00,200
180 DATA 440.00,300
190 DATA 0,200
200 DATA 493.88,300,493.88,300
210 DATA 523.25,300,587.33,300
220 DATA 587.33,300,523.25,300
230 DATA 493.88,300,440.00,300
240 DATA 392.00,300,392.00,300
250 DATA 440.00,300,493.88,300
260 DATA 440.00,450,392.00,200
270 DATA 392.00,300
280 DATA 0,200
290 DATA 440.00,300,440.00,300
300 DATA 493.88,300,392.00,300
310 DATA 440.00,300,493.88,150
320 DATA 523.25,150,493.88,300
330 DATA 392.00,300,440.00,300
340 DATA 493.88,150,523.25,150
350 DATA 493.88,300,440.00,300
```

```
360 DATA 392.00,300,440.00,300
370 DATA 293.66,550
380 DATA 0,200
390 DATA 493.88,300,493.88,300
400 DATA 523.25,300,587.33,300
410 DATA 587.33,300,523.25,300
420 DATA 493.88,300,440.00,300
430 DATA 392.00,300,392.00,300
440 DATA 440.00,300,493.88,300
450 DATA 440.00,450,392.00,200
460 DATA 392.00,300
```

Note/Frequency Table for SOUND command

Note	Frequency
C	16.4
C sharp / D flat	17.3
D	18.4
D sharp / E flat	19.5
E	20.6
F	21.8
F sharp / G flat	23.1
G	24.5
G sharp / A flat	26
A	27.5
A sharp / B flat	29.1
B	30.9
C	32.7
C sharp / D flat	34.7
D	36.7
D sharp / E flat	38.9
E	41.2
F	43.7
F sharp / G flat	46.3
G	49
G sharp / A flat	51.9
A	55
A sharp / B flat	58.3
B	61.7
C	65.4
C sharp / D flat	69.3
D	73.4
D sharp / E flat	77.8
E	82.4
F	87.3
F sharp / G flat	92.5
G	98
G sharp / A flat	104
A	110

Note	Frequency
A sharp / B flat	117
B	123
C	131
C sharp / D flat	139
D	147
D sharp / E flat	156
E	165
F	175
F sharp / G flat	185
G	196
G sharp / A flat	208
A	220
A sharp / B flat	233
B	247
C	262
C sharp / D flat	277
D	294
D sharp / E flat	311
E	330
F	349
F sharp / G flat	370
G	392
G sharp / A flat	415
A	440
A sharp / B flat	466
B	494
C	523
C sharp / D flat	554
D	587
D sharp / E flat	622
E	659
F	698
F sharp / G flat	740
G	784
G sharp / A flat	831
A	880

Note	Frequency
A sharp / B flat	932
B	988
C	1047
C sharp / D flat	1109
D	1175
D sharp / E flat	1245
E	1319
F	1397
F sharp / G flat	1480
G	1568
G sharp / A flat	1661
A	1760
A sharp / B flat	1865
B	1976
C	2093
C sharp / D flat	2217
D	2349
D sharp / E flat	2489
E	2637
F	2794
F sharp / G flat	2960
G	3136
G sharp / A flat	3322
A	3520
A sharp / B flat	3729
B	3951

Project #4

CGMMSTICK Input Example and Analog Input Example

Simple input is pretty simple to test using MMIDE. All of the 20 pins of the CGMMSTICK can be set to be digital inputs. To test a simple button input, just use this testing facility.

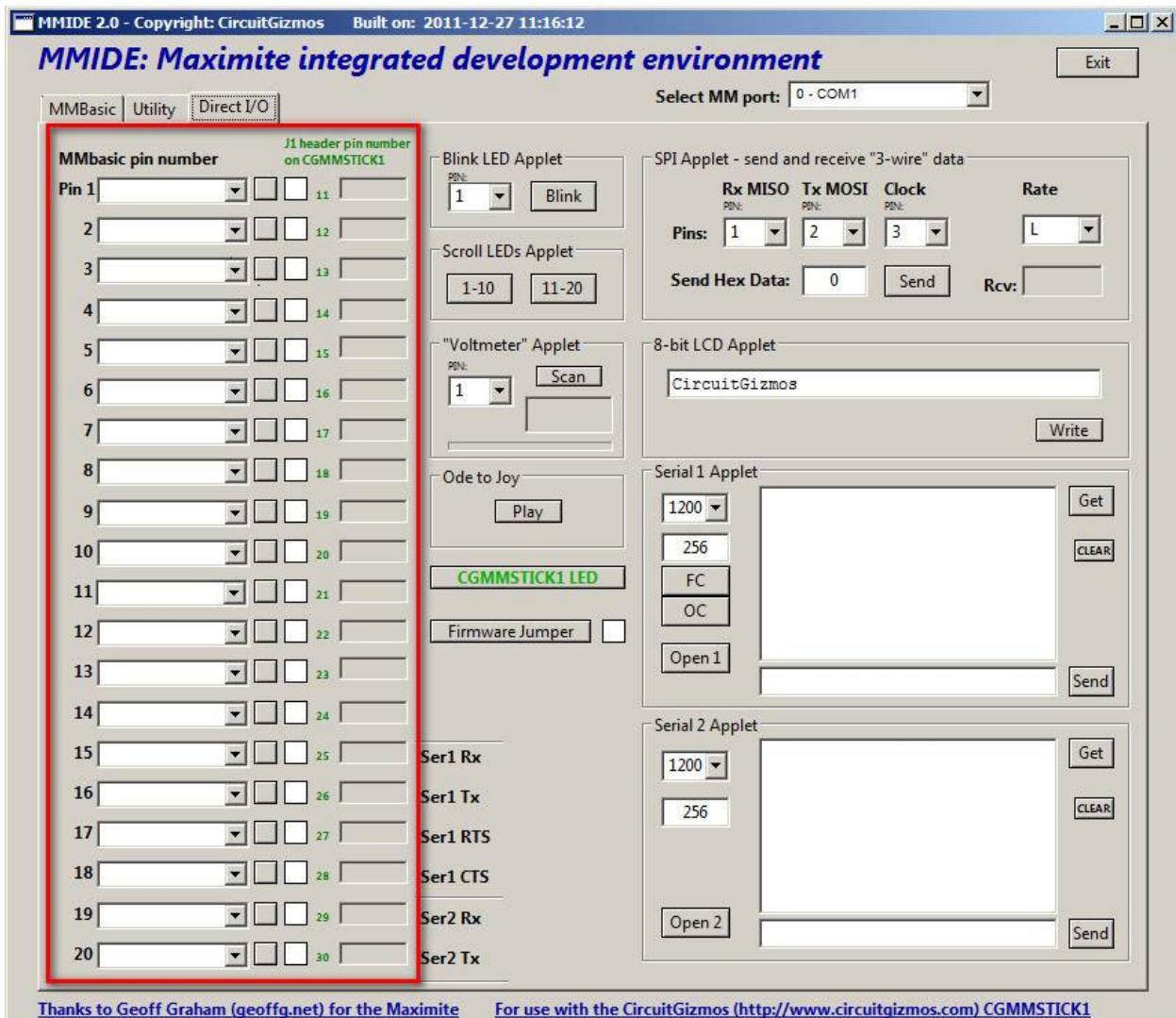


Figure 47: All 20 of the CGMMSTICK1 lines can be controlled for testing.

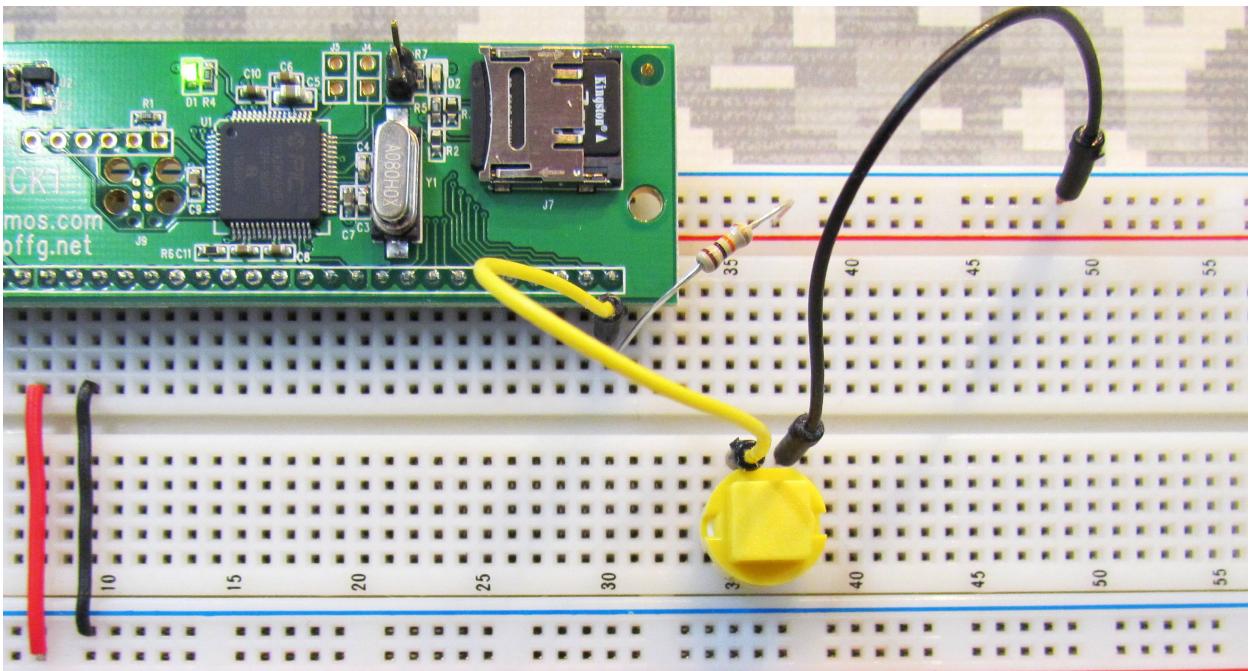


Figure 48: CGMMSTICK1 button input.

A simple button input for the CGMMSTICK would use a 10 kilo ohm resistor to pull the input line (in this case MMBasic pin 20) high, then the button press would short the line low.



Figure 49: Testing the input.

With the right pin set to “Digital Input” both states of the button can be tested using MMIDE to do the work. In the picture above the red square and value of 1 indicate that the button is not pushed – the resistor is pulling the input line high.

Ten pins (Pin 1-10 from the perspective of MMBasic) can be used to read analog values from 0V up to a maximum of 3.3V.

MMIDE provides a voltmeter applet that can be set to any of those ten pins and will repeatedly measure the voltage on the selected pin for display as both a small bar graph and numerically.

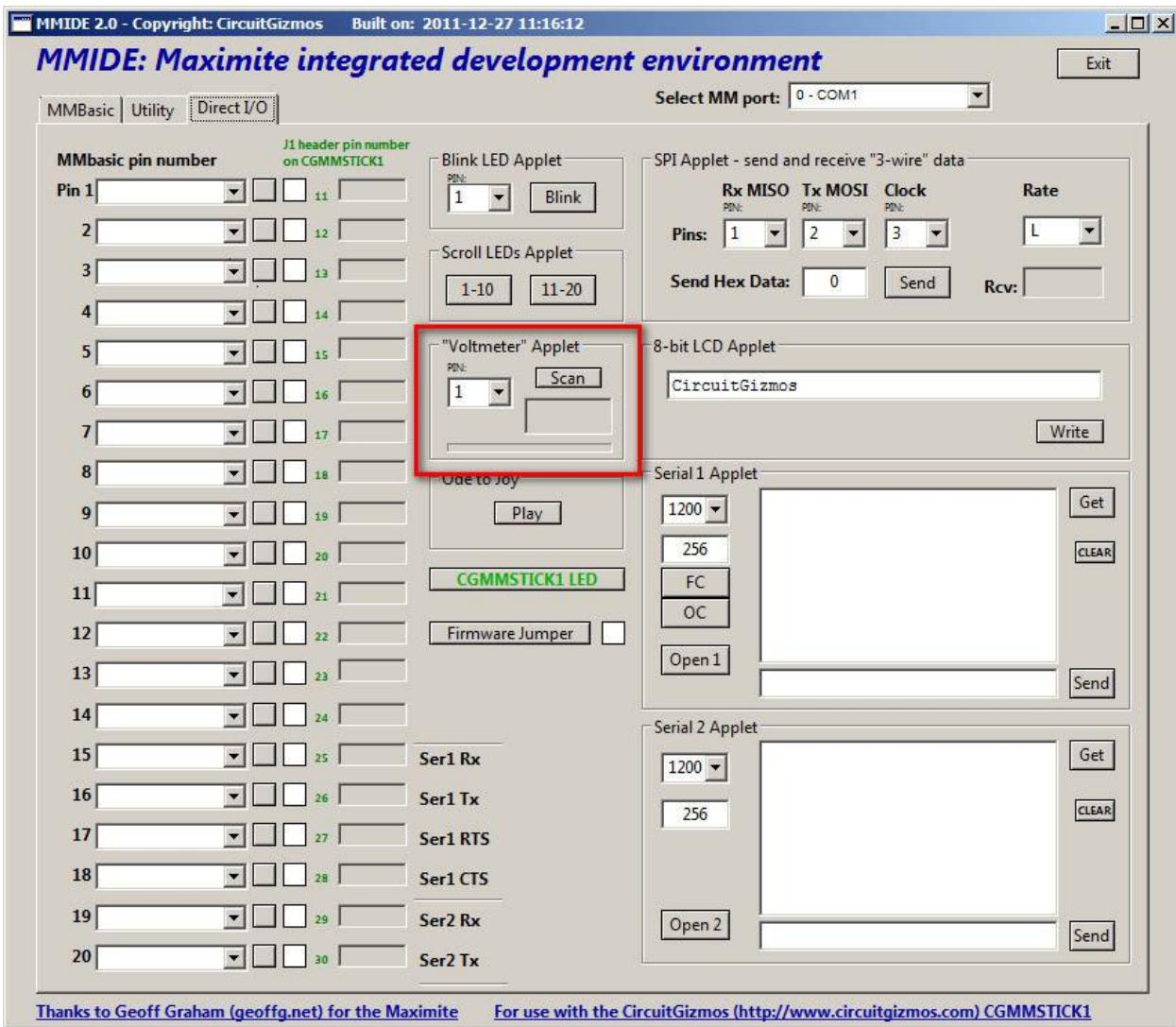


Illustration 50: MMIDE "voltmeter" application.

Project #5

CGMMSTICK LCD Interface Example

The Direct I/O page of MMIDE can also be used to test out an LCD display. In this example an LCD display is interfaced to the first ten (Pins 1-10 from MMBasic's perspective) and clicking the "Write" button in the 8-bit LCD Applet will write the text string that is in the text box above the button.

The LCD is wired as follows:

Pin 1 (11 on the connector) connects to the LCD data/control line.

Pin 2 connects to the LCD enable line.

The LCS read/write line is connected to ground. Ground and 5V power the LCD.

Pins 3-10 connect to the LCD D0-D7 data lines.

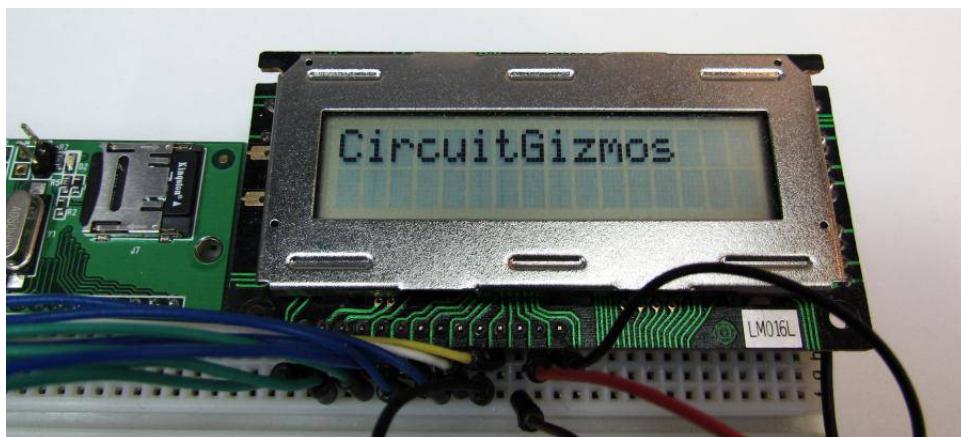


Figure 51: "CircuitGizmos" written to the LCD display

Prior to writing the text to the LCD, the applet writes the command sequence of &h33, &h33, &h38, &h0C, &h01. The LCD data sheet will help you understand the commands.

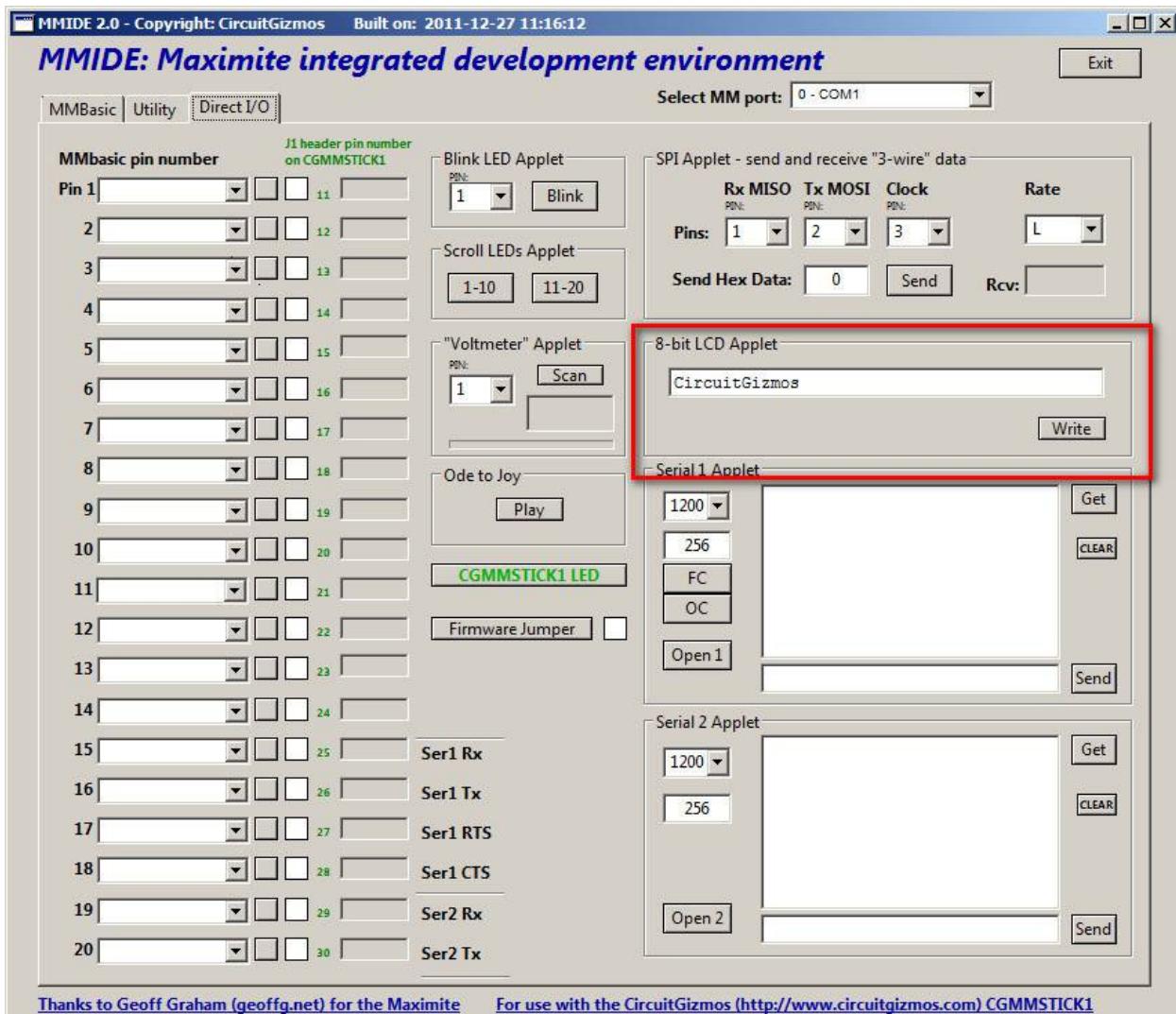


Illustration 52: MMIDE LCD control applet.

Project #6

CGMMSTICK Vacuum Fluorescent Display Example

I've had a couple of these display modules sitting around for a while. They have a 10-pin connector, not the 'normal' 14 that an LCD module has. I also don't have a data sheet for the display, but a couple of old data sheets for different but somewhat similar displays. Every time I bump into the modules in my box of stuff I wonder what they look like when activated.

It is pretty easy to use MMBasic to just try these modules, even if I'm not confident that the data that I have on the modules is correct. I'll give it a good try with the limited information that I have.

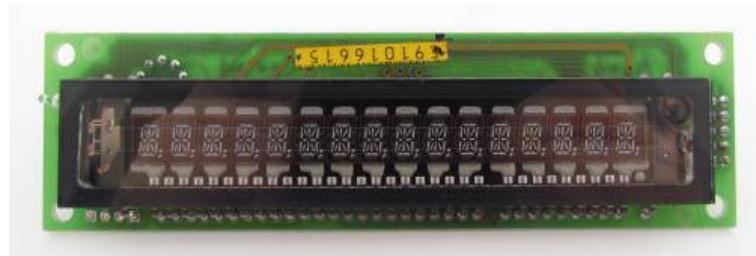


Figure 53: Vacuum Fluorescent Display (VFD) Module

I decided to wire up the display to a CGMMSTICK to test them. It is a 5V display, so I'll use OC outputs pulled to 5V through a 10k resistor. The display has a reset line, data in (serially transmitted, synched with the clock line), and clock. Data is most significant bit first.

According to the info I had:

Pins 2, 4, 6, 8, 10 are ground

Pin 1 +5V

Pin 3 clock

Pin 5 data

Pin 7 reset

I powered the module before connecting to the CGMMSTICK. Nothing appeared on the display, but I didn't get a cloud of smoke either.

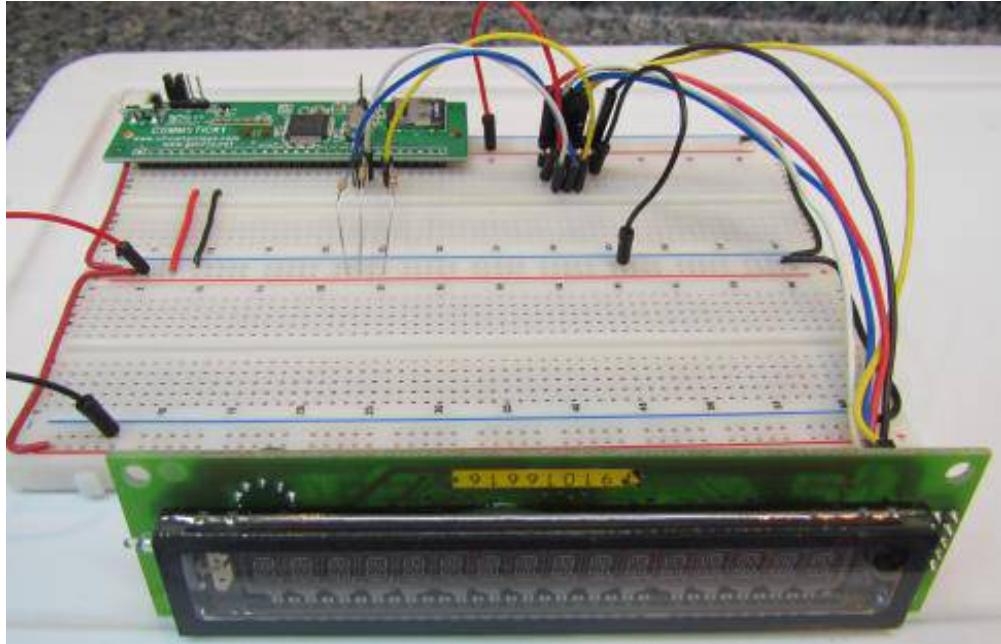


Figure 54: VFD Test Circuit

The data sheet for a similar module says that the value that it takes to write "A" is 1, "B" is 2. This doesn't follow ASCII.

Below is the code I used to test the display:

```
' VFD test code

' I/O init
' Open collector data
SETPIN 12, 9 : PIN(12) = 1

' Open collector clock
SETPIN 13, 9 : PIN(13) = 0

' Open collector reset
SETPIN 15, 9 : PIN(15) = 1

' Values for characters
```

```
_A = 1 : _B = 2 : _C = 3 : _D = 4 : _E = 5  
_F = 6 : _G = 7 : _H = 8 : _I = 9 : _J = 10  
_K = 11 : _L = 12 : _M = 13 : _N = 14 : _O = 15  
_P = 16 : _Q = 17 : _R = 18 : _S = 19 : _T = 20  
_U = 21 : _V = 22 : _W = 23 : _X = 24 : _Y = 25 : _Z = 26
```

```
_AT = 0 : _SPACE = 32
```

```
_ZERO = 48 : _ONE = 49 : _TWO = 50 : _THREE = 51  
_FOUR = 52 : _FIVE = 53 : _SIX = 54 : _SEVEN = 55  
_EIGHT = 56 : _NINE = 57
```

```
' Initialize and display message
```

```
VFDRESET
```

```
VFDDATA _SPACE  
VFDDATA _C  
VFDDATA _I  
VFDDATA _R  
VFDDATA _C  
VFDDATA _U  
VFDDATA _I  
VFDDATA _T  
VFDDATA _SPACE  
VFDDATA _G  
VFDDATA _I  
VFDDATA _Z  
VFDDATA _M  
VFDDATA _O  
VFDDATA _S  
VFDDATA _SPACE
```

```
END
```

```
' VFD display reset routine
```

```
Sub VFDRESET
```

```

' pulse the reset line
PIN(15) = 1 : PAUSE 1
PIN(15) = 0 : PAUSE 1
PIN(15) = 1 : PAUSE 1

' Clear the display
VFDDATA &h20
VFDDATA &hFF

End Sub

' Write data to display, MSB first
Sub VFDDATA (valu)

PIN(13) = 1

PIN(12) = valu AND &B1000000
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B0100000
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B0010000

```

```

PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B00010000
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B00001000
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B00000100
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B00000010
PIN(13) = 0 : PIN(13) = 1

PIN(12) = valu AND &B00000001
PIN(13) = 0 : PIN(13) = 1

```

End Sub

And the result:

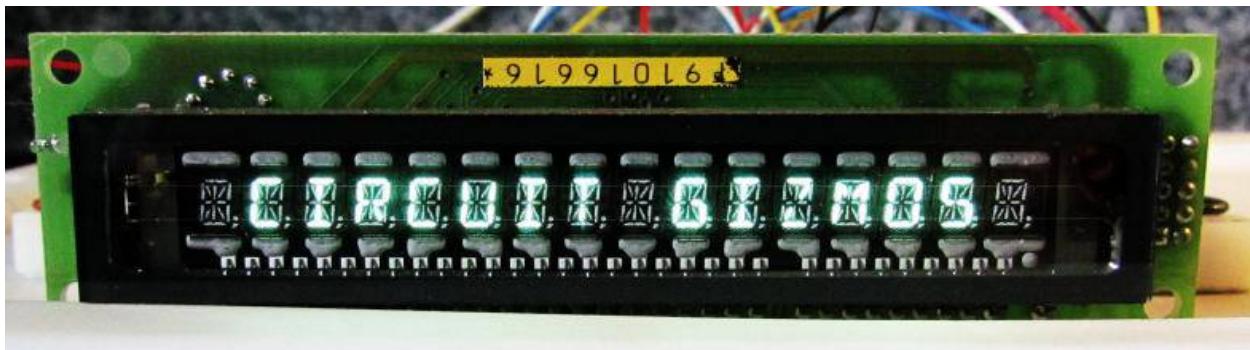


Figure 55: "CIRCUIT GIZMOS" written to the VFD. Looks better to the eye than to the camera.

Project #7

CGCOLORMAX1 Serial I/O Example

Two serial ports are available on the CGMMSTICK and CGCOLORMAX for asynchronous serial communications.

MMIDE can again be used to demonstrate serial communication.

This example uses the CGCOLORMAX1, since it has on-board RS232 circuitry.

The CGCOLORMAX2 has optional RS232 circuitry. The same functionality is possible with the RS232 line driver chip added, but the connector/pin names and numbers will be different.

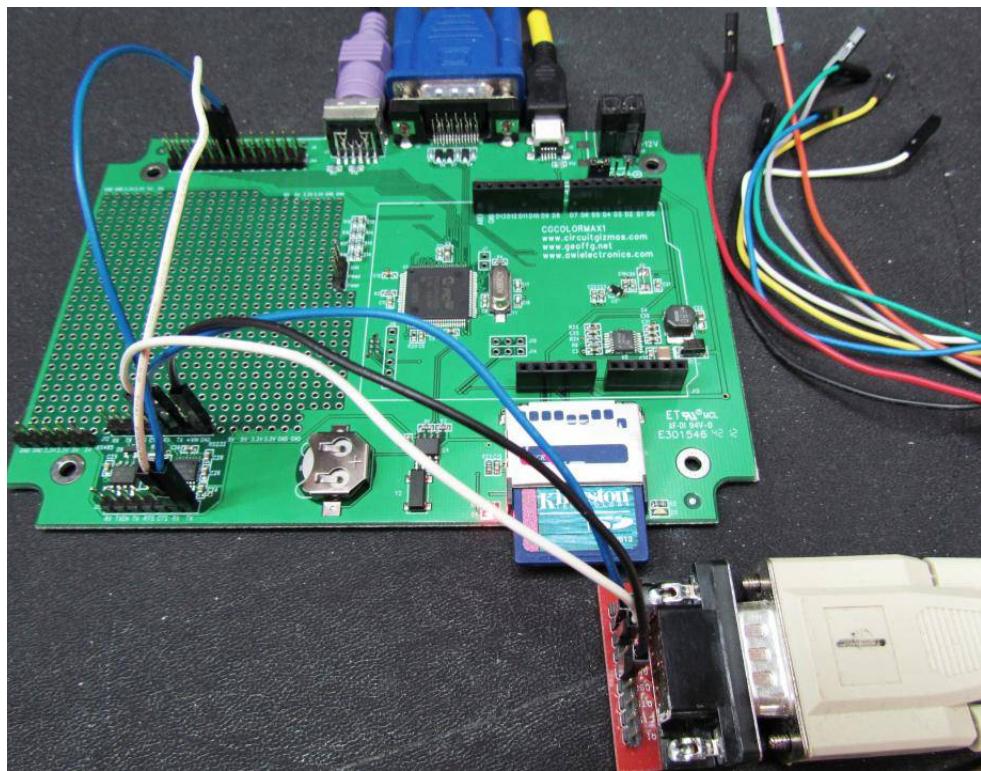


Figure 56: CGCOLORMAX1 hardware used for serial example.

OK bear with me here. There are a lot of pin numbers that need to be kept straight.

From the Maximite manual:

COM1: uses pin 15 for receive data (data in) and pin 16 for transmit data (data out). If flow control is specified pin 17 will be used for RTS (receive flow control – it is an output) and pin 18 will be CTS (transmit flow control – it is an input).

COM2: uses pin 19 for receive data (data in) and pin 20 for transmit data (data out) in the monochrome Maximite and D0 for receive data and pin D1 for transmit data on the Colour Maximite.

We will use COM1 on the CGCOLORMAX1 for this example.

The MMBasic firmware transmits data out of the Maximite on what the software refers to as I/O Pin 16. I/O Pin 16 is located on the CGCOLORMAX1 on J9 (and J9a). If you count pins from the square pin (1) on J9 and zig zag back and forth on that connector then I/O Pin 16 is located on that connector's 14th pin. On the hardware picture above I connected to J9.14 with a blue wire.

This TX signal needs to go to the RS232 circuit connector on the CGCOLORMAX1. This would be J10, along the edge of the printed circuit board. The blue wire needs to connect to pin 7 on J10. The circuit converts this TX signal to RS232 voltage levels and the TX signal then appears on J12, pin 6. This is connected to a PC serial port RS232 input via the 2nd blue wire.

The PC receives this signal from the CGCOLORMAX1 and transmits a reply to the receive circuit via a white wire connected to J12 pin 5. The RS232 circuit on the CGCOLORMAX1 converts the voltage level to something safe for the CGCOLORMAX1 and that received signal appears on J10, pin 6.

A white wire connects the J10 pin 6 signal to J9 (or J9a) Pin 12. That pin is associated with software I/O Pin 15 to receive the date.

Whew! Did you follow all of that?

Keep in mind that RS232 circuit pin numbers will be different for the CGCOLORMAX2.

A serial port application on the PC is set to the right baud (in this case 1200) to match that of the MMIDE applet.

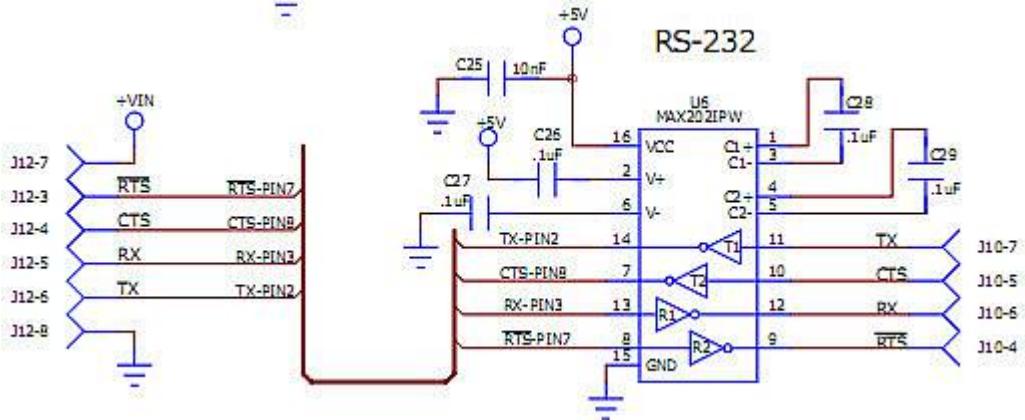


Illustration 57: The RS232 circuit of the CGCOLORMAX1. The driver chip is populated on the CGCOLORMAX1.

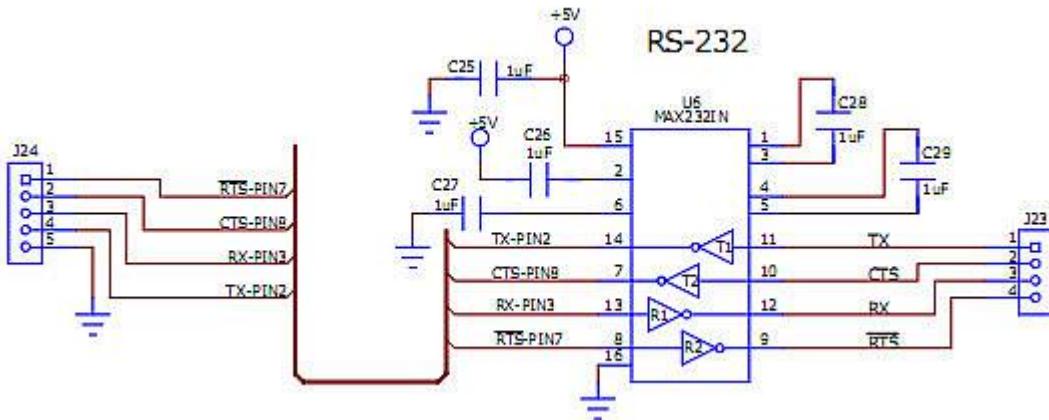


Illustration 58: The RS232 circuit of the CGCOLORMAX2. The driver chip is not populated on the CGCOLORMAX2. The chip is through-hole and can be added by soldering it in place.

The CGCOLORMAX1 and CGCOLORMAX2 both contain circuitry that supports RS232 communication. On the CGCOLORMAX1, the RS232 driver is included on the board. The CGCOLORMAX2 contains all of the associated circuitry, except for the RS232 driver chip. That chip can be purchased separately.

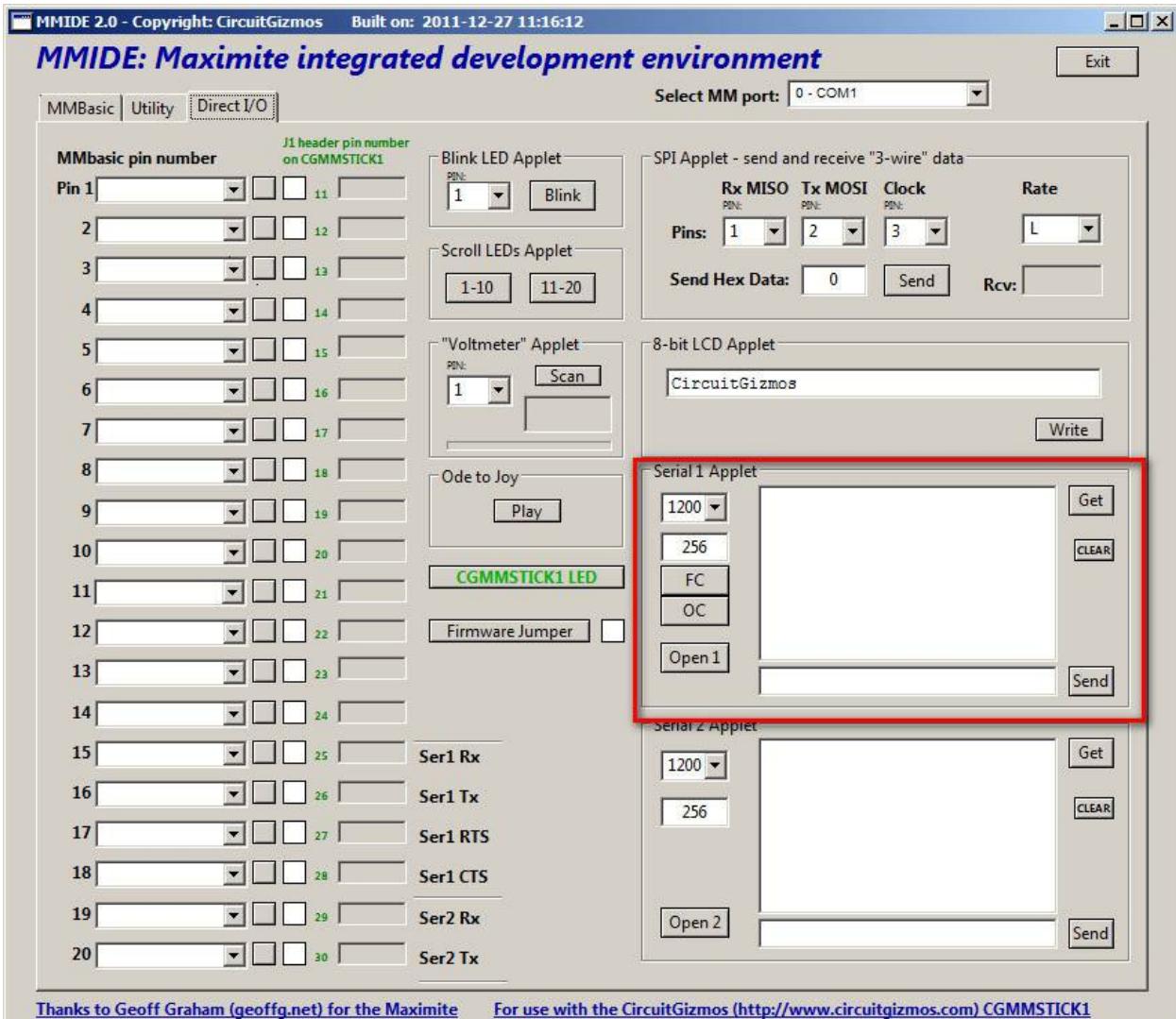


Figure 59: Using the serial applet for port 1.

The MMIDE applet communicates with the PC through the hardware described above.

The “Open 1” button sends the command:

```
OPEN "COM1:1200, 256" AS #1
```

to the CGCOLORMAX1 to open the serial port.

The text string that you type in the send text area is sent to the serial port when the “Send” button is pushed.

```
PRINT #1, "This is a test"
```

"This is a test" is then sent out the serial port of the CGCOLORMAX1 to the PC.

A string typed in the serial port application on the PC is received by the CGCOLORMAX1 into the serial buffer. Pressing the "Get" button sends a command from MMIDE to the CGCOLORMAX1 to get stuff out of it's buffer and then MMIDE displays it.

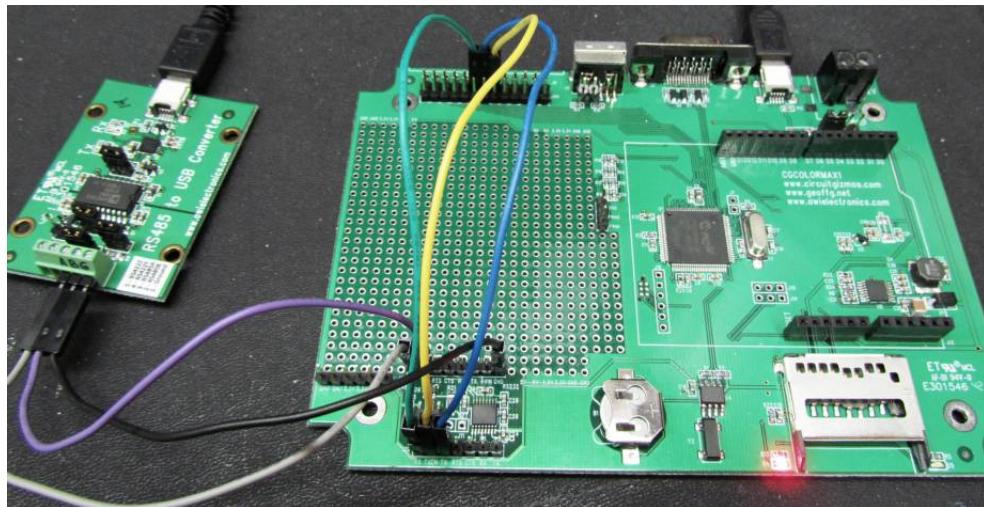


Illustration 60: CGCOLORMAX1 connected to an RS485 device.

MMBasic (version 4.2+) supports the driver enable needed to correctly operate an RS485 driver chip. "DE" is the option available for the OPEN command that uses a selected port line for control of an RS485 chip.

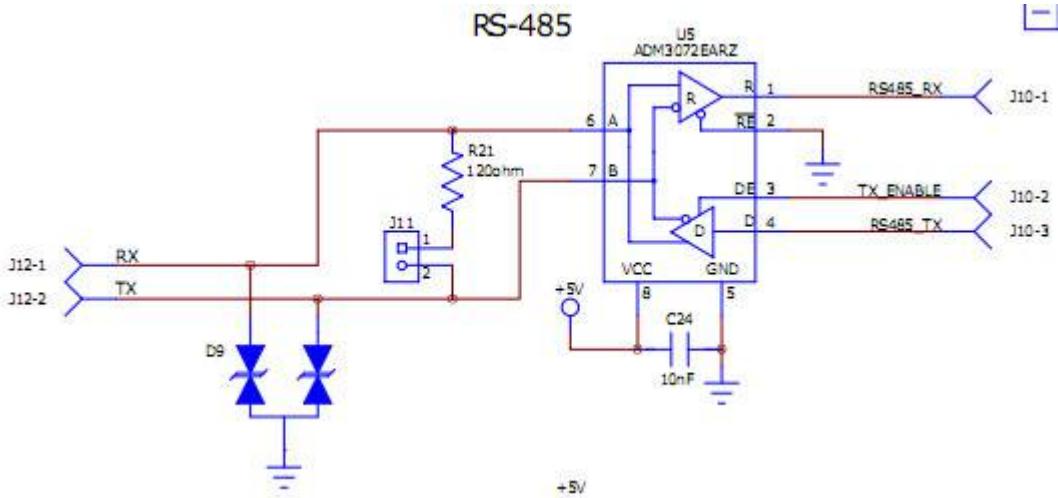


Illustration 61: The RS485 circuit of the CGCOLORMAX1. The driver chip is populated on the CGCOLORMAX1.

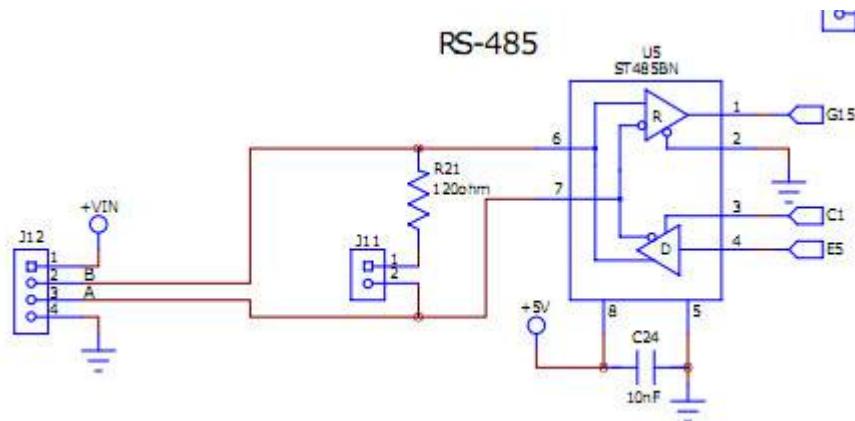


Illustration 62: The RS485 circuit of the CGCOLORMAX2. The driver chip is not populated on the CGCOLORMAX2. The chip is through-hole and can be added by soldering it in place.

The CGCOLORMAX1 and CGCOLORMAX2 both contain circuitry that supports RS485 communication. On the CGCOLORMAX1, the RS485 driver is included on the board. The CGCOLORMAX2 contains all of the associated circuitry, except for the RS485 driver chip. That chip can be purchased separately.

Serial Communications

Information from Geoff Graham <http://geoffg.net>

Two serial ports are available for asynchronous serial communications. They are labeled COM1: and COM2: and are opened in a manner similar to opening a file. After being opened they will have an associated file number (like an opened disk file) and you can use any commands that operate with a file number to read and write to/from the serial port. A serial port is also closed using the CLOSE command.

The following is an example:

```
' open the first serial port with a speed of 4800 baud
OPEN "COM1:4800" AS #5

'send the string "Hello" out of the serial port
PRINT #5, "Hello"

' get up to 20 characters from the serial port
Data$ = INPUT$(20, #5)

' close the serial port
CLOSE #5
```

The Serial OPEN Command

A serial port is opened using the command:

```
OPEN comspec$ AS #fnbr
```

The transmission format is fixed at 8 data bits, no parity and one stop bit.

'fnbr' is the file number to be used. It must be in the range of 1 to 10. The # is optional.

'comspec\$' is the communication specification and is a string (it can be a string variable) specifying the serial port to be opened and optional parameters.

It has the form "COMn: baud, buf, int, intlevel, FC, OC" where:

```
COMn: baud, buf, int, intlevel, FC, OC
```

where:

'n' is the serial port number for either COM1: or COM2:

'baud' is the baud rate, either 19200, 9600, 4800, 2400, 1200, 600, 300 bits per second. Default is 9600.

'buf' is the buffer sizes in bytes. Two of these buffers will be allocated from memory, one for transmit and one for receive. The default size is 256 bytes.

'int' is the line number or label of an interrupt routine to be invoked when the serial port has received some data. The default is no interrupt.

'intlevel' is the number of characters that must be waiting in the receive queue before the receive interrupt routine is invoked. The default is 1 character.

All parameters except the serial port name (COMn:) are optional. If any one parameter is left out then all the following parameters must also be left out and the defaults will be used.

Three options can be optionally added, these are FC, DE, and OC.

'FC' will enable hardware flow control. Flow control can only be specified on COM1: and it enables two extra signals, Request To Send (receive flow control) and Clear To Send (transmit flow control). Default is no flow control.

'DE' will enable the Data output Enable (DE) signal for RS485. The DE signal will go high just before a byte is transmitted and will go low when the last byte in the transmit buffer has been sent.

'OC' will force the output pins (Tx and optionally RTS) to be open collector and can be used on both COM1: and COM2:. The default is normal (0 to 3.3V) output. Open collector is like having a transistor that when turned on connects the output to ground, but when off leaves the output floating.

These options must be at the end of 'comspec' and, if specified, OC must be last. FC and DE apply only to COM1 and are mutually exclusive (you cannot specify both).

Opening a serial port using all the defaults:

```
OPEN "COM2:" AS #2
```

Opening a serial port specifying only the baud rate (4800 bits per second):

```
OPEN "COM2:4800" AS #1
```

Opening a serial port specifying the baud rate (9600 bits per second) and buffer size (1KB) but no flow control:

```
OPEN "COM1:9600, 1024" AS #8
```

The same as above but with receive flow control (RTS) and transmit flow control (CTS) enabled:

```
OPEN "COM1:9600, 1024, FC" AS #8
```

An example specifying everything including an interrupt, an interrupt level, flow control and open collector:

```
OPEN "COM1:19200, 1024, ComIntLabel, 256, FC, OC" AS #5
```

Serial Input/Output Pin Allocation

COM1: uses pin 15 for receive data (data in) and pin 16 for transmit data (data out). If flow control is specified pin 17 will be used for RTS (receive flow control – it is an output) and pin 18 will be CTS (transmit flow control – it is an input). If the Data output Enable (DE) signal for RS485 is enabled it will be on pin 17.

COM2: uses pin 19 for receive data (data in) and pin 20 for transmit data (data out) in the monochrome Maximite and D0 for receive data and pin D1 for transmit data on the Color Maximite.

When a serial port is opened the pins used by the port are automatically set to input or output as required and the SETPIN and PIN commands are disabled for the pins. When the port is closed (using the CLOSE command) all pins used by the serial port will be set to a not-configured state and the SETPIN command can then be used to reconfigure them.

The signal polarity is standard for devices running at TTL voltages (not RS232). Idle is voltage high, the start bit is voltage low, data uses a high voltage for logic 1 and the stop bit is voltage high. The flow control pins (RTS and CTS) use a low voltage to signal stop sending data and high as OK to send. These signal levels allow you to directly connect to devices like the EM-408 GPS module (which uses TTL voltage levels).

Reading and Writing the Serial Port

Once a serial port has been opened you can use any commands or functions that use a file number to write and read from the port. Generally the PRINT command is the best method for transmitting data and the INPUT\$() function is the most convenient way of getting data that has been received. When using the INPUT\$() function the number of characters specified will be the maximum number of characters returned but it could be less if there are less characters in the receive buffer. In fact the INPUT\$() function will immediately return an empty string if there are no characters available in the receive buffer.

The LOC() function is also handy; it will return the number of characters waiting in the receive buffer (ie, the number characters that can be retrieved by the INPUT\$() function). The EOF() function will return true if there are no characters waiting. The LOF() function will return the space (in characters) remaining in the transmit buffer.

When outputting to a serial port (ie, using PRINT #n, data) the command will pause if the output buffer is full and wait until there is sufficient space to place the new data in the buffer before returning. If the receive buffer overflows with incoming data the serial port will automatically discard the oldest data to make room for the new data.

Serial ports can be closed with the CLOSE command. This will discard any characters waiting in the buffers, return the buffer memory to the memory pool and set all pins used by the port to the not configured state. A serial port is also automatically closed when commands such as RUN and NEW are issued.

Opening a Serial Port as the Console

A serial port can be opened as the console for MMBasic. The command is:

```
OPEN comspec AS CONSOLE
```

In this case any characters received from the serial port will be treated the same as keystrokes received from the keyboard and any characters sent to the video output will also be transmitted via the serial port. This enables a user with a terminal at the end of the serial link to exercise remote control of MMBasic. For example, via a modem.

Note that only one serial port can be opened "AS CONSOLE" at a time and it will remain open until explicitly closed using the CLOSE CONSOLE command. It will not be closed by commands such as NEW and RUN.

Project #8

CGMMSTICK SPI Interface Example

As an example of the operation of the SPI function of the CGMMSTICK, this example interfaces to a SPI potentiometer.

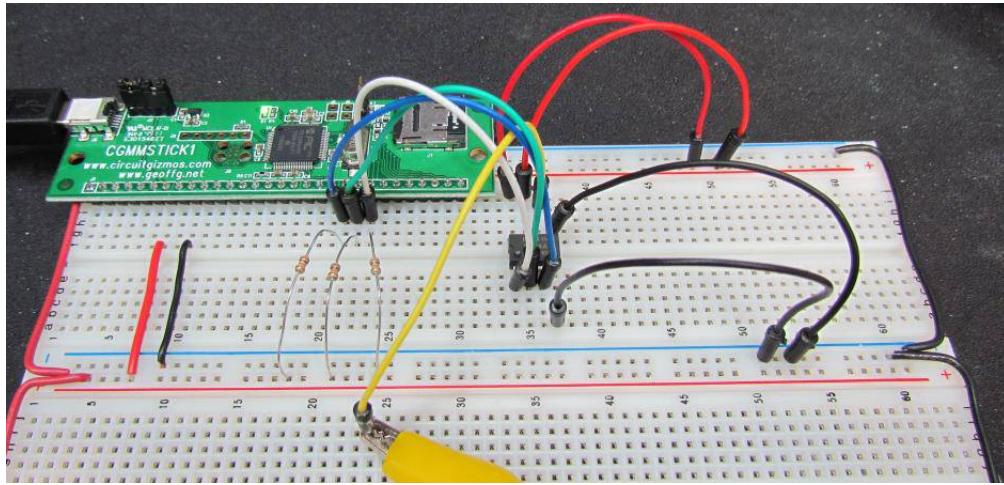


Figure 63: SPI circuit using an MCP41100 serial potentiometer.

An MCP41100 SPI potentiometer was selected from Microchip Technology. This is a 100k potentiometer that can be adjusted in 256 steps from one end of the dial to another.

Block Diagram

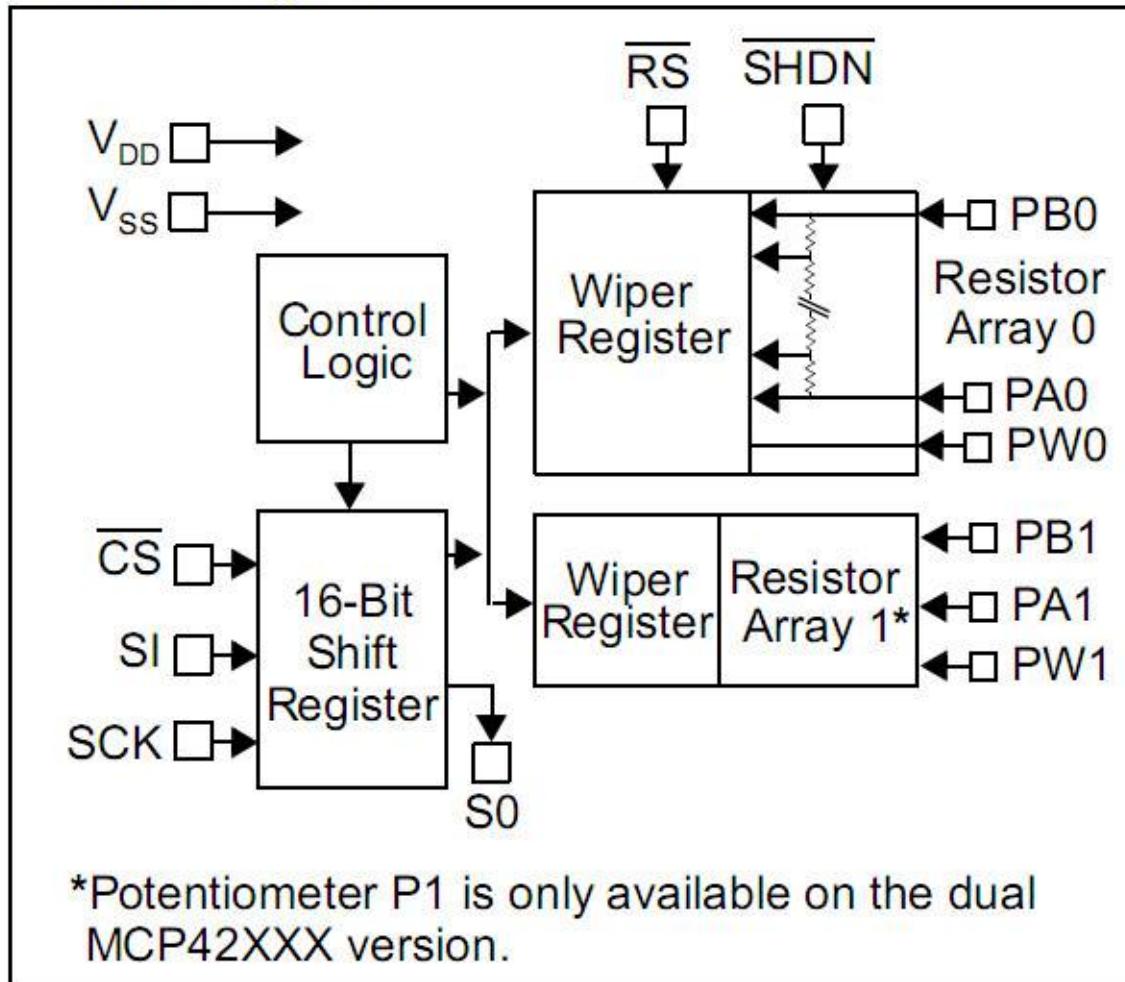


Illustration 64: DPI pot block diagram.

PB0 and PA0 were connected to 5V and ground respectively. PW0 is the 'wiper' and will move in position between 0V and 5V under control of the CGMMSTICK via the commands sent through MMIDE.

The SPI serial lines are all set as OC (open collector) with a pull up resistor of 10k ohms.

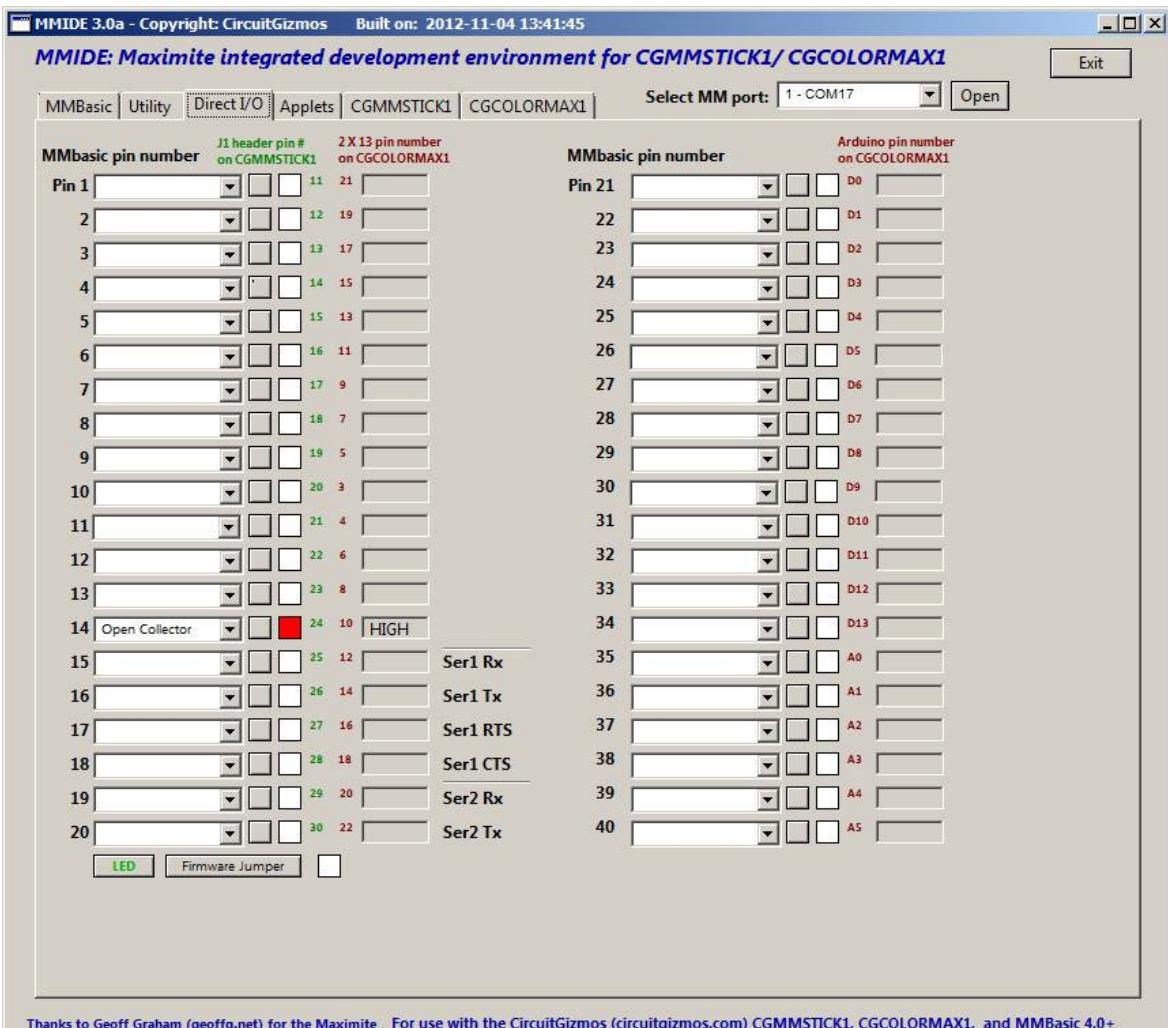


Figure 65: The chip select line for the digital pot.

The CS (chip select) line of the digital pot is connected to I/O Pin 14, in open collector mode (with 10k ohm pull up). It is active low. The data transfer starts with this line high. The line is brought low, serial data is transmitted, and the line returned high.

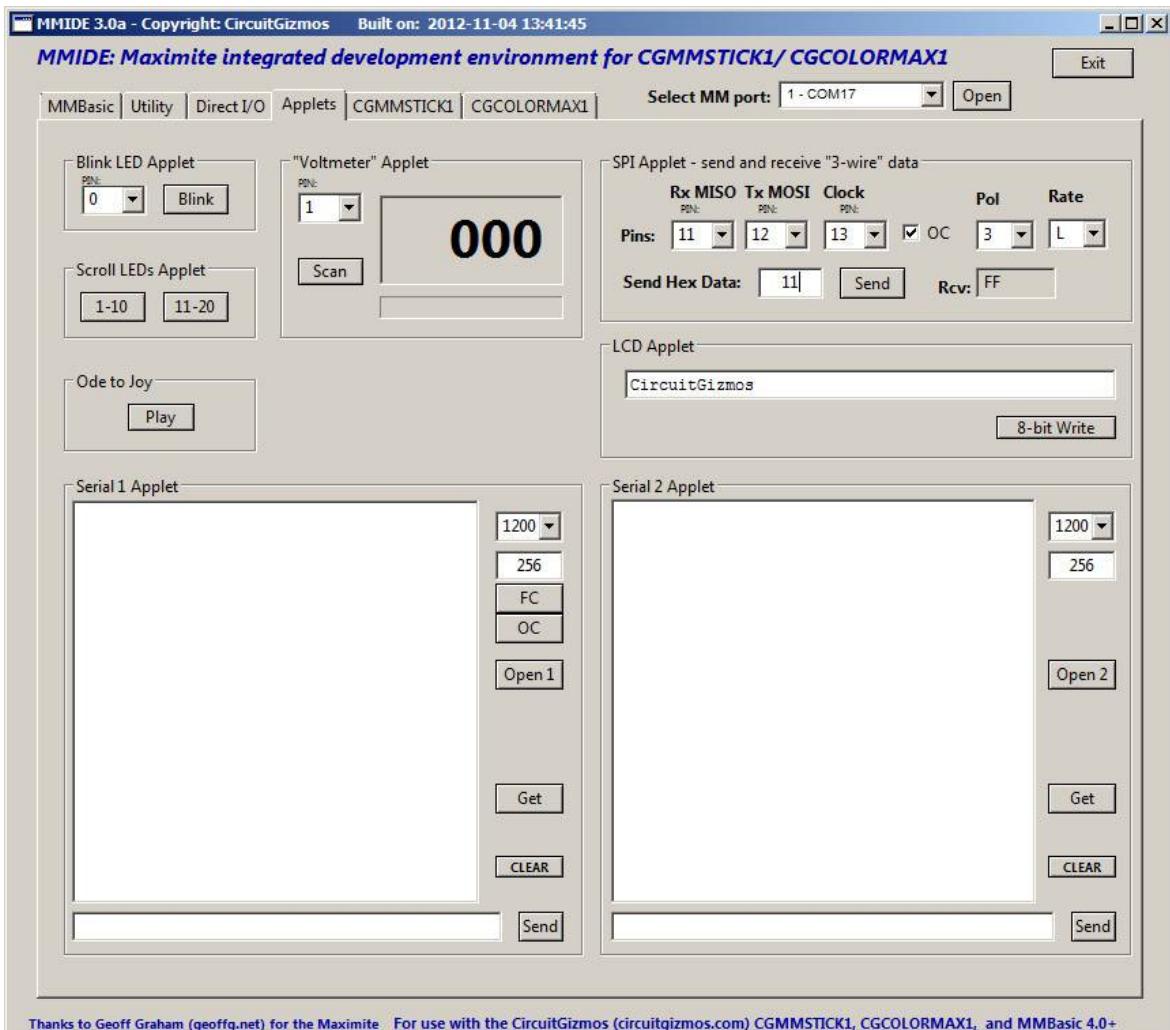


Figure 66: The SPI applet settings appropriate for the MCP41100.

The SPI applet is used to transmit two bytes to the digital pot.

Pins 11, 12, and 13 are used to communicate with the digital pot. Note that this hardware uses the 8-pin MCP41100, so there actually is no data read from the digital pot.

The clock and data pins are set to OC. The rate is L – low. The polarity for clock/data 1s set to 3 - clock is active low, data is captured on the rising edge and output on the falling edge.

The chip is selected (CS low) and 11 is clocked to the chip. This is the command byte for this chip. Then a value from 0 (0v) to 255 (5V) is clocked to the chip before returning high. When the CS is high the pot switches to the commanded setting.

Serial Peripheral Interface (SPI)

Information from Geoff Graham <http://geoffg.net>

The Serial Peripheral Interface (SPI) communications protocol is used to send and receive data between integrated circuits.

The SPI function in MMBasic acts as the master (ie, MMBasic generates the clock).

The syntax of the function is:

```
received_data = SPI( rx, tx, clk, data_to_send, speed, mode, bits )
```

Data_to_send, speed, mode and bits are all optional. If not required they can be represented by either empty space between the commas or left off the end of the list.

Where:

'rx' is the pin number for the data input (MISO)

'tx' is the pin number for the data output (MOSI)

'clk' is the pin number for the clock generated by MMBasic (CLK)

'data_to_send' is optional and is an integer representing the data byte to send over the output pin. If it is not specified the 'tx' pin will be held low.

'speed' is optional and is the speed of the clock. It is a single letter either H, M or L where H is 3 MHz, M is 500 KHz and L is 50 KHz. Default is H.

'mode' is optional and is a single numeric digit representing the transmission mode – see Transmission Format below. The default mode is 3.

'bits' is optional and represents the number of bits to send/receive. Range is 1 to 23 (this limit is defined by how many bits can be stored in a floating point number). The default is 8.

The SPI function will return the data received during the transaction as an integer. Note that a single SPI transaction will send data while simultaneously receiving data from the slave (which is often discarded).

Examples

Using all the defaults:

```
A = SPI(11, 12, 13)
```

Specifying the data to be sent:

```
A = SPI(11, 12, 13, &hE4)
```

Setting the mode but using the defaults for data to send and speed:

```
A = SPI(11, 12, 13, , , 2)
```

An example specifying everything including a 12 bit data transfer:

```
A = SPI(11, 12, 13, &hE4, M, 2, 12)
```

Transmission Format

The most significant bit is sent and received first. The format of the transmission can be specified by the 'mode' as follows:

Mode 0. CPOL 0, CPHA 0 - Clock is active high, data is captured on the rising edge and output on the falling edge

Mode 1. CPOL 0, CPHA 1 - Clock is active high, data is captured on the falling edge and output on the rising edge

Mode 2. CPOL 1, CPHA 0 - Clock is active low, data is captured on the falling edge and output on the rising edge

Mode 3. CPOL 1, CPHA 1 - Clock is active low, data is captured on the rising edge and output on the falling edge

Before invoking this function the 'rx' pin must be configured as an input using the SETPIN command and the 'tx' and 'clk' pins must be configured as outputs (either normal or open collector) again using the SETPIN command. The clock pin should also be set to the correct polarity (using the PIN function) before the SETPIN command so that it starts as inactive.

The SPI enable signal is often used to select a slave and "prime" it for data transfer. This signal is not generated by this function and (if required) should be generated using the PIN function on another pin.

The SPI function does not "take control" of the I/O pins like the serial and I2C protocols and the PIN command will continue to operate as normal on them. Also, because the I/O pins can be changed between function calls it is possible to communicate with many different SPI slaves on different I/O pins.

The following example will send the command &h80 and receive two bytes from the slave SPI device. Because the mode, speed and number of bits are not specified the defaults are used.

```
' set rx pin as a digital input
```

```
SETPIN 18, 2

` set tx pin as an output
SETPIN 19, 8

` set clk pin high then set it as an output
PIN(20) = 1 : SETPIN 20, 8

` pin 11 will be used as the enable signal
PIN(11) = 1 : SETPIN 11, 8

` assert the enable line (active low)
PIN(11) = 0

` send the command and ignore the return
junk = SPI(18, 19, 20, &h80)

` get the first byte from the slave
byte1 = SPI(18, 19, 20)

` get the second byte from the slave
byte2 = SPI(18, 19, 20)

` deselect the slave
PIN(11) = 1
```

Project #9

CGMMSTICK I2C Interface Example

As an example of the operation if the I2C function of the CGMMSTICK, this example interfaces to a PCF8574A I/O expansion device.

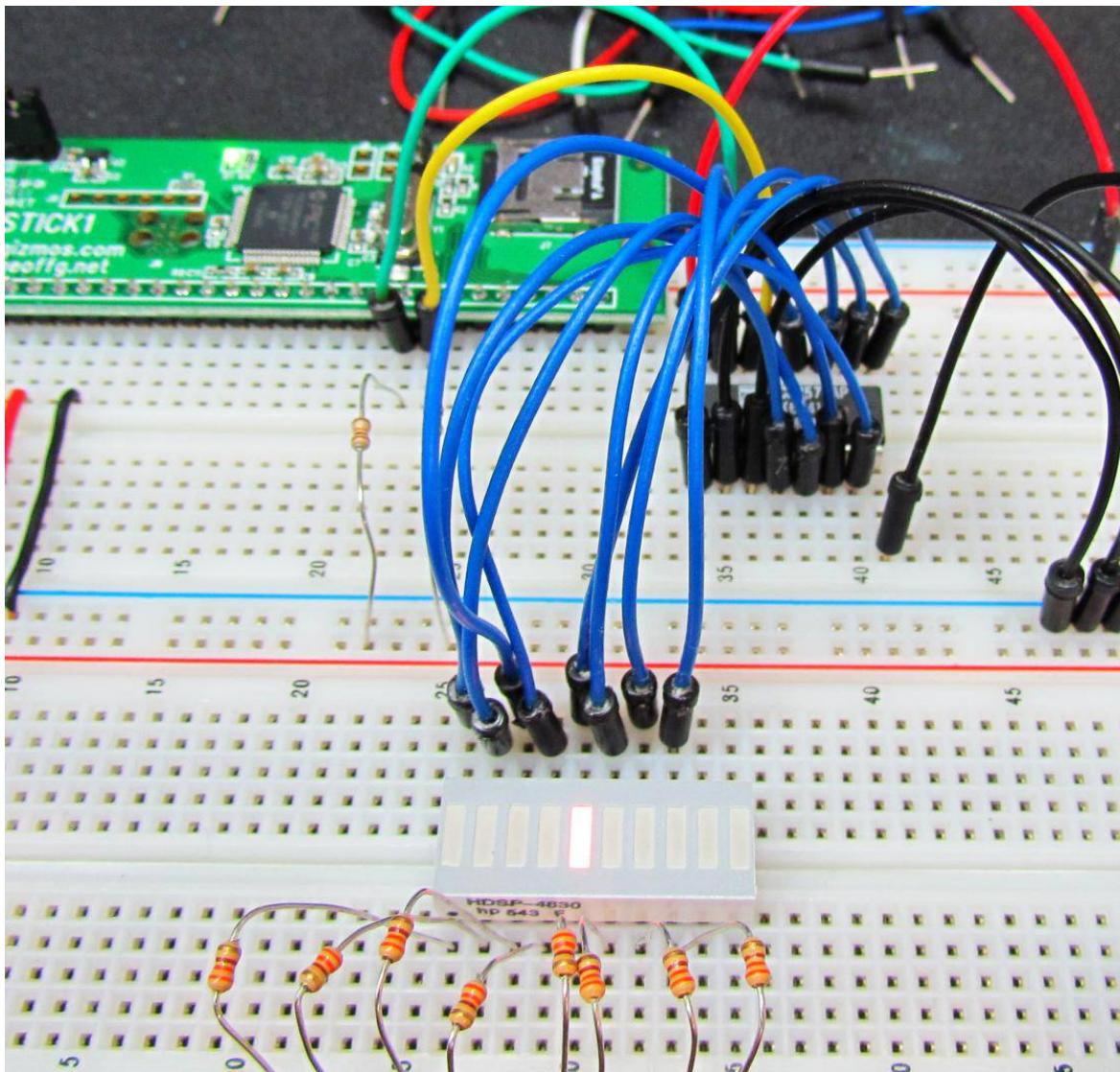


Figure 67: PCF8574A I2C port expander circuit.

I/O Pins 12 and 13 are used for I2C, as described in the documentation.

The code used in this example assumes that the address pins (A0-A2) are all grounded, giving this chip an address (offset) of 0.

The example code runs a 'cyon' pattern on the LEDs.

```

' Enable I2C
' speed = 100 kHz
' timeout = 100 milliseconds
I2CEN 100, 100

DO

    FOR loop1 = 0 TO 7

        cylon = 2^loop1
        cylon = (cylon XOR &hFF)

        ' Send I2C command to PCF8574A
        ' 8574A address = &h38
        ' send 1 byte
        I2CSEND &h38, 0, 1, cylon

        PAUSE 100

    NEXT loop1

    FOR loop2 = 6 TO 1 STEP -1

        cylon = 2^loop2
        cylon = (cylon XOR &hFF)

        ' Send I2C command to PCF8574A
        ' 8574A address = &h38
        ' send 1 byte
        I2CSEND &h38, 0, 1, cylon

        PAUSE 100

    NEXT loop2

LOOP

' Disable I2C
I2CDIS

```

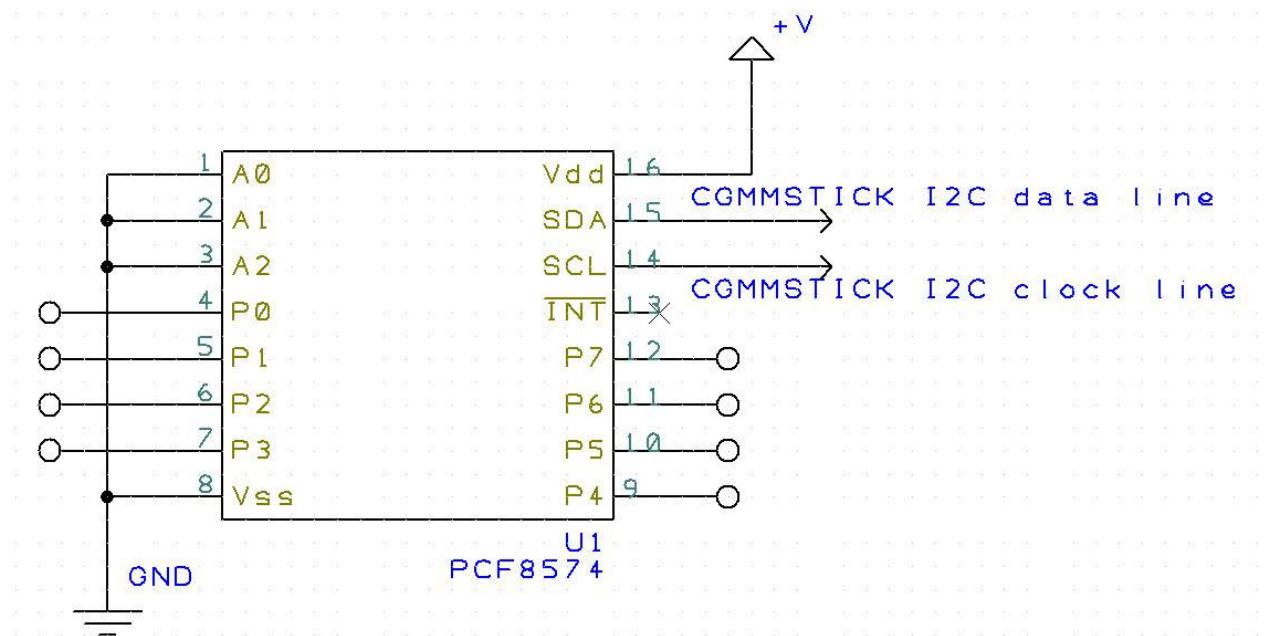


Illustration 68: Simplified schematic for general CGMMSTICK to PCG8574 interface. This chapter's example also includes the resistors and LEDs on the output lines as well as pull up resistors for the I2C lines.

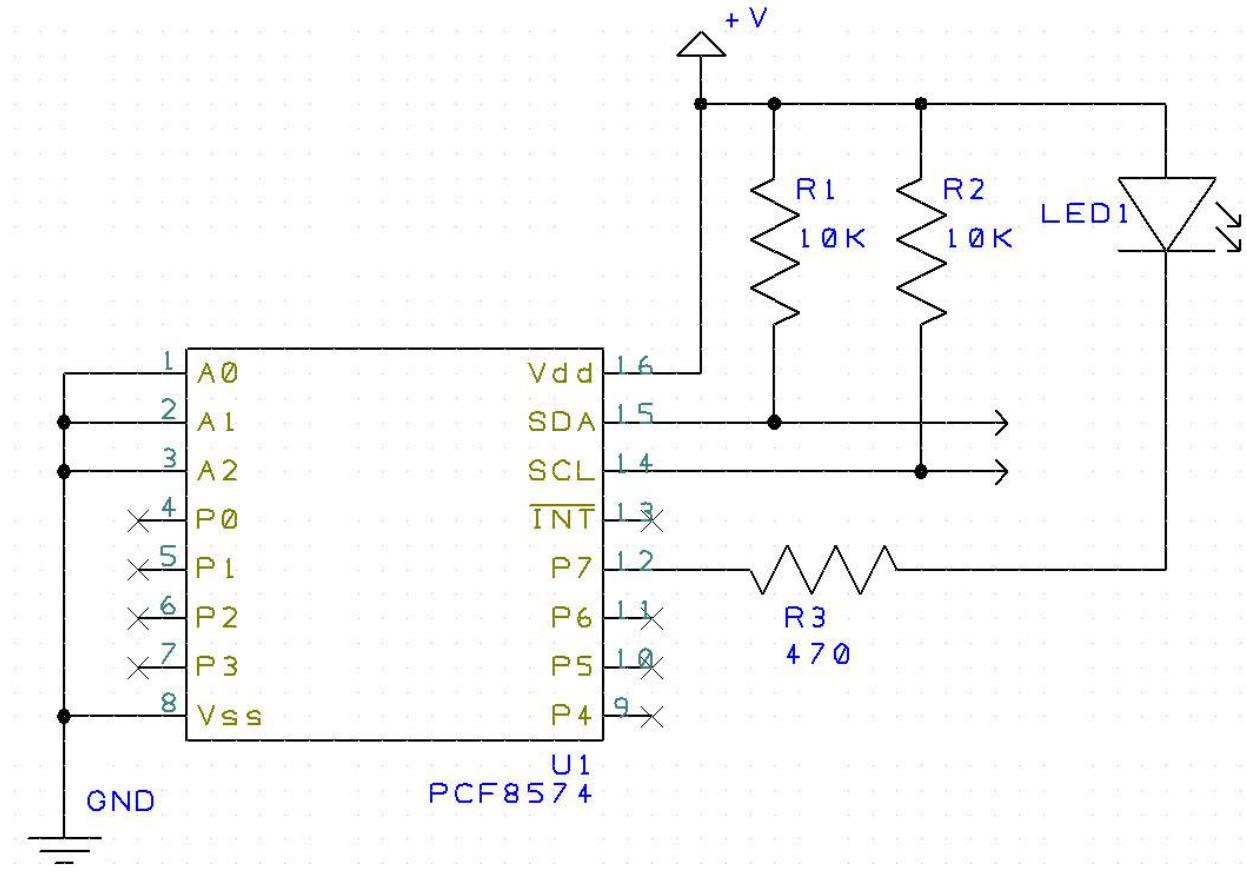


Illustration 69: Schematic including I2C pull up resistors and the resistor/LED combination for one of the eight I/O lines.

Inter Integrated Circuit (I2C) bus

Information from Geoff Graham <http://geoffg.net>

The Inter Integrated Circuit (I2C) bus was developed by the electronics giant Philips for the transfer of data between integrated circuits. It has been adopted by many manufacturers and can be used to communicate with many devices including memories, clocks, displays, speech module, etc. Using the I2C bus is complicated and if you do not need to communicate with these devices you can safely ignore this section.

This implementation was developed by Gerard Sexton and fully supports master and slave operation, 10 bit addressing, address masking and general call, as well as bus arbitration (ie, bus collisions in a multi-master environment).

In the master mode, there is a choice of 2 modes: interrupt and normal. In normal mode, the I2C send and receive commands will not return until the command completes or a timeout occurs (if the timeout option has been specified). In interrupt mode, the send and receive commands return immediately allowing other MMBasic commands to be executed while the send and receive are in progress. When the send/receive transactions have completed, an MMBasic interrupt will be executed. This allows you to set a flag or perform some other processing when this occurs.

When enabled the I2C function will take control of the external I/O pins 12 and 13 and override SETPIN and PIN() commands for these pins. Pin 12 becomes the I2C data line (SDA) and pin 13 the clock (SCL). Both of these pins should have external pullup resistors installed (typical values are 10KΩ for 100KHz or 2KΩ for 400 kHz).

Be aware that when running the I2C bus at above 150 kHz the cabling between the devices becomes important. Ideally the cables should be as short as possible (to reduce capacitance) and also the data and clock lines should not run next to each other but have a ground wire between them (to reduce crosstalk). If the data line is not stable when the clock is high, or the clock line is jittery, the I2C peripherals can get "confused" and end up locking the bus (normally by holding the clock line low). If you do not need the higher speeds then operating at 100 kHz is the safest choice.

There are four commands for master mode; I2CEN, I2CDIS, I2CSEND and I2CRCV. For slave mode the commands are; I2CSEN, I2CSDIS, I2CSSEND and I2CSRCV. The master and slave modes can be enabled simultaneously; however, once a master command is in progress, the slave function will be "idle" until the master releases the bus. Similarly, if a slave command is in progress, the master commands will be unavailable until the slave transaction completes.

Both the master and slave modes use an MMBasic interrupt to signal a change in status. These interrupt routines operate the same as a general interrupt on an external I/O pin (see page 4 for a description) and must be terminated with an IRETURN command to return control to the main program when completed. The automatic variable MM.I2C will hold the result of a command or action.

Project #10

CGMMSTICK 1-Wire Interface Example

As an example of the operation of the 1-wire function of the CGMMSTICK, this example interfaces to a DS1822 temperature device.

From the DS1822 datasheet:

The DS1822 digital thermometer provides 9- to 12-bit centigrade temperature measurements and has an alarm function with NV user-programmable upper and lower trigger points. The DS1822 communicates over a 1-Wire bus that by definition requires only one data line (and ground) for communication with a central microprocessor. It has an operating temperature range of -55°C to +125°C and is accurate to $\pm 2.0^{\circ}\text{C}$ over the range of -10°C to +85°C.

Unique 1-Wire® interface requires only one port pin for communication.

Each device has a unique 64-bit serial code stored in an on-board ROM.

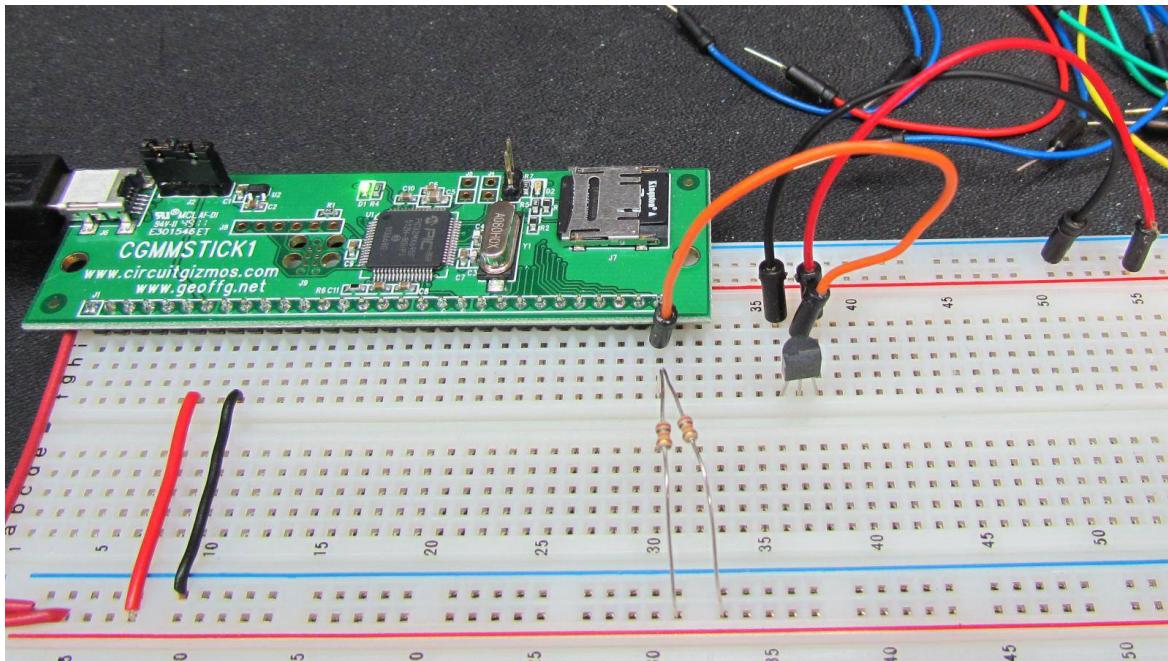
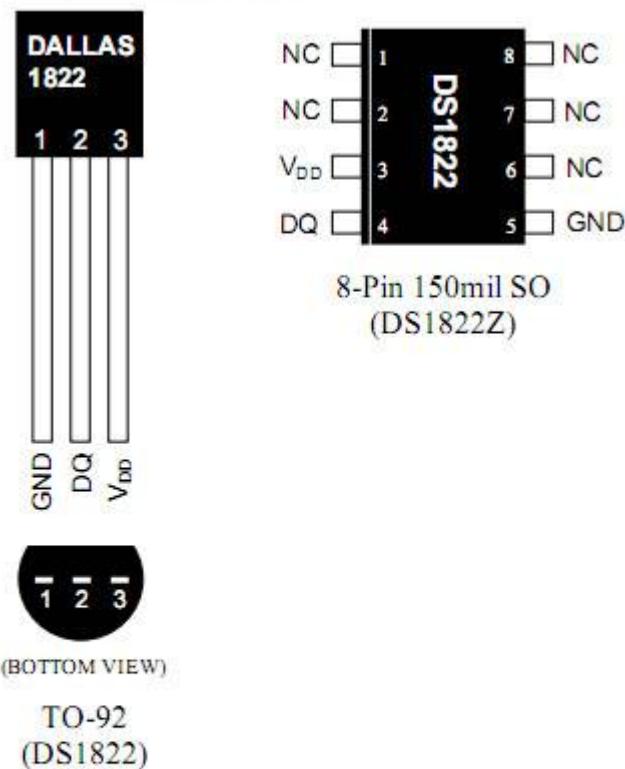


Illustration 70: 1-wire hardware.

The hardware couldn't be simpler.

PIN ASSIGNMENT



TO-92
(DS1822)

PIN DESCRIPTION

- GND - Ground
- DQ - Data In/Out
- V_{DD} - Power Supply Voltage
- NC - No Connect

Illustration 71: 1-wire temperature device.

The DS1822 pin 1 was connected to ground, pin 3 connected to 5V, and the center pin, pin 2, connected to the CGMMSTICK I/O pin 20. The center pin is the data pin of the 1-wire device. I/O pin 20 has a 5k ohm (two 10k in parallel) pullup.

The screenshot shows the MMIDE 3.0a interface. The title bar reads "MMIDE 3.0a - Copyright: CircuitGizmos Built on: 2012-11-04 14:28:59". The menu bar includes "MMBasic", "Utility", "Direct I/O", "Applets", "CGMMSTICK1", and "CGCOLORMAX1". The toolbar has buttons for "Select MM port: 1 - COM17", "Open", "Run", "Stop", "New", and "CLEAR SCROLLBACK". The main window displays the following BASIC code:

```

' Reset the 1w bus, DS1822 device
OWRESET 20

' Start temperature conversion
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &h44 - Start conversion
OWWRITE 20, 1, 2, &hCC, &h44

' Wait a second for conversion process to finish
PAUSE 100

' Read data/temperature command
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &hBE - read data command
OWWRITE 20, 1, 2, &hCC, &hBE

' Read data/temperature from DS1822
' Pin 20, send reset after, read two bytes
OWREAD 20, 2, 2, LowTemp, HighTemp

' Combine and adjust according to data sheet
Cel = ((HighTemp And &b111) * 256 + LowTemp) / 16

' Print Celsius and convert/print Fahrenheit
Print "Celsius" Cel
Print "Fahrenheit" ( 9 / 5 ) * Cel + 32;

> run
Celsius 28.1875
Fahrenheit 82.7375
>

```

At the bottom of the window, a note reads: "Thanks to Geoff Graham (geoffg.net) for the Maximite. For use with the CircuitGizmos (circuitgizmos.com) CGMMSTICK1, CGCOLORMAX1, and MMBasic 4.0+."

Illustration 72: 1-wire program loaded and running.

From the software perspective reading from a 1-wire device isn't as simple as just sending a read command. You have to write to the device first to tell it you want to read.

Furthermore it takes a moment for the device to start and complete the temperature measurement process.

First the 'bus' should be reset. In this case the 1-wire bus is I/O Pin 20 and only contains the single DS1822 device.

```

' Reset the 1w bus, DS1822 device
OWRESET 20

```

The reset command selects the pin (bus) to reset and optionally can receive an indication that there are 1 or more devices present on the bus. This puts a reset pulse on the bus.

All devices have a serial number, so that if there are multiple devices on a single I/O Pin (bus), any individual device can be addressed for access.

In this example there is only a single device, so rather than talking to it by addressing it specifically, a command is used that skips the need for a serial number. This is called "Skip ROM" in the 1-wire terminology.

From the data sheet:

SKIP ROM [CCh]

The master can use this command to address all devices on the bus simultaneously without sending out any ROM code information. For example, the master can make all DS1822s on the bus perform simultaneous temperature conversions by issuing a Skip ROM command followed by a Convert T [44h] command.

Note that the Read Scratchpad [BEh] command can follow the Skip ROM command only if there is a single slave device on the bus. In this case time is saved by allowing the master to read from the slave without sending the device's 64-bit ROM code. A Skip ROM command followed by a Read Scratchpad command will cause a data collision on the bus if there is more than one slave since multiple devices will attempt to transmit data simultaneously.

The DS1822 will be told to start a temperature conversion.

```
' Start temperature conversion
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &h44 - Start conversion
OWWRITE 20, 1, 2, &hCC, &h44
```

The OWWRITE command selects a pin/bus to write to. A reset pulse again is sent to that bus and then two bytes are sent. &hCC is the command byte to skip the serial number, and &h44 tells the DS1822 to start the temperature conversion process.

The conversion process takes a little while to complete, so the program pauses to allow that conversion to finish.

```
' Wait a second for conversion process to finish
PAUSE 100
```

After that pause, a command is sent that tells the DS1822 that the next communication will be a read.

```
' Read data/temperature command
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &hBE - read data command
OWWRITE 20, 1, 2, &hCC, &hBE
```

This again resets the bus and uses "Skip ROM" since this example uses only a single 1-wire device. &hBE is the command to read.

The read command is next to read the temperature value.

```
' Read data/temperature from DS1822
' Pin 20, send reset after, read two bytes
OWREAD 20, 2, 2, LowTemp, HighTemp
```

Note that the bus is NOT reset at the start of the read command. The DS1822 expects the read to follow the command to read without a reset between the commands.

OWREAD reads the selects bus line. The options to the command allow for a selected number of bytes to be read. In this case we need to read only the first two bytes of the multi-byte register, the registers that contain the temperature data. A bus reset happens after the OWREAD command.

Math is performed to combine the two bytes that were read from the temperature device according to the DS1822 data sheet.

```
' Combine and adjust according to data sheet
Cel = ((HighTemp And &b111) * 256 + LowTemp) / 16
```

The resulting Celsius temperature is printed, then converted to Fahrenheit and that is also printed.

```
' Print Celsius and convert/print Fahrenheit
Print "Celsius" Cel
Print "Fahrenheit" ( 9 / 5 ) * Cel + 32;
```

1-Wire Interface

Information from Geoff Graham <http://geoffg.net>

The 1-Wire protocol was invented by Dallas Semiconductor to communicate with chips using a single signalling line. It is mostly used in communicating with the DS18B20 and DS18S20 temperature measuring chips. This implementation was developed for MMBasic by Gerard Sexton.

There are four commands that you can use:

OWRESET pin [,presence]

OWWRITE pin, flag, length, data [, data...]

OWREAD pin, flag, length, data [, data...]

OWSEARCH pin, srchflag, ser [,ser...]

Where:

pin - the MMBasic I/O pin to use

presence - an optional variable to receive the presence pulse (1 = device response, 0 = no device)

flag - a combination of the following options:

1 - send reset before command

2 - send reset after command

4 - only send/recv a bit instead of a byte of data

8 - invoke a strong pullup after the command (the pin will be set high and open drain disabled)

length - length of data to send or receive

data - data to send or receive

srchflag - a combination of the following options:

1 - start a new search

2 - only return devices in alarm state

4 - search for devices in the requested family (first byte of ser)

8 - skip the current device family and return the next device

16 - verify that the device with the serial number in ser is available

If srchflag = 0 (or 2) then the search will return the next device found

ser - serial number (8 bytes) will be returned (srchflag 4 and 16 will also use the values in ser)

After the command is executed, the pin will be set to the not configured state unless flag option 8 is used. The data and ser arguments can be a string, array or a list of variables.

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

There are two utility functions available:

OWCRC8(len, cdata [, cdata...]) Processes the cdata and returns the 8 bit CRC

OWCRC16(len, cdata [, cdata...]) Processes the cdata and returns the 16 bit CRC

Where:

len - length of data to process

cdata - data to process

The cdata can be a string, array or a list of variables

Project #11

CGCOLORMAX LCD Shield Example

This example project shows how to program an LCD Shield installed on the CGCOLORMAX.

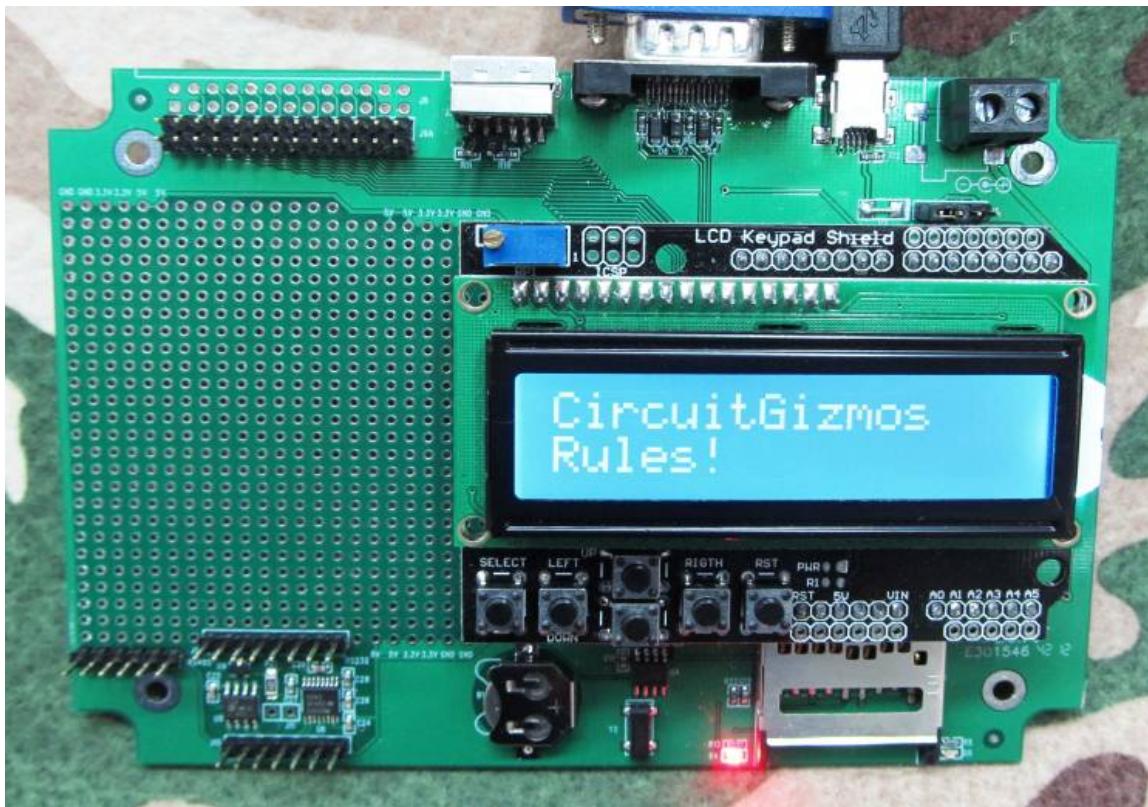


Illustration 73: A CGCOLORMAX1 with an LCD Shield attached to it. The message is right...

To use Shields, the Shield footprint on the CGCOLORMAX is populated with female single inline sockets. Two six-by-one and two eight-by-one headers are soldered in place on the CGCOLORMAX so that a Shield (or a stack of them) can be added to the board.

The LCD Shield is actually two distinct circuits: an interface to a character-based LCD module, and a group of five buttons that can be read programmatically. (A sixth button is connected to the CGCOLORMAX reset line. Be careful as this will restart the ColorMax.) There is a line to control the LCD back light, too.

The buttons connect to Shield line A0 which is read by MMBasic with PIN(35). This line should be set to be an analog input. Without any buttons pressed the input reads approximately 3.3V. Pressing the left, right, up, or down button returns a different voltage. A function translates the voltages to the integer values 0-4 with 0 being no button pressed.

The function created to return the state of the buttons is below:

```
' -----
' Return button state
' 0 = no button
' 1 = right, 2 = up, 3 = down, 4 = left
' Because A/D on CGCOLORMAX is 3.3V max and this is a
' 5V resistor chain the SEL button does not work.
' -----

FUNCTION LCDShield_Button( )
    ' Button Input line
    SETPIN 35, 1
    LCDShield_Button = 0
    IF PIN(35) < 2.85 THEN
        LCDShield_Button = 4
    ENDIF
    IF PIN(35) < 1.95 THEN
        LCDShield_Button = 3
    ENDIF
    IF PIN(35) < 1.1 THEN
        LCDShield_Button = 2
    ENDIF
    IF PIN(35) < .32 THEN
        LCDShield_Button = 1
    ENDIF
END FUNCTION
```

The LCD connects to Shield signals D4-D9 which to MMBasic are pins 25-30. There are 4 data lines connected to communicate to the LCD as well as a line to indicate to the LCD that you wish to send a command or a character and a line that is a strobe for the end of data transfer. The pin direction is set up, four bits of data (value passed to the subroutine) are set, and the enable line pulsed. This sends a nibble of a command to the LCD.

The function created to send a command to the LCD is:

```
' -----
' Write 4-bit command
' -----
SUB LCDShield_Command4( LCDShield_value )
    ' Data lines
    SETPIN 25, 8 : SETPIN 26, 8 : SETPIN 27, 8 : SETPIN 28, 8
    ' Register Select line
    SETPIN 29, 8
    ' Enable line
    SETPIN 30, 8
    ' Command (not displayable character)
    PIN(29) = 0
    ' Enable high
    PIN(30) = 1 : PAUSE 1
    ' Value onto data lines
    PIN(28) = LCDShield_value AND &B00001000
    PIN(27) = LCDShield_value AND &B00000100
    PIN(26) = LCDShield_value AND &B00000010
    PIN(25) = LCDShield_value AND &B00000001
    ' Enable low then high
    PIN(30) = 0 : PAUSE 1
    PIN(30) = 1 : PAUSE 1
END SUB
```

To write a character so that the LCD will display it, the module needs to be told that is is not getting a command but instead something to display. Also two nibbles of data are sent to the display, first the high nibble of the character to display and then the low nibble.

```
' -----
' Write 8-bit character
' -----
SUB LCDShield_Character( LCDShield_value )
    ' Character (not command)
    PIN(29) = 1
    ' Enable high
    PIN(30) = 1 : PAUSE 1
    ' Value (high nibble) onto data lines
    PIN(28) = LCDShield_value AND &B10000000
    PIN(27) = LCDShield_value AND &B01000000
    PIN(26) = LCDShield_value AND &B00100000
    PIN(25) = LCDShield_value AND &B00010000
    ' Enable low then high
    PIN(30) = 0 : PAUSE 1
    PIN(30) = 1 : PAUSE 1
    ' Value (low nibble) onto data lines
    PIN(28) = LCDShield_value AND &B00001000
    PIN(27) = LCDShield_value AND &B00000100
    PIN(26) = LCDShield_value AND &B00000010
    PIN(25) = LCDShield_value AND &B00000001
    ' Enable low then high
    PIN(30) = 0 : PAUSE 1
    PIN(30) = 1 : PAUSE 1
END SUB
```

A string subroutine uses the character subroutine to send a string to the display:

```
' -----
' Write string
' -----
SUB LCDShield_String( LCDShield_string$ )
    FOR LCDShield_loop = 1 to LEN(LCDShield_string$)
        LCDShield_Character( ASC(MID$(LCDShield_string$, LCDShield_loop )))
    NEXT
END SUB
```

The back light of the LCD display can be controlled by turning a line on and off:

```
' -----
' Backlight on/off
' 0 = off, 1 = on
' -----
SUB LCDShield_Backlight ( LCDShield_blstate )
    ' pin = 31
    SETPIN 31, 8
    PIN(31) = LCDShield_blstate
END SUB
```

With this library in place, some simple example code can operate the LCD.

First the LCD back light is tested by flashing the display:

```
' Flash display
FOR a = 1 TO 5
    LCDShield_Backlight( 0 ) : PAUSE 40
    LCDShield_Backlight( 1 ) : PAUSE 60
NEXT a
```

A series of nibbles are sent to Initialize the LCD. Four-bit (nibble) mode is selected repeatedly to be sure, and other LCD characteristics are set. The display is then cleared.

```
' Initialize display
' Select 4 bit mode
LCDShield_Command4(&H03)
LCDShield_Command4(&H03)
LCDShield_Command4(&H03)
LCDShield_Command4(&H02)

' Select 4 bit mode, two LCD lines
LCDShield_Command4(&H02)
LCDShield_Command4(&H0C)

' Select display on, cursor off, no blink
LCDShield_Command4(&H00)
LCDShield_Command4(&H0C)

' Clear display
LCDShield_Command4(&H00)
LCDShield_Command4(&H01)
```

A message to my Mom is then sent by sending each character one at a time. Then the program pauses for a second:

```
' Write Hi Mom! message to line 1
LCDShield_Character(72)
LCDShield_Character(73)

LCDShield_Character(32)

LCDShield_Character(77)
LCDShield_Character(111)
LCDShield_Character(109)

LCDShield_Character(33)

PAUSE 1000
```

The display is then cleared:

```
' Clear display
LCDShield_Command4(&H00)
LCDShield_Command4(&H01)
```

A new message is sent to the LCD by writing a few strings.

```
' Write message to line 1
LCDShield_String ( "CircuitGizmos" )

' Move to line 2
LCDShield_Command4 (&H0C)
LCDShield_Command4 (&H00)

' Write message to line 2
LCDShield_String ( "Rules!" )
```

The state of the buttons is displayed for two minutes.

```
FOR a = 1 TO 120
    PRINT LCDShield_Button()
    PAUSE 1000
NEXT a
```

When no button is pressed, 0 is printed. When a button is pressed the numbers 1-4 are printed.

The program in its entirety:

```
' =====
'
' LCDShield example
'

' =====

' Flash display
for a = 1 to 5
    LCDShield_Backlight( 0 ) : PAUSE 40
    LCDShield_Backlight( 1 ) : PAUSE 60
next a

' Initialize display
' Select 4 bit mode
LCDShield_Command4(&H03)
LCDShield_Command4(&H03)
LCDShield_Command4(&H03)
LCDShield_Command4(&H02)

' Select 4 bit mode, two LCD lines
LCDShield_Command4(&H02)
LCDShield_Command4(&H0C)

' Select display on, cursor off, no blink
LCDShield_Command4(&H00)
LCDShield_Command4(&H0C)

' Clear display
LCDShield_Command4(&H00)
LCDShield_Command4(&H01)

'

' Write Hi Mom! message to line 1
LCDShield_Character(72)
LCDShield_Character(73)

LCDShield_Character(32)

LCDShield_Character(77)
LCDShield_Character(111)
```

```

LCDShield_Character(109)

LCDShield_Character(33)

PAUSE 1000

' Clear display
LCDShield_Command4(&H00)
LCDShield_Command4(&H01)

' Write message to line 1
LCDShield_String ( "CircuitGizmos" )

' Move to line 2
LCDShield_Command4(&H0C)
LCDShield_Command4(&H00)

' Write message to line 2
LCDShield_String ( "Rules!" )

for a = 1 to 500
    PRINT LCDShield_Button()
    PAUSE 1000
next a

' -----
' LCDShield library
' -----

' -----
' Backlight on/off
' 0 = off, 1 = on
' -----

SUB LCDShield_Backlight ( LCDShield_bstate )
    ' pin = 31
    SETPIN 31, 8
    PIN(31) = LCDShield_bstate
END SUB

' -----
' Write 4-bit command

```

```

' -----
SUB LCDShield_Command4( LCDShield_value )
    ' Data lines
    SETPIN 25, 8 : SETPIN 26, 8 : SETPIN 27, 8 : SETPIN 28, 8

    ' Register Select line
    SETPIN 29, 8

    ' Enable line
    SETPIN 30, 8

    ' Command
    PIN(29) = 0

    ' Enable high
    PIN(30) = 1 : PAUSE 1

    ' Value onto data lines
    PIN(28) = LCDShield_value AND &B00001000
    PIN(27) = LCDShield_value AND &B00000100
    PIN(26) = LCDShield_value AND &B00000010
    PIN(25) = LCDShield_value AND &B00000001

    ' Enable low then high
    PIN(30) = 0 : PAUSE 1
    PIN(30) = 1 : PAUSE 1

END SUB

' -----
' Write 8-bit character
' -----
SUB LCDShield_Character( LCDShield_value )
    ' Character
    PIN(29) = 1

    ' Enable high
    PIN(30) = 1 : PAUSE 1

    ' Value (high nibble) onto data lines

```

```

PIN(28) = LCDShield_value AND &B10000000
PIN(27) = LCDShield_value AND &B01000000
PIN(26) = LCDShield_value AND &B00100000
PIN(25) = LCDShield_value AND &B00010000

' Enable low then high
PIN(30) = 0 : PAUSE 1
PIN(30) = 1 : PAUSE 1

' Value (low nibble) onto data lines
PIN(28) = LCDShield_value AND &B00001000
PIN(27) = LCDShield_value AND &B00000100
PIN(26) = LCDShield_value AND &B00000010
PIN(25) = LCDShield_value AND &B00000001

' Enable low then high
PIN(30) = 0 : PAUSE 1
PIN(30) = 1 : PAUSE 1

END SUB

' -----
' Write string
' -----
SUB LCDShield_String( LCDShield_string$ )
    FOR LCDShield_loop = 1 to LEN(LCDShield_string$)
        LCDShield_Character( ASC(MID$(LCDShield_string$, LCDShield_loop )))
    NEXT
END SUB

' -----
' Return button state
' 0 = no button
' 1 = right, 2 = up, 3 = down, 4 = left
' Because A/D on CGCOLORMAX is 3.3V max and this is a
' 5V resistor chain the SEL button does not work.
' -----
FUNCTION LCDShield_Button( )
    ' Button Input line
    SETPIN 35, 1

```

```
LCDShield_Button = 0
IF PIN(35) < 2.85 THEN
    LCDShield_Button = 4
ENDIF
IF PIN(35) < 1.95 THEN
    LCDShield_Button = 3
ENDIF
IF PIN(35) < 1.1 THEN
    LCDShield_Button = 2
ENDIF
IF PIN(35) < .32 THEN
    LCDShield_Button = 1
ENDIF
END FUNCTION
```

Project #12

CGKEYCHIP1 Custom Keypad Example

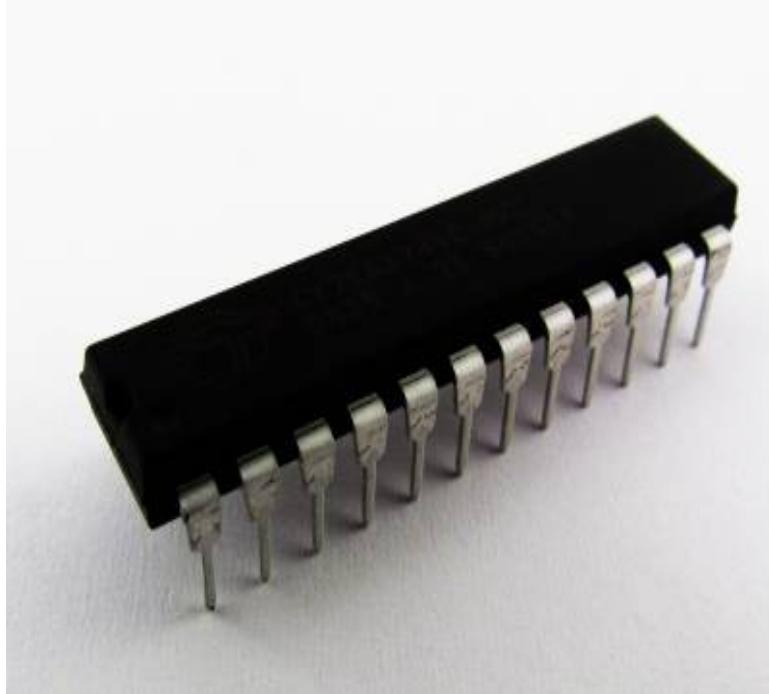


Figure 74: CGKEYCHIP1

The Maximite design uses a PS/2 connection as a keyboard interface. You can use a full keyboard connected to either a CGCOLORMAX or the CGMMSTICK.

If you want to embed either of these two setups into a project, you might choose to replace the full keyboard with something either more compact or custom for your application. The CGKEYCHIP1 wired into a "ColorMax" or "stick" lets you create your own keyboard by adding your own buttons.

The CGKEYCHIP1 is a 24 pin chip in a DIP (through-hole) package. Four wires connect to the Maximite that include ground, power, and the two PS/2 communication signals (data/clock). You can wire in twelve distinct buttons for the keyboard keys plus three "modifier" buttons.

The CGKEYCHIP1 has two sets of keys that it can transmit:

Set1	Set2
F1	Up Arrow
F2	Down Arrow
F3	Left Arrow
F4	Right Arrow
F5	Space Bar
F6	Page Up
F7	Page Down
F8	Enter
F9	End
F10	Home
F11	Tab
F12	Backspace

An input pin on the chip selects between these sets of keys.

The CGKEYCHIP1 runs off of the 5V that the PS/2 connection provides. Both chip grounds should be connected to ground. The pins marked "NoConnect" should not be connected to anything.

Supplied with the CGKEYCHIP1 is a 1.3 kohm resistor that has to be connected from pin 11 to pin 15 for the chip to operate correctly

The PS/2 data signals are open-collector signals. When you connect the GCKEYCHIP to a CGCOLORMAX there are pull up resistors already on the ColorMax. When you connect to a CGMMSTICK and CGVGAKBD1 combination, the CGVGAKBD1 board already has pull up resistors in place for the two PS/2 lines. No additional resistors are needed in this case.

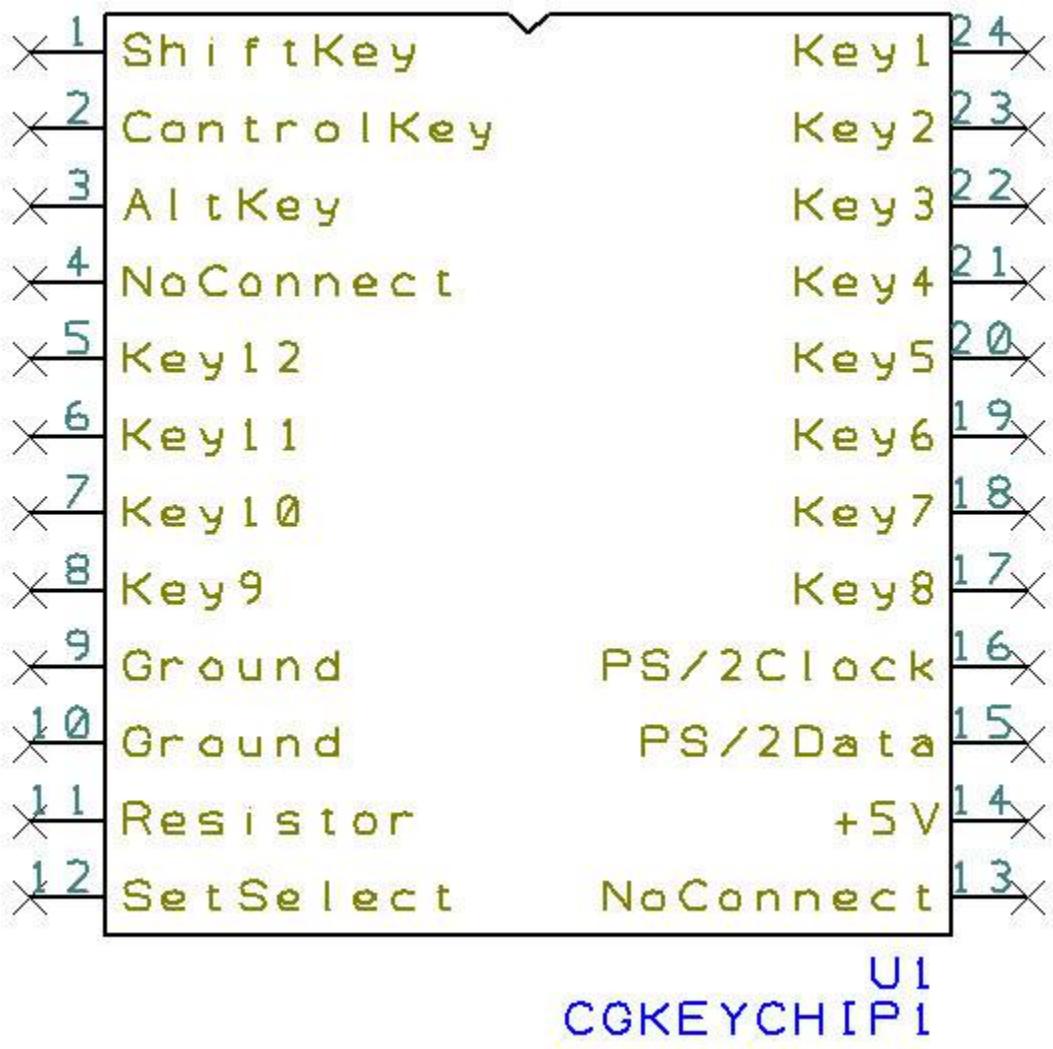


Figure 75: CGKEYCHIP1 pinout.

When you connect the CGKEYCHIP1 to just a CGMMSTICK, you will need to attach two pull-up resistors in order to have the chip function correctly. Each of the two PS/2 lines should be pulled to 5V through a 10 kohm resistor.

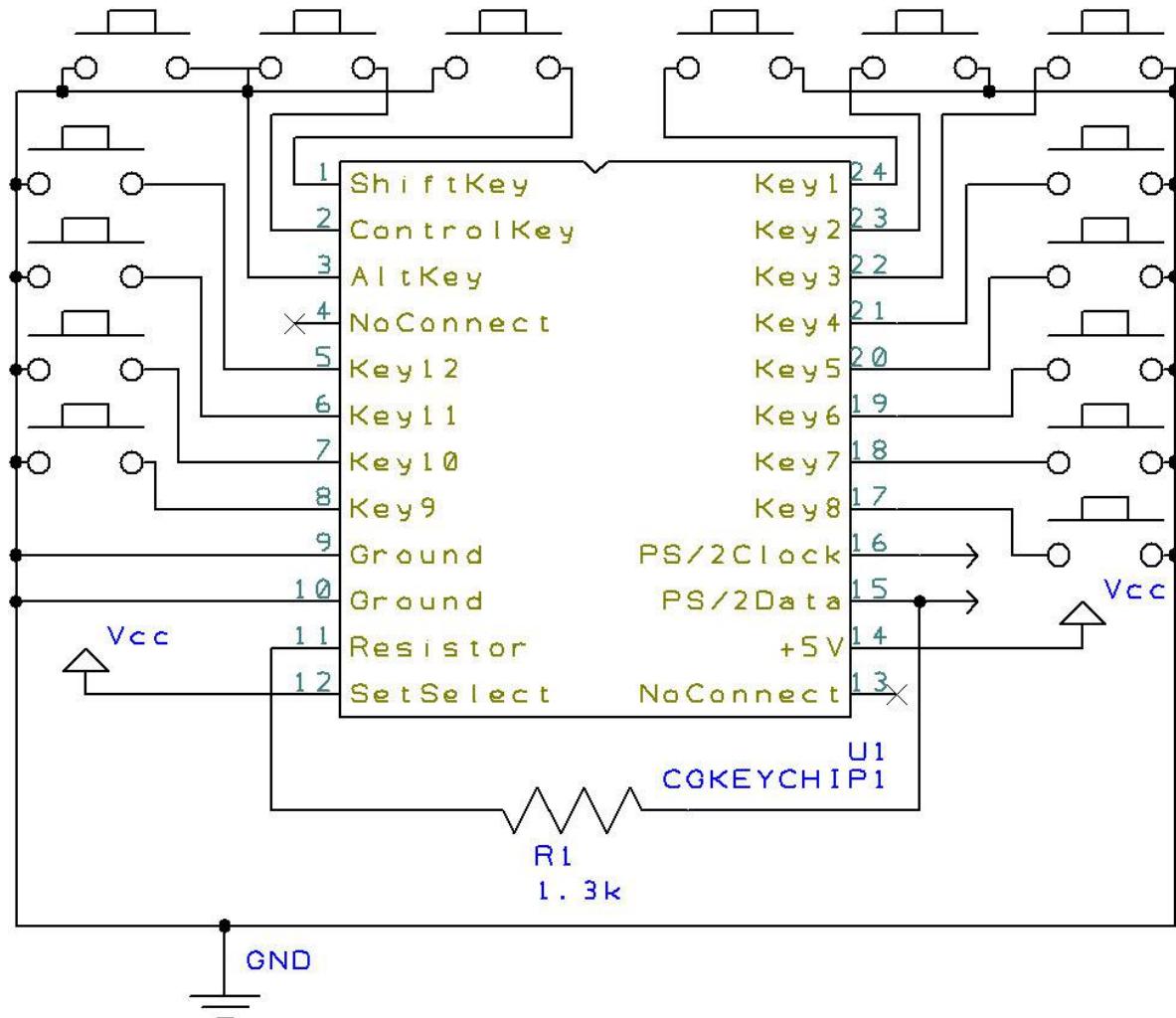


Figure 76: Example CGKEYCHIP1 schematic.

The SetSelect input to the CGKEYCHIP1 selects the key set that you wish to use, either set 1 or set 2. Connecting this input directly to 5V will select set 1, while connecting this input to ground selects set 2.

The key inputs are activated by momentary contact to ground. Simple SPST buttons can be used.

You can write a very simple MMBasic program to check the codes that the keys transmit against the list of codes found in this document:

```
do
  a$ = inkey$
  if a$ <> "" then
    print asc(a$)
  endif
loop
```

When run, set 1 will return these codes:

Key	Plain	Control	Shift	Shift+Control
F1	145	209	177	241
F2	146	210	178	242
F3	147	211	179	243
F4	148	212	180	244
F5	149	213	181	245
F6	150	214	182	246
F7	151	215	183	247
F8	152	216	184	248
F9	153	217	185	249
F10	154	218	186	250
F11	155	219	187	251
F12	156	220	188	252

Set 2 returns these codes :

Key	Plain	Control
Up Arrow	128	192
Down Arrow	129	193
Left Arrow	130	194
Right Arrow	131	195
Space Bar	32	0
Page Up	136	200
Page Down	137	201
Enter	13	13
End	135	199
Home	134	198
Tab	9	9
Backspace	8	8

MMBasic interprets the Alt as a key by itself, not a modifier. The keycode for the Alt key is 139, Control-Alt is 203, Shift-Alt is 171, Control-Shift-Alt is 235. This program will return two key codes when Alt is used, first the Alt code, then the key code. Control-Shift-Alt-F1, for example, returns 235 241.

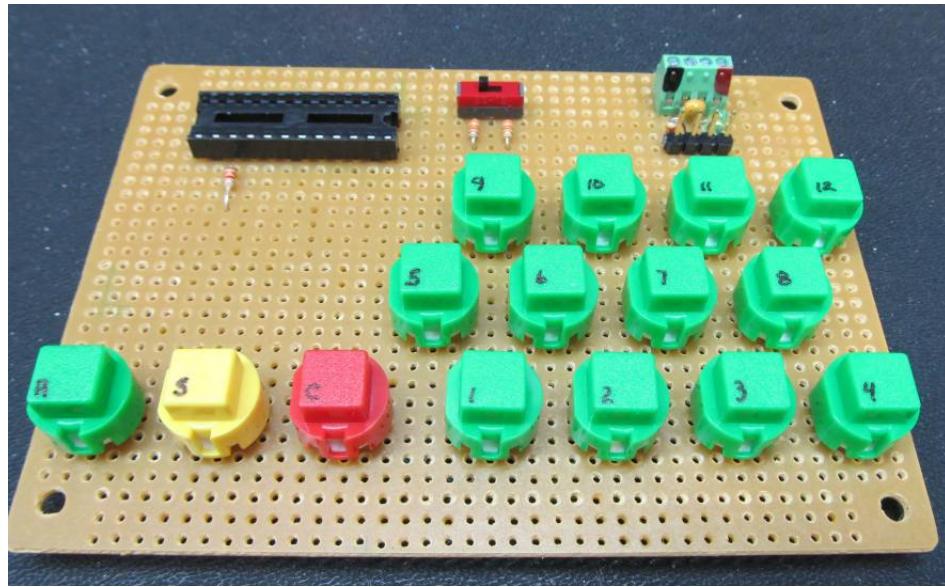


Figure 77: Example of circuit board ready for CGKEYCHIP1

See the list of Special Keyboard Keys.

CGMMSTICK1 Technical Information

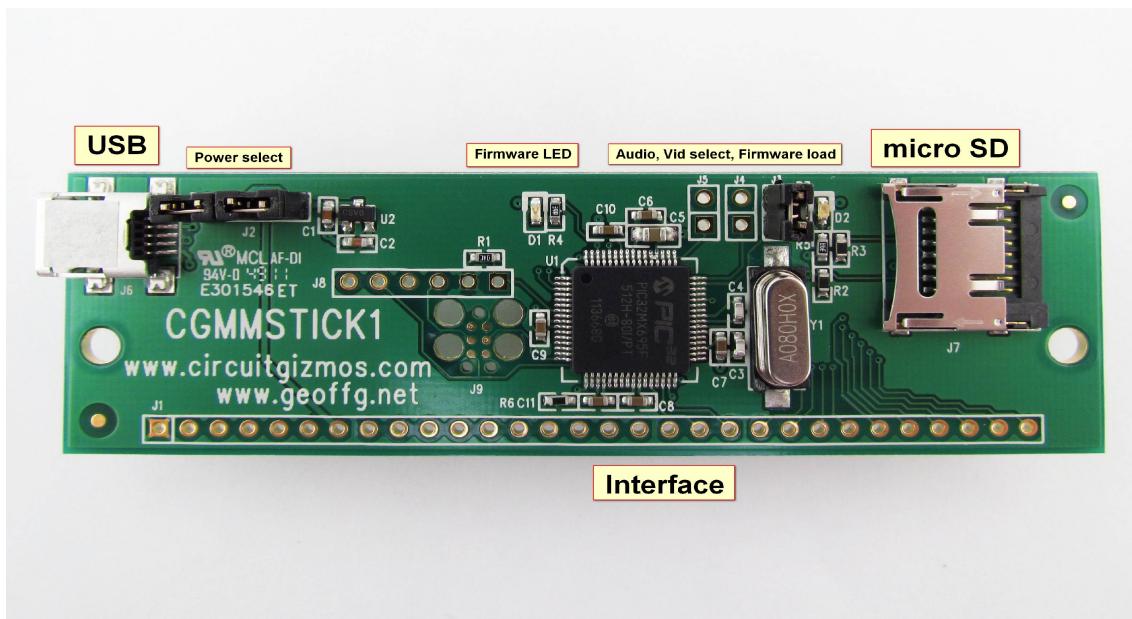


Figure 78: CGSimmstick1 connection information

The power select jumper nearest to the USB jack (left one if you hold the board so that you can read the silk-screen text) connects 5V USB to the 3.3V regulator to power the microcontroller.

The right jumper connects the 5V line on the 30-pin row to the 3.3V regulator so that J1 can power the microcontroller.

Both in place puts 5V USB onto the J1 30-pin connector.

The CGMMSTICK1 can be powered by a PC USB connection, but it can also be powered through the USB connector by using a wall-wart or car power adapter that provides +5V via a USB mini-B plug. Things such as cell phone chargers for AC (wall) use, or car cigarette lighter chargers will work if they provide enough current and a regulated +5V.

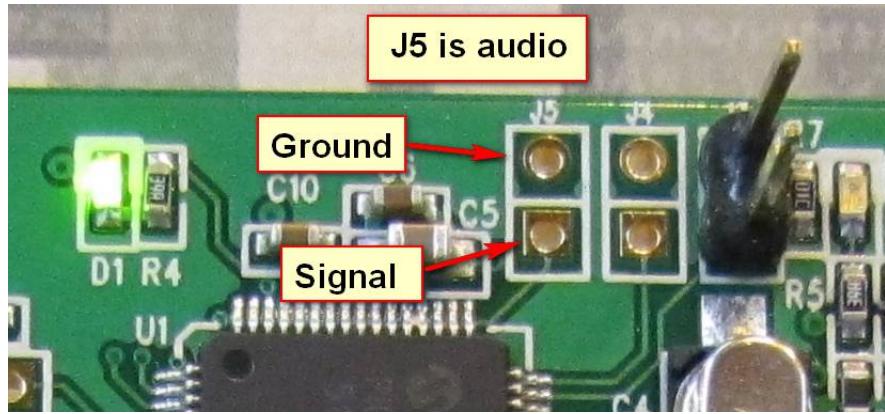


Illustration 79: Call-outs for CGMMSTICK1

J5 is audio out - square pad is signal, round is ground.

J4 is video select. Open is VGA. insert a wire link for composite video. Setting is tested on powerup.

J3 with a header in place is firmware reload. Put the header in place and then power up the CGMMSTICK. The bootloader will then run.

J8 and J9 are PIC32 programming headers used by the factory.

D1 – Green “Firmware” LED.

D2 – Blue SD card access LED.

SD card insertion (CGMMSTICK1)

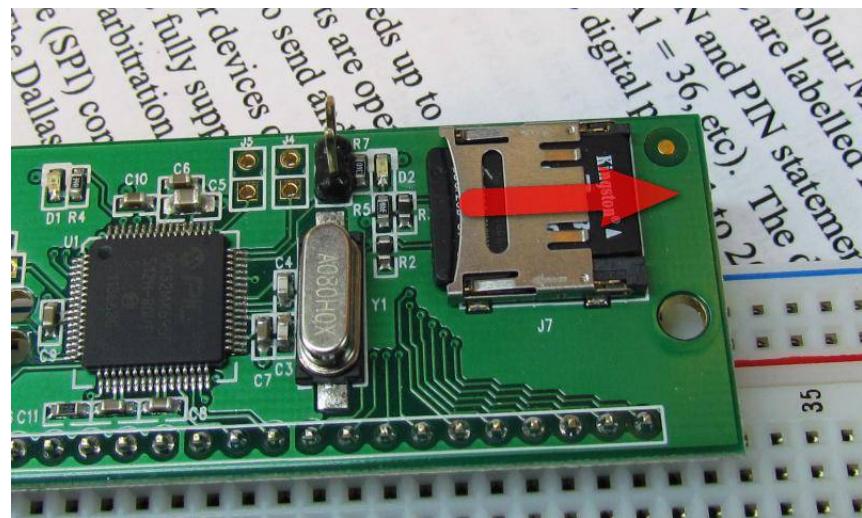


Figure 80: Slide the top of the shell outward.

The top of the SD card holder shell slides to the side slightly to open.



Illustration 81: The shell flips open.

The micro SD card connector has a little clam shell lid. It needs to be slid to one side to open, then back to lock closed.

The photos above show opening the micro SD clam shell.

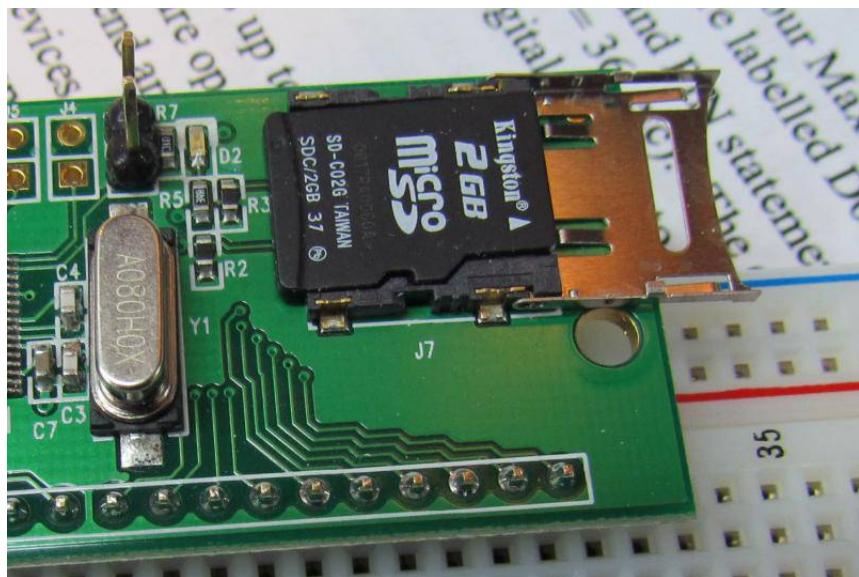


Figure 82: Insert or remove the micro SD card.

SD card insertion is simple.

SD card removal (CGMMSTICK1)



Figure 83: Close the shell.

The cover for the SD card holder flips closed.

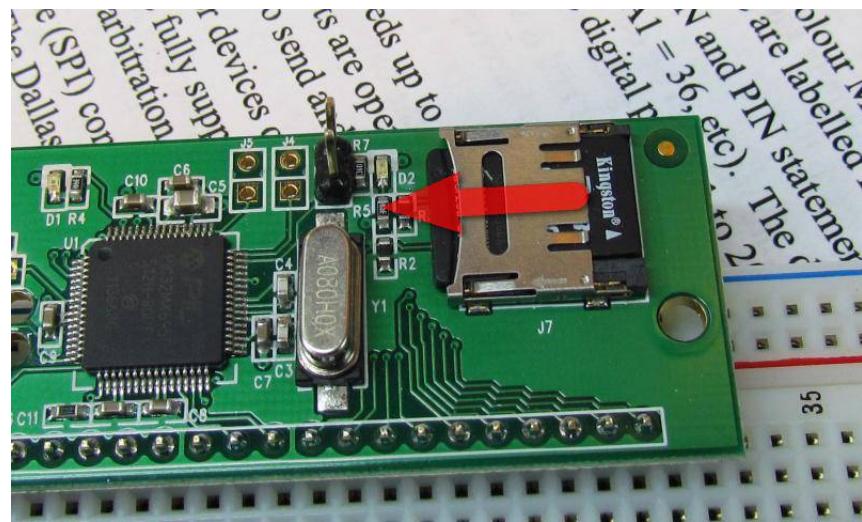


Figure 84: Slide the top of the shell inward.

The photos above show closing the micro SD clam shell.

J1 Pinout (CGMMSTICK1)

J1 Pin	Function	Analog	Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	Additional Function
1	Reg 3.3V									
2	Video1									
3	Video2									
4	NC									
5	VSYNC									
6	KBDDATA									
7	+5V									
8	KBDCLK									
9	GROUND									
10	Hsync									
11	I/O 1	Analog In	Digital In				Digital Out			
12	I/O 2	Analog In	Digital In				Digital Out			
13	I/O 3	Analog In	Digital In				Digital Out			
14	I/O 4	Analog In	Digital In				Digital Out			
15	I/O 5	Analog In	Digital In				Digital Out			
16	I/O 6	Analog In	Digital In				Digital Out			
17	I/O 7	Analog In	Digital In				Digital Out			
18	I/O 8	Analog In	Digital In				Digital Out			
19	I/O 9	Analog In	Digital In				Digital Out			
20	I/O 10	Analog In	Digital In				Digital Out			
21	I/O 11		Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	
22	I/O 12		Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	
23	I/O 13		Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	
24	I/O 14		Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	
25	I/O 15		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 1 Receive
26	I/O 16		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 1 Transmit
27	I/O 17		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 1 RTS
28	I/O 18		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 1 CTS
29	I/O 19		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 2 Receive
30	I/O 20		Digital In				Digital Out	Open Collector	5V Tolerant	Serial 2 Transmit

CGVGAKBD1 Schematic

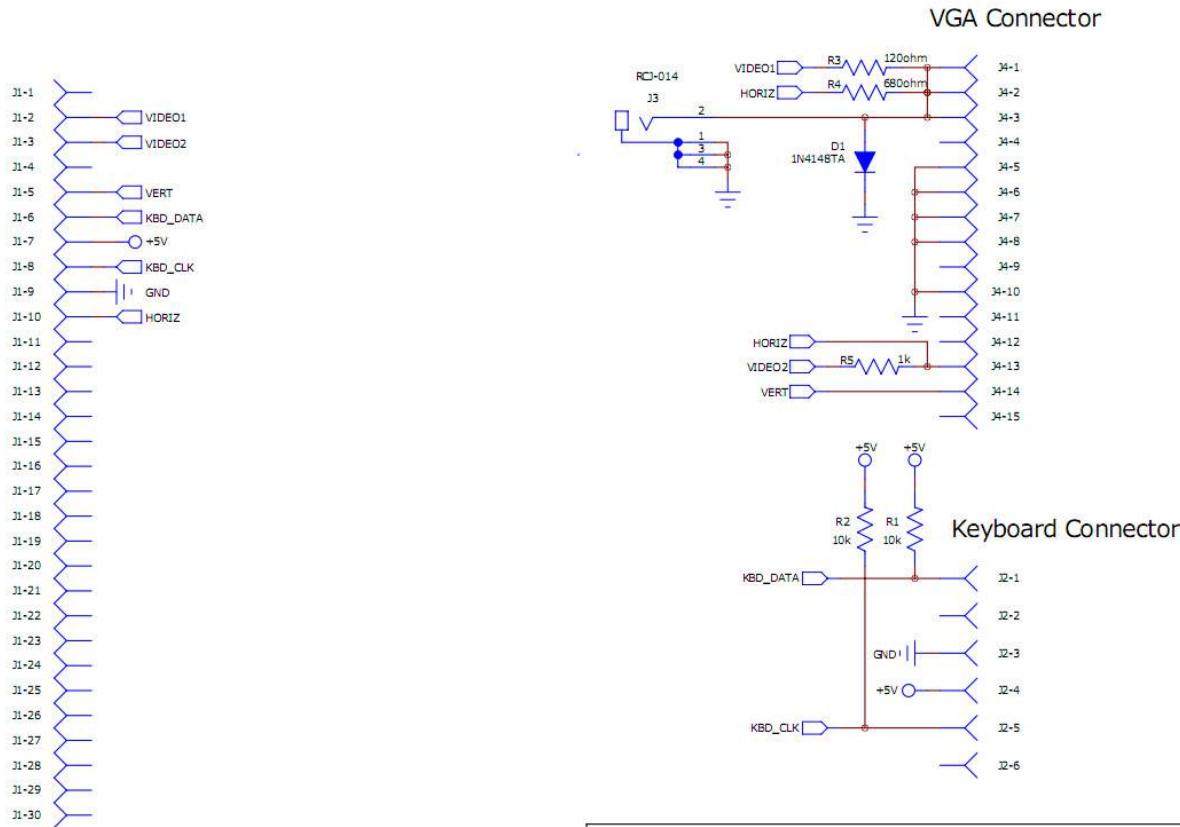


Figure 85: Connection between the CGMMSTICK1 30-pin connector and VGA, keyboard, and RCA jack (composite video).

Not shown in the schematic is J5, which parallels all of the connections of J1 from pin 11 to pin 30.

The screen resolution for composite display in PAL (default) mode is 304 horizontal, and 216 vertical. Screen resolution in NTSC is 304 horizontal, and 180 vertical.

Use CONFIG COMPOSITE to set either PAL or NTSC. NTSC is used for composite video in the United States.

Composite mode is set using a jumper (or a wire) in place on J4 on the CGMMSTICK which is sampled on power up.

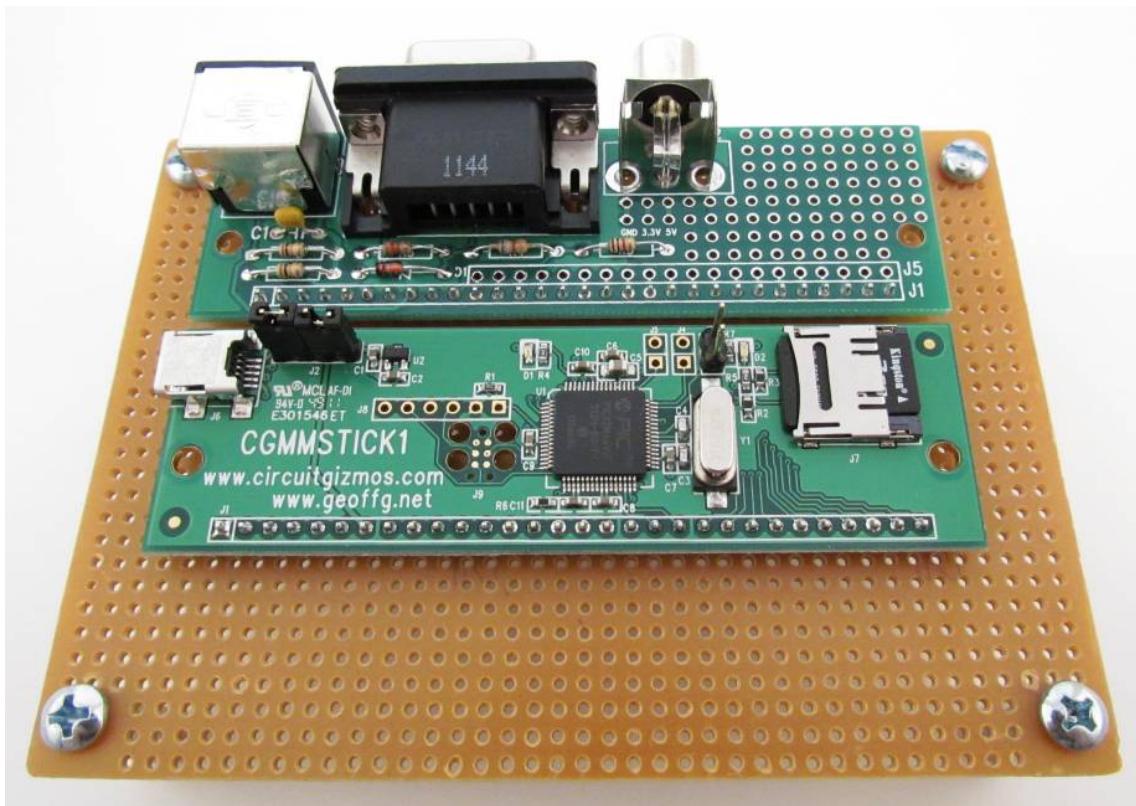


Illustration 86: Figure 67: The CGVGAKBD1 board paired up with a CGMMSTICK1 built on a wiring board.

Connections from The CGMMSTICK to CGVGAKBD board should be from pin 1 to pin 1, 2 to 2, etc.



Figure 87: A CGMMSTICK1 and CGVGAKBD1 board in action.

The CGMMSTICK and CGVGAKBD combination can be used as a stand-alone computer.

Schematic (CGMMSTICK1)

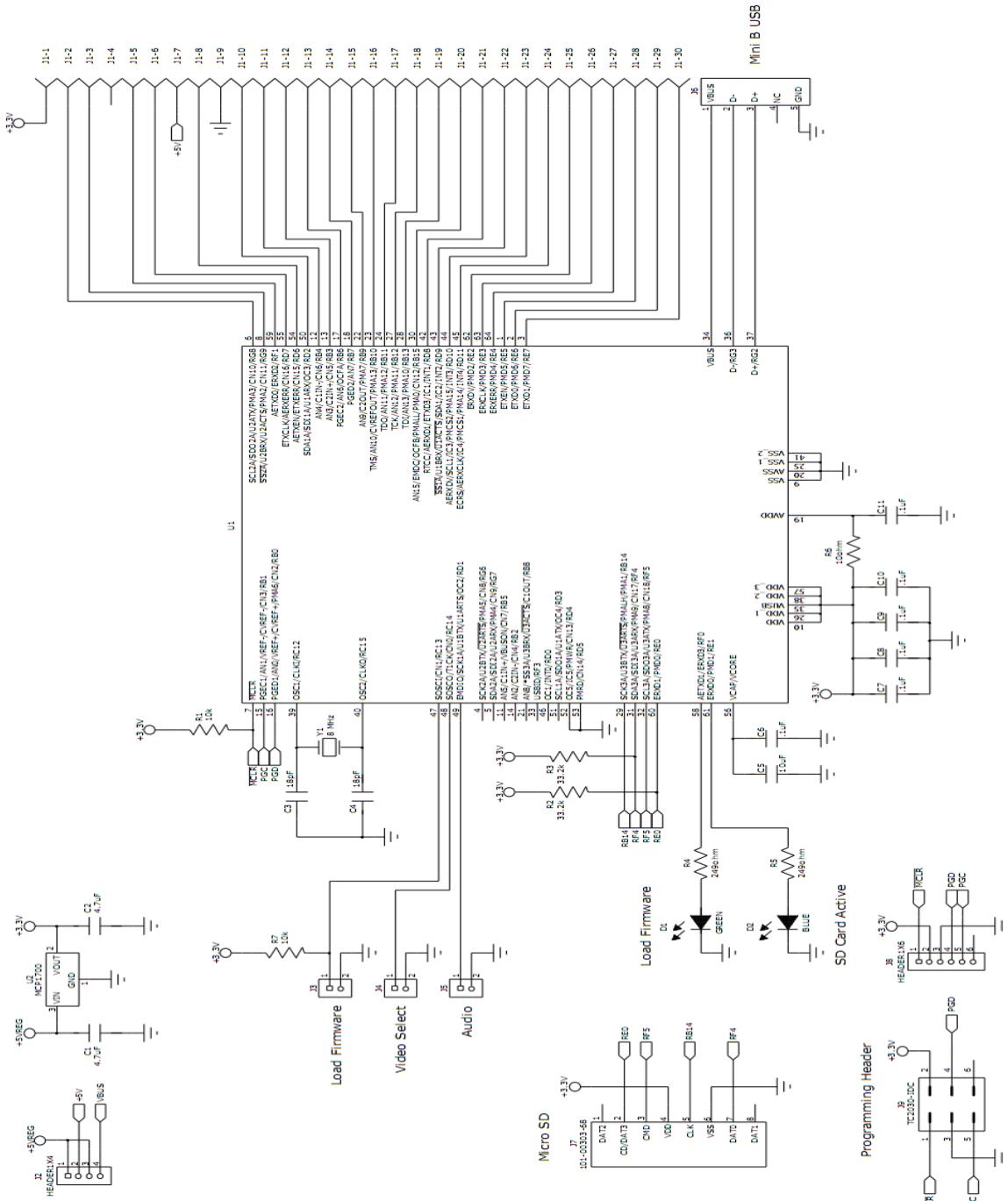


Illustration 88: CGMMSTICK schematic.

Firmware Update (CGMMSTICK1)

The CGMMSTICK1 is loaded at the factory with a bootloader and with the version of MMBasic that is current at the time of board production. New versions of MMBasic are released periodically, and the CGMMSTICK1 uses a bootloader that allows for MMBasic update.

Upgrades are done via the USB interface when the boot loader is running.

Powering up the CGMMSTICK1 will normally cause MMBasic to run. If a header is in place on J3, then on power up the bootloader will run.

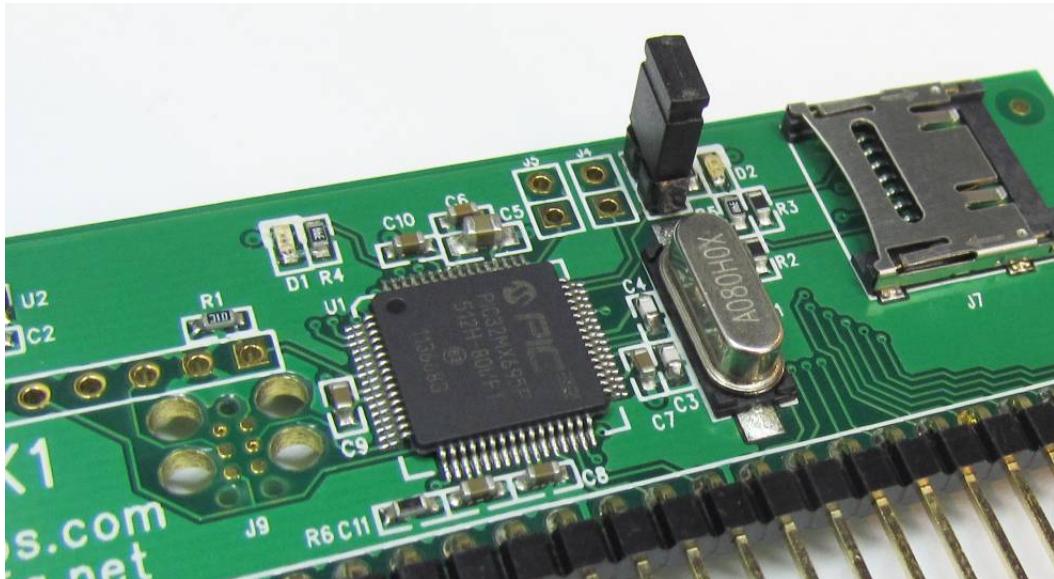


Illustration 89: The bootloader jumper in place on a CGMMSTICK.

When the bootloader is running, the green LED on the CGMMSTICK1 will flash on and off. During this time, a PC program can connect through USB to the CGMMSTICK1 for updating firmware. *Bootloader.exe* will load MMBasic onto a CGMMSTICK1. The program is part of a zip file download from CircuitGizmos. *Bootloader.exe* runs as a stand-alone program without needing installation.

Updated firmware is available from <http://geoffg.net/maximite.html>

The CGMMSTICK1 runs the firmware made for the original monochrome Maximite. Download the updated firmware zip file and un-zip it to a directory on your PC.

Firmware Upgrade Steps (CGMMSTICK1)

Start with the CGMMSTICK1 unpowered.

Have the bootloader jumper described above in place on the device while you apply power by connecting the USB cable to your PC. The green CGMMSTICK1 power LED will rapidly flash to indicate that the boot loader is in control. (At this point the bootloader is running and the jumper can be removed.)

The computer should automatically recognize the device and load the appropriate driver. (The CGMMSTICK1 will show up in the Windows Device Manager as a Human Interface Device when connected to the bootloader.)

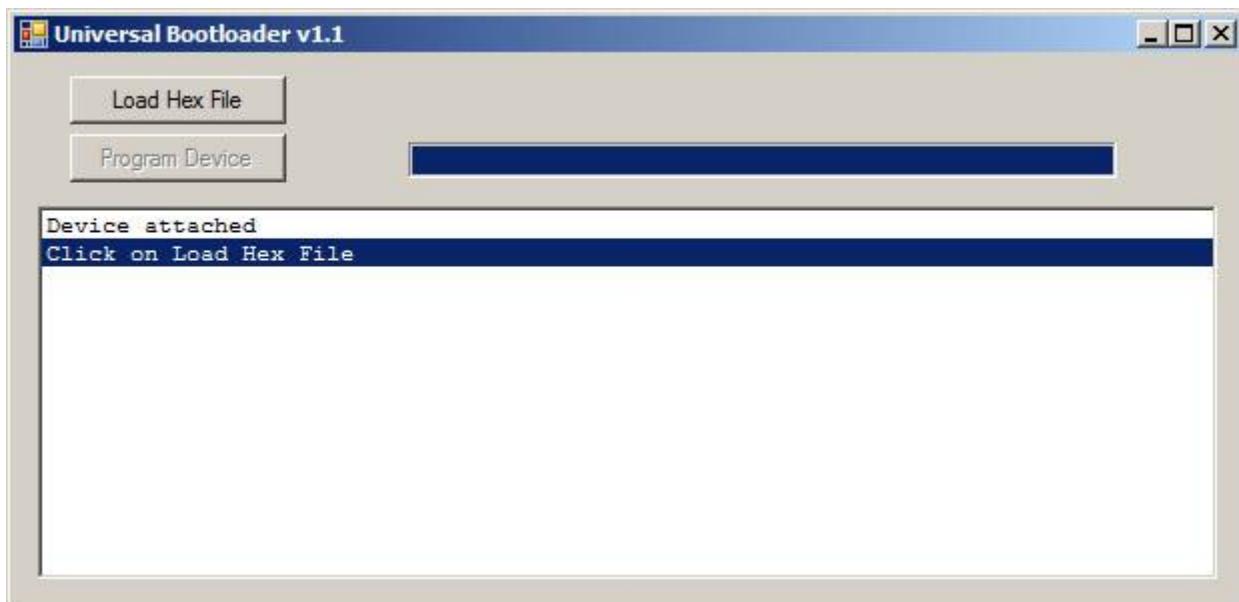


Illustration 90: Bootloader.exe has detected and connected to a CGMMSTICK/CGCOLORMAX in bootloader mode.

Run BootLoader.exe it will automatically detect the device and show the message "Device attached".

If the **Load Hex File** button in bootLoader.exe is grayed out it means that the CGMMSTICK1 is not connected or not in boot load mode. Check the USB cable and that the green CGMMSTICK1 LED is flashing.

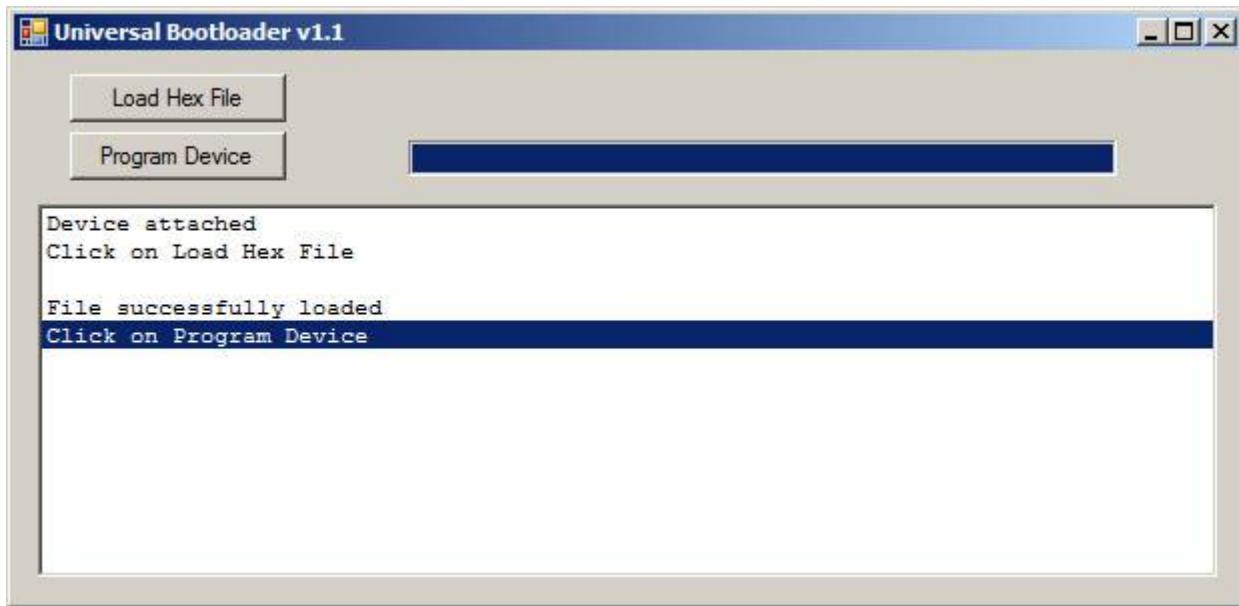


Illustration 91: Bootloader.exe with a firmware hex file loaded.

Click on the **Load Hex File** button and load the firmware upgrade file. The firmware file will have a .hex extension.

An example hex file name might be *Maximite_MMBasic_V4.3.hex*.

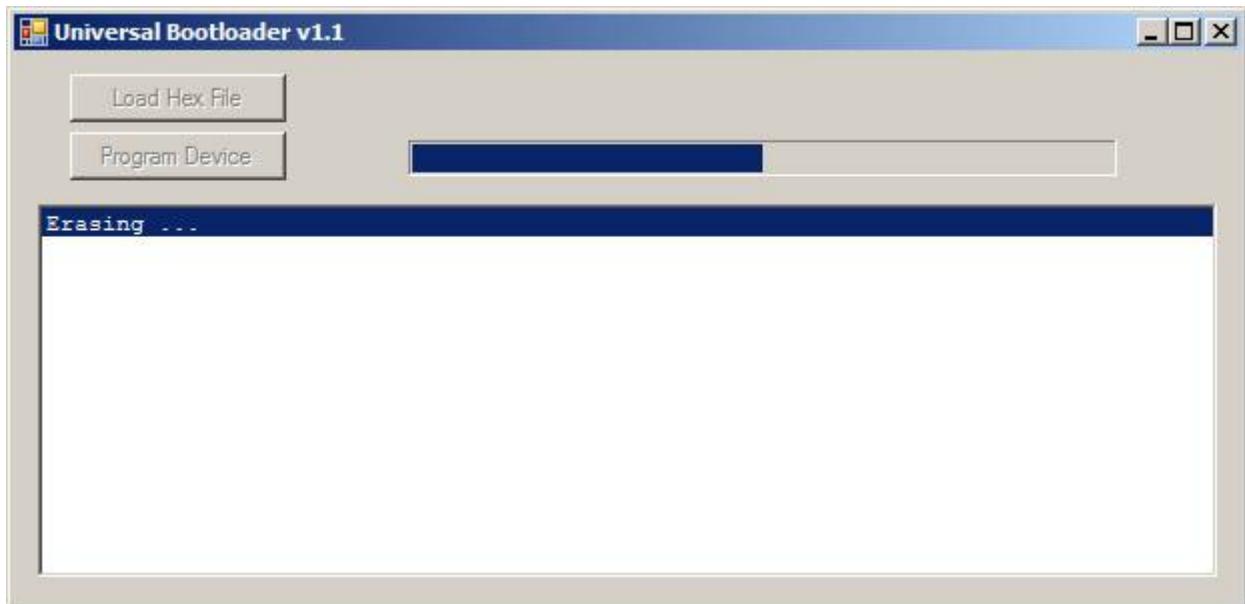


Illustration 92: Bootloader.exe erasing the CGMMSTICK/CGCOLORMAX firmware.

After the firmware hex file is loaded into bootloader.exe, press the **Program Device** button.

Bootloader.exe will erase the old firmware first. This isn't a full erase of the CGMMSTICK1 chip, as a full erase would also erase the bootloader and that needs to stay on the chip. Everything BUT the bootloader is erased, which means that the contents of the on-chip A: drive is also completely cleared.

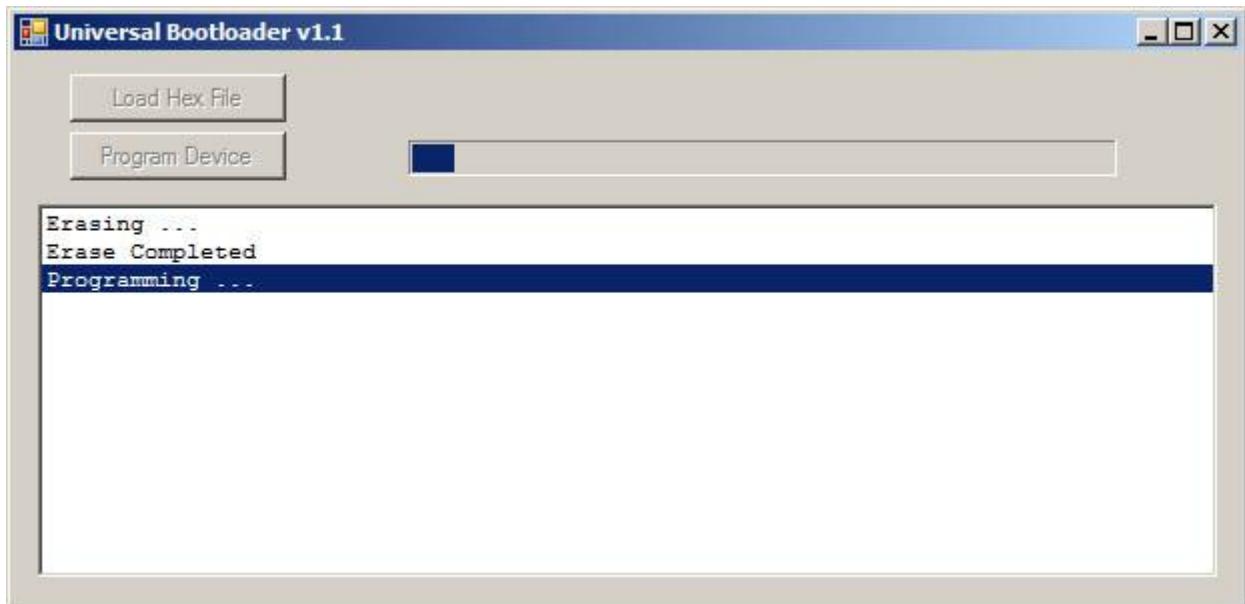


Illustration 93: After erasing, bootloader.exe will program the new firmware to the CGMMSTICK/CGCOLORMAX.

Once erased, bootloader will program the new firmware to the CGMMSTICK1. There is a progress bar that shows the progress of this programming operation. Immediately after programming bootloader.exe will double-check to make sure that the firmware was correctly loaded with a verification step.

The entire programming and verification process should take only about a minute.

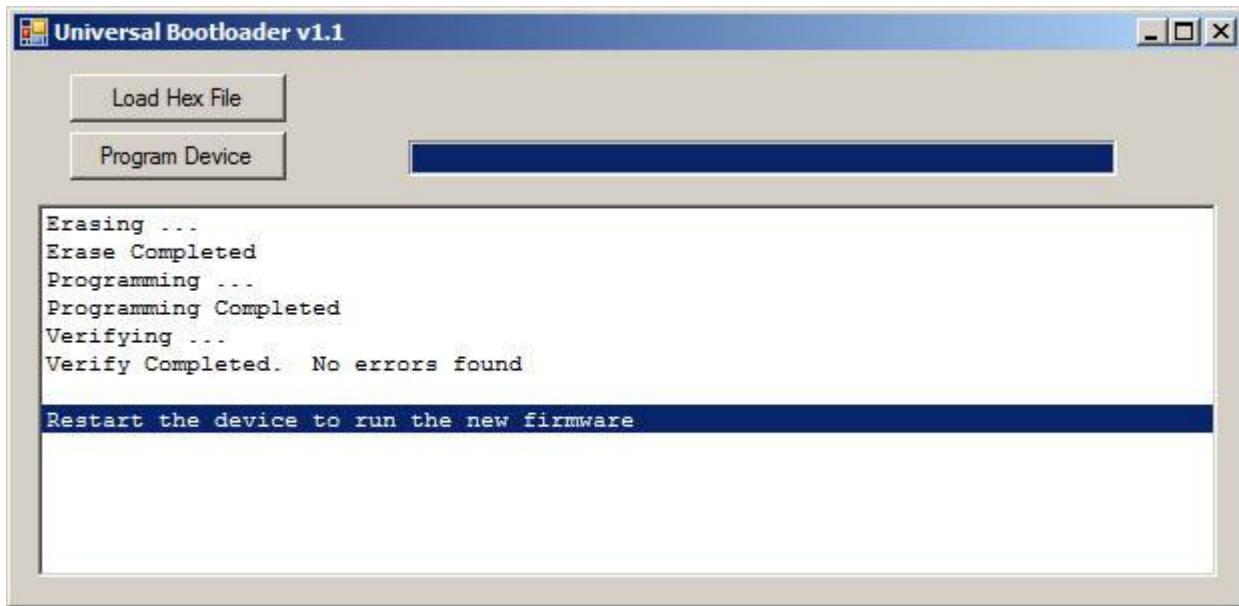


Illustration 94: Programming finished.

After verification the green CGMMSTICK1 power LED will flash slowly. This indicates that the new firmware on the CGMMSTICK1 is completely programmed.

Remove the power connection from the CGMMSTICK1. Make sure that the J3 jumper has also been removed.

Power the CGMMSTICK1 again and the updated MMBasic firmware will run.

CGCOLORMAX1 Technical Information

Original version of the COLORMAX.

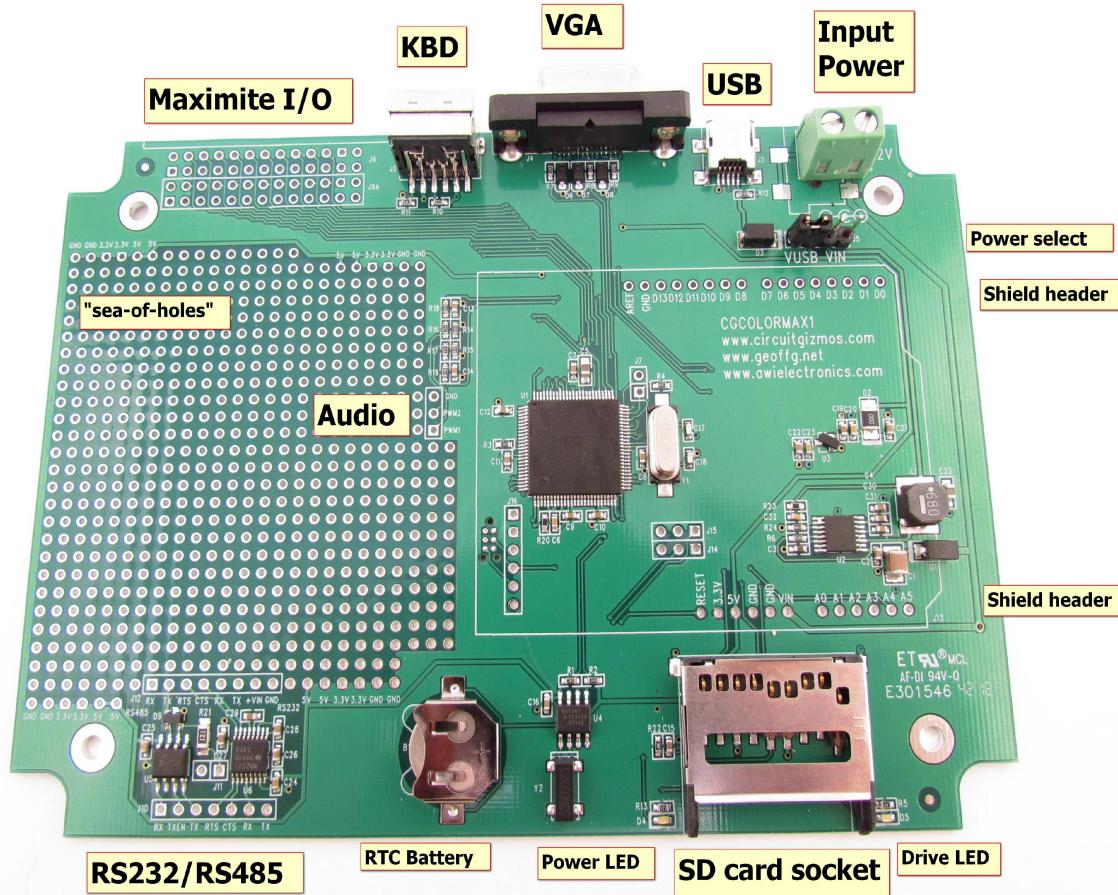


Figure 95: CGCOLORMAX1 rev 1 I/O callouts.

Power Supply (CGCOLORMAX1)

The power select jumper (J5) near to the USB jack and to input power selects powering the board from either the USB connector or from the input power jack. Place a header on VUSB to power the board from USB. Move the header to VIN to power the board from the input power connector.

The USB connection provides 5V to the board. The USB hub that you use with this board should supply at least 250mA. Typical powered hubs provide 500mA. 3.3V is regulated internally to power the internal circuitry.

Input voltage (J6) is 8-18VDC. The power supply should be rated at being able to supply a minimum of 250 millamps. 5V is regulated from this, and 3.3V is then regulated from 5V.

The 5V regulator can regulate 700mA for on-board circuits. The 3.3V supply can regulate 200mA (subtracted from the 5V 700mA). Existing 3.3V circuits consume 150mA on the 3.3V line. This leaves 3.3V 50mA, and 5V 500mA for user circuitry.



Illustration 96: Barrel jack and screw terminal.

J6 can be populated with either the supplied screw terminal, or the 5.5mm/2.1mm barrel jack. The screw terminal screw nearest to the USB jack is ground, the screw nearest to the edge of the board is positive. The barrel connection is center positive.

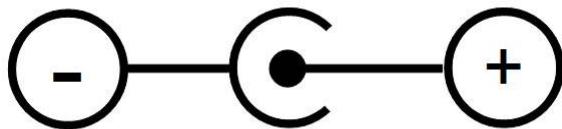


Illustration 97: Center positive barrel jack symbol.

The CGCOLORMAX1 can be powered by a PC USB connection, but it can also be powered through the USB connector by using a wall-wart or car power adapter that provides +5V via a USB mini-B plug. Things such as cell phone chargers for AC (wall) use, or car cigarette lighter chargers will work if they provide enough current and a regulated +5V.

Rear Connector (CGCOLORMAX1)

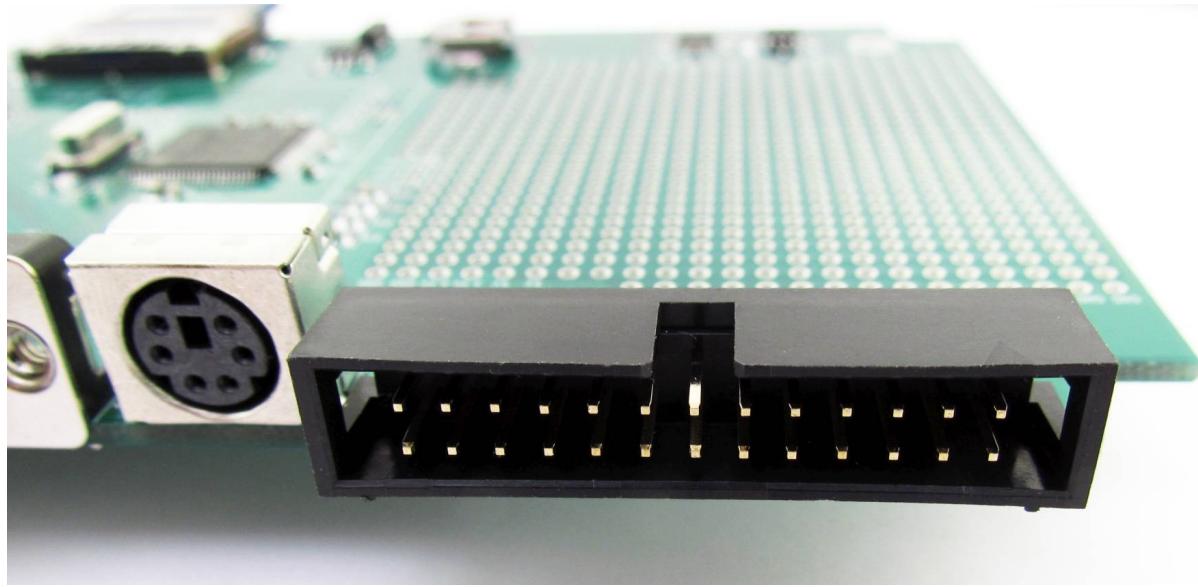


Illustration 98: View of 2x13 "Maximite connector" from back of board. Pin 1 is in the upper right, pin 2 is below it.

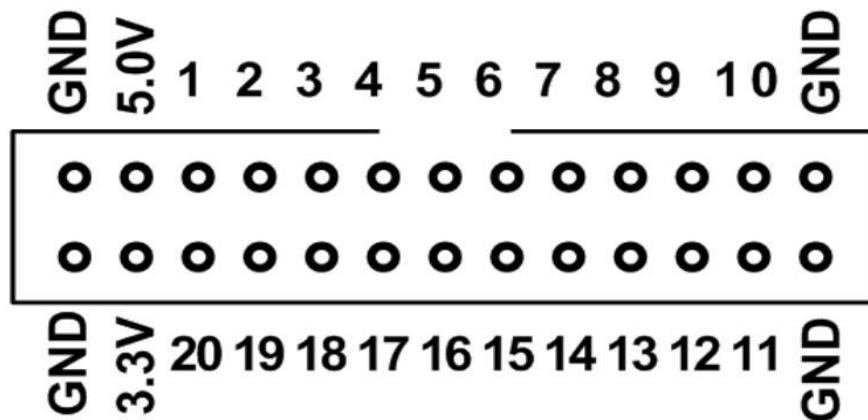
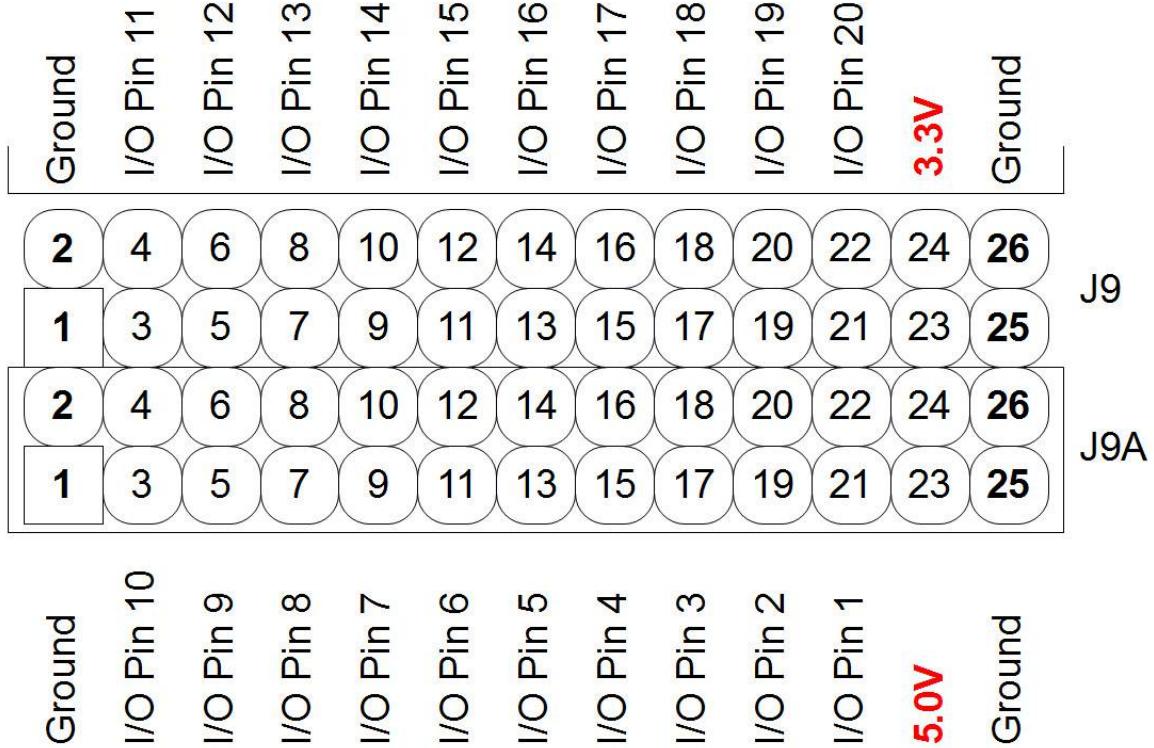


Figure 99: View of 2x13 "Maximite connector" from back of board. This is J9 for the CGCOLORMAX1. Pin 1 is the GND in the upper right, pin 2 is the GND below it.



Top view of J9/J9A for CGCOLORMAX1

Figure 100: View of the unpopulated J9/J9A connector from the top of the board.

J9 (duplicated on J9A)– Maximite I/O Connector:

Connector Pin	Use
1	Ground
2	Ground
3	I/O Pin 10
4	I/O Pin 11
5	I/O Pin 9
6	I/O Pin 12
7	I/O Pin 8
8	I/O Pin 13
9	I/O Pin 7
10	I/O Pin 14
11	I/O Pin 6
12	I/O Pin 15
13	I/O Pin 5
14	I/O Pin 16
15	I/O Pin 4
16	I/O Pin 17
17	I/O Pin 3
18	I/O Pin 18
19	I/O Pin 2
20	I/O Pin 19
21	I/O Pin 1
22	I/O Pin 20
23	5.0V
24	3.3V
25	Ground
26	Ground

Shield Connections (CGCOLORMAX1)

Arduino Shield footprint J13	
Shield Pins	Pin from software perspective
D0	I/O Pin 21
D1	I/O Pin 22
D2	I/O Pin 23
D3	I/O Pin 24
D4	I/O Pin 25
D5	I/O Pin 26
D6	I/O Pin 27
D7	I/O Pin 28
D8	I/O Pin 29
D9	I/O Pin 30
D10	I/O Pin 31
D11	I/O Pin 32
D12	I/O Pin 33
D13	I/O Pin 34
A0	I/O Pin 35
A1	I/O Pin 36
A2	I/O Pin 37
A3	I/O Pin 38
A4	I/O Pin 39 (see J15 information)
A5	I/O Pin 40 (see J14 information)

J15 – Shield A4 connection. Connect pins 1-2 to have an I2C connection to SDA on the programming header. Connect 2-3 to connect A4 to I/O Pin 12.

J14 – Shield A5 connection. Connect pins 1-2 to have an I2C connection to SCL on the programming header. Connect 1-3 to connect A5 to I/O Pin 13.

A4/A5 on the Shield connection lead to J15/J14 respectively. A4/A5 can be analog inputs or I2C connections depending on how you may want to use them for the shields that you choose. They are not connected to anything to begin with on the CGCOLORMAX1.

If you wish to use A4/A5 for the I2C interface that MMBasic supports, you will want to connect A4 to I/O Pin 12 (J15 pins 2-3) and connect A5 to I/O Pin 13 (J14 pins 1-3).

If you wish to use A4/A5 for the analog interface that MMBasic supports, you will want to connect J15 pins 1-2 and connect J14 pins 1-2.

Interface Circuits (CGCOLORMAX1)

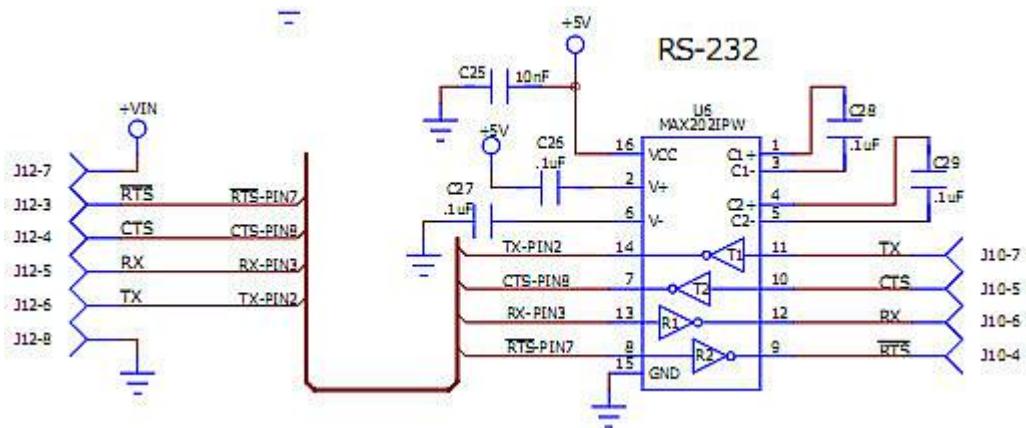


Illustration 101: The RS232 circuit of the CGCOLORMAX1. The driver chip is populated on the CGCOLORMAX1.

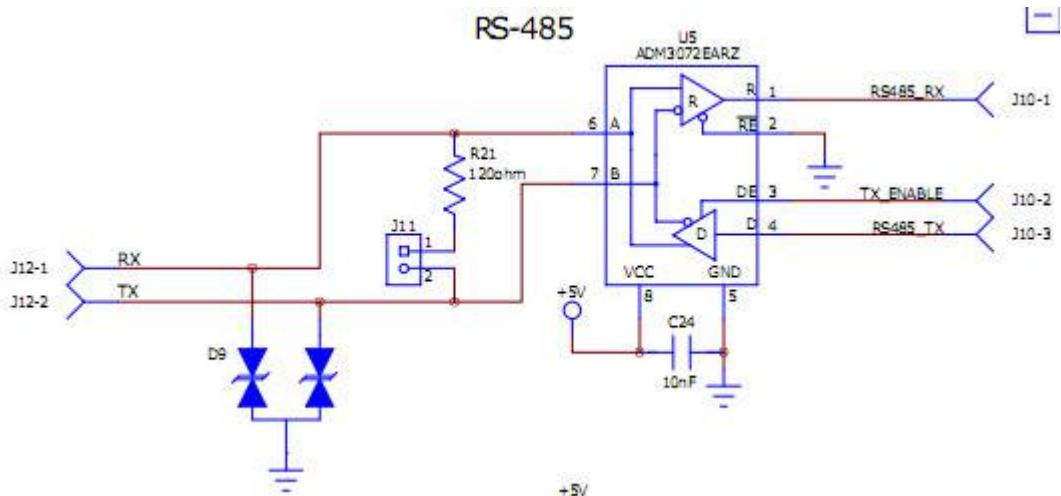


Illustration 102: The RS485 circuit of the CGCOLORMAX1. The driver chip is populated on the CGCOLORMAX1.

J10 – RS232/RS485 (CGCOLORMAX1 connection side)

<i>Pin</i>	<i>Use</i>
1	RS485 Rx (to CGCOLORMAX1)
2	RS485 Drive enable (from CGCOLORMAX1)
3	RS485 Tx (from CGCOLORMAX1)
4	RS232 RTS (to CGCOLORMAX1)
5	RS232 Rx (to CGCOLORMAX1)
6	RS232 CTS (from CGCOLORMAX1)
7	RS232 Tx (from CGCOLORMAX1)

J11 – A jumper here will enable RS485 line termination.

J12 – RS232/RS485 (line/cable side)

<i>Pin</i>	<i>Use</i>
1	RS485 RX (Should be labeled 'RS485 A')
2	RS485 TX (Should be labeled 'RS485 B')
3	RS232 RTS
4	RS232 CTS
5	RS232 Rx
6	RS232 TX
7	+Vin (Connected to J6)
8	Ground

Misc Connections (CGCOLORMAX1)

J7 – A jumper in place is for firmware reload. The two connections can be temporarily shorted together with a short piece of solid wire while applying USB power to the board to put the board on bootloader mode.

J8 – Stereo audio (PWM) out. Pin1 is PWM1, pin2 is PWM2, pin3 is ground. (The silkscreen label for J8 is missing from the surface of the board. J8 can be found next to the sea of holes with the PWM1 and PWM2 labels.)

J16 (1x6) / J17 (2x3 pattern next to J16, no silk screen label) are PIC32 programming headers.

D4 – Red “Power/Firmware” LED.

D5 – Blue SD card access LED.

There are connections to ground, +5V, and +3.3V in the “sea-of-holes” prototype area for connecting the custom circuits to power.

CGCOLORMAX1 information: One of the SMT parts is rotated in order to work with the layout. This will be fixed on a later revision.

Some early CGCOLORMAX1 boards supported only 12V-18V DC, rather than 8V-18V DC. On the CGCOLORMAX1 board R6 is supposed to be a 120k resistor and it is incorrectly a 1k resistor. This was fixed on subsequent production runs.

Pin Outs and Function (CGCOLORMAX1)

J9 Pin / J13 Pin	Function	Analog	Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	Additional Function
21	I/O 1	Analog In	Digital In				Digital Out			
19	I/O 2	Analog In	Digital In				Digital Out			
17	I/O 3	Analog In	Digital In				Digital Out			
15	I/O 4	Analog In	Digital In				Digital Out			
13	I/O 5	Analog In	Digital In				Digital Out			
11	I/O 6	Analog In	Digital In				Digital Out			
9	I/O 7	Analog In	Digital In				Digital Out			
7	I/O 8	Analog In	Digital In				Digital Out			
5	I/O 9	Analog In	Digital In				Digital Out			
3	I/O 10	Analog In	Digital In				Digital Out			
4	I/O 11		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
6	I/O 12		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
8	I/O 13		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
10	I/O 14		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
12	I/O 15		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 Receive
14	I/O 16		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 Transmit
16	I/O 17		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 RTS
18	I/O 18		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 CTS
20	I/O 19		Digital In				Digital Out	O.C.	5V Tolerant	
22	I/O 20		Digital In				Digital Out	O.C.	5V Tolerant	
D0	I/O 21		Digital In				Digital Out	O.C.	5V Tolerant	Serial 2 Receive
D1	I/O 22		Digital In				Digital Out	O.C.	5V Tolerant	Serial 2 Transmit
D2	I/O 23		Digital In				Digital Out	O.C.	5V Tolerant	
D3	I/O 24		Digital In				Digital Out	O.C.	5V Tolerant	
D4	I/O 25		Digital In				Digital Out	O.C.	5V Tolerant	
D5	I/O 26		Digital In				Digital Out	O.C.	5V Tolerant	
D6	I/O 27		Digital In				Digital Out	O.C.	5V Tolerant	
D7	I/O 28		Digital In				Digital Out	O.C.	5V Tolerant	
D8	I/O 29		Digital In				Digital Out	O.C.	5V Tolerant	
D9	I/O 30		Digital In				Digital Out	O.C.	5V Tolerant	
D10	I/O 31		Digital In				Digital Out	O.C.	5V Tolerant	
D11	I/O 32		Digital In				Digital Out	O.C.	5V Tolerant	
D12	I/O 33		Digital In				Digital Out	O.C.	5V Tolerant	
D13	I/O 34		Digital In				Digital Out	O.C.	5V Tolerant	
A0	I/O 35	Analog In	Digital In				Digital Out			
A1	I/O 36	Analog In	Digital In				Digital Out			
A2	I/O 37	Analog In	Digital In				Digital Out			
A3	I/O 38	Analog In	Digital In				Digital Out			
A4	I/O 39	Analog In	Digital In				Digital Out			
A5	I/O 40	Analog In	Digital In				Digital Out			

Firmware Update (CGCOLORMAX1)

The CGCOLORMAX1 is loaded at the factory with a bootloader and with the version of MMBasic that is current at the time of board production. New versions of MMBasic are released periodically, and the CGCOLORMAX1 uses a bootloader that allows for MMBasic update.

Upgrades are done via the USB interface when the boot loader is running.

Powering up the CGCOLORMAX1 will normally cause MMBasic to run. If a wire jumper is in place on J7, then on power up the bootloader will run.

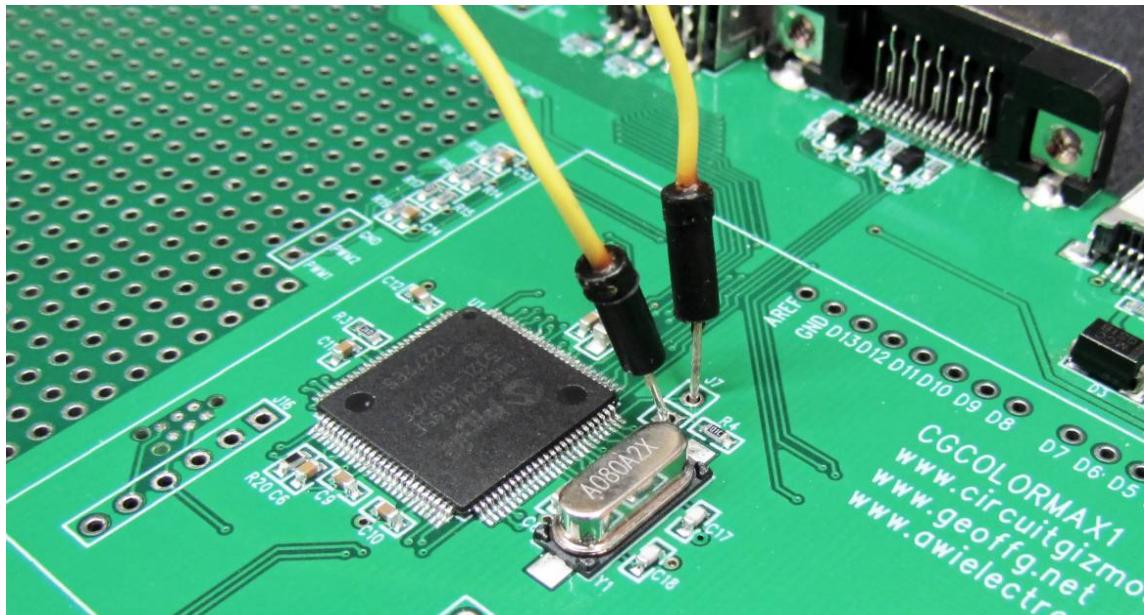


Illustration 103: A bootloader wire jumper in place on a CGCOLORMAX1.

The jumper can be any simple thin wire momentarily held in place to electrically connect both contacts of J7. Even a bare paperclip would work.

When the bootloader is running, the red LED on the CGCOLORMAX1 will flash on and off. During this time, a PC program can connect through USB to the CGCOLORMAX1 for updating firmware. *Bootloader.exe* will load MMBasic onto a CGCOLORMAX1. The program is part of a zip file download from CircuitGizmos. *Bootloader.exe* runs as a stand-alone program without needing installation.

Firmware Upgrade Steps

Start with the CGCOLORMAX1 unpowered.

Have the bootloader jumper wire described above in place on the device while you apply power by connecting the USB cable to your PC. The red CGCOLORMAX1 power LED will rapidly flash to indicate that the boot loader is in control. (At this point the bootloader is running and the jumper can be removed.)

The computer should automatically recognize the device and load the appropriate driver. (The CGCOLORMAX1 will show up in the Windows Device Manager as a Human Interface Device when connected to the bootloader.)

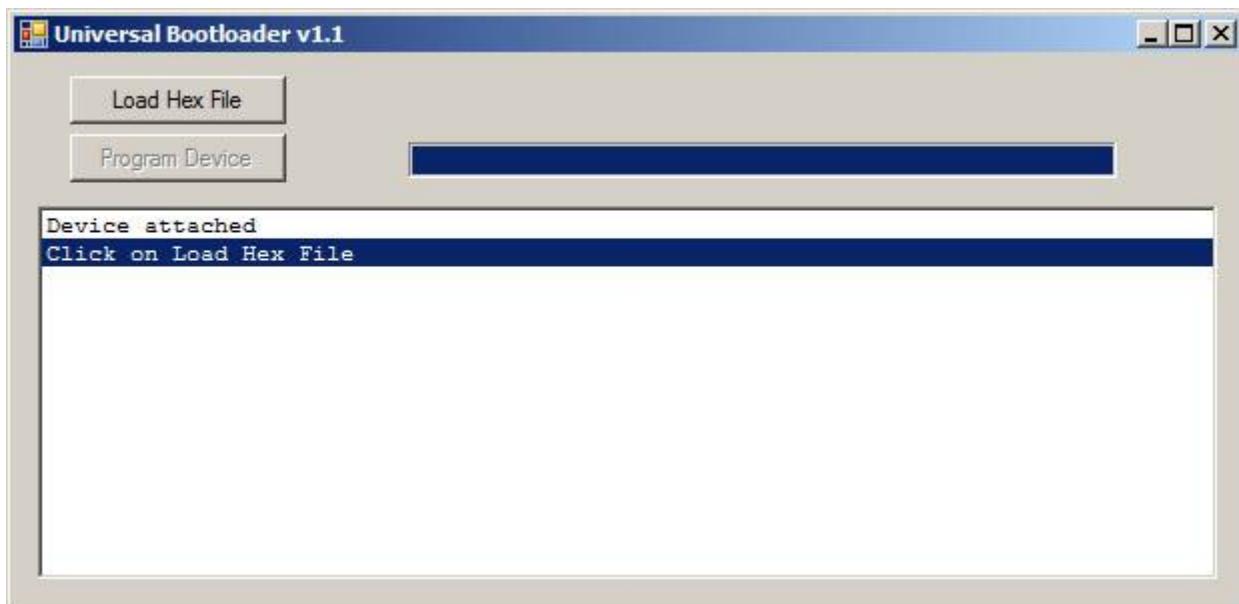


Illustration 104: Bootloader.exe has detected and connected to a CGMMSTICK/CGCOLORMAX in bootloader mode.

Run BootLoader.exe it will automatically detect the device and show the message "Device attached".

If the **Load Hex File** button in bootLoader.exe is grayed out it means that the CGMMSTICK1 is not connected or not in boot load mode. Check the USB cable and that the red CGCOLORMAX1 LED is flashing.

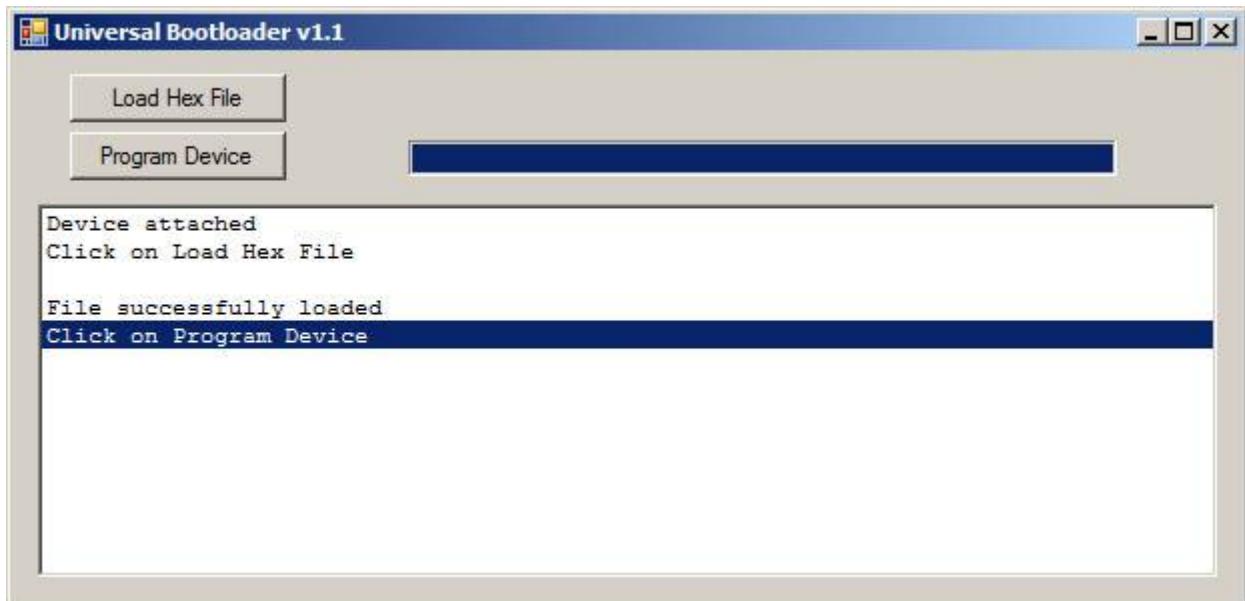


Illustration 105: Bootloader.exe with a firmware hex file loaded.

Click on the **Load Hex File** button and load the firmware upgrade file. The firmware file will have a .hex extension.

An example hex file name might be *Maximite_MMBasic_V4.3.hex*.

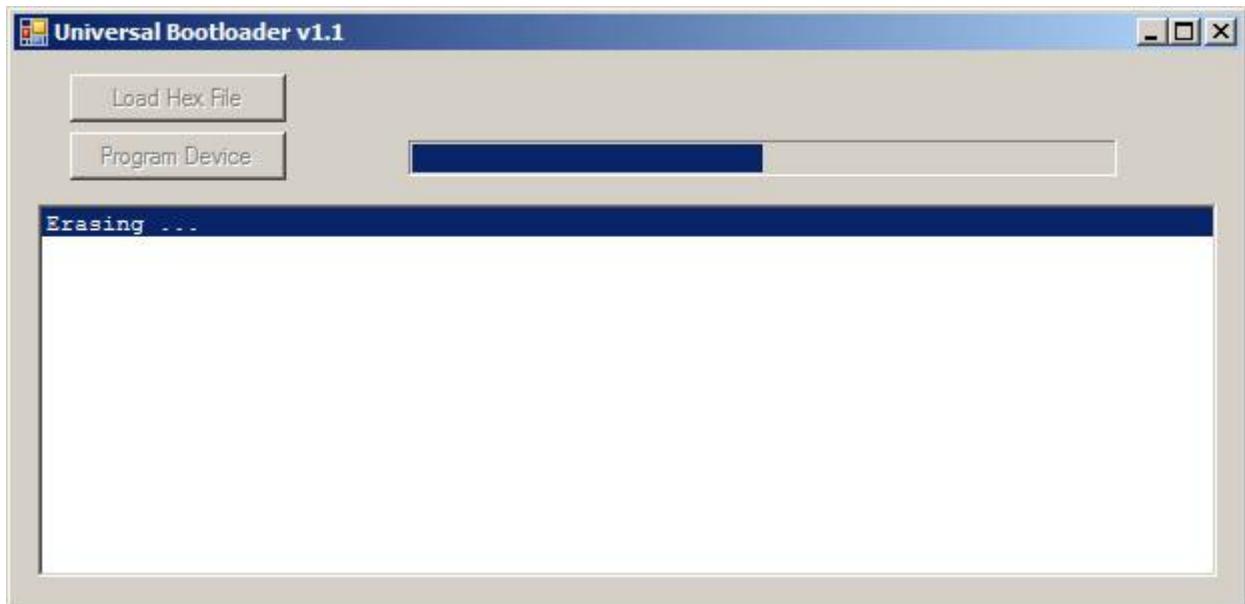


Illustration 106: Bootloader.exe erasing the CGMMSTICK/CGCOLORMAX firmware.

After the firmware hex file is loaded into bootloader.exe, press the **Program Device** button.

Bootloader.exe will erase the old firmware first. This isn't a full erase of the CGCOLORMAX1 chip, as a full erase would also erase the bootloader and that needs to stay on the chip. Everything BUT the bootloader is erased, which means that the contents of the on-chip A: drive is also completely cleared.

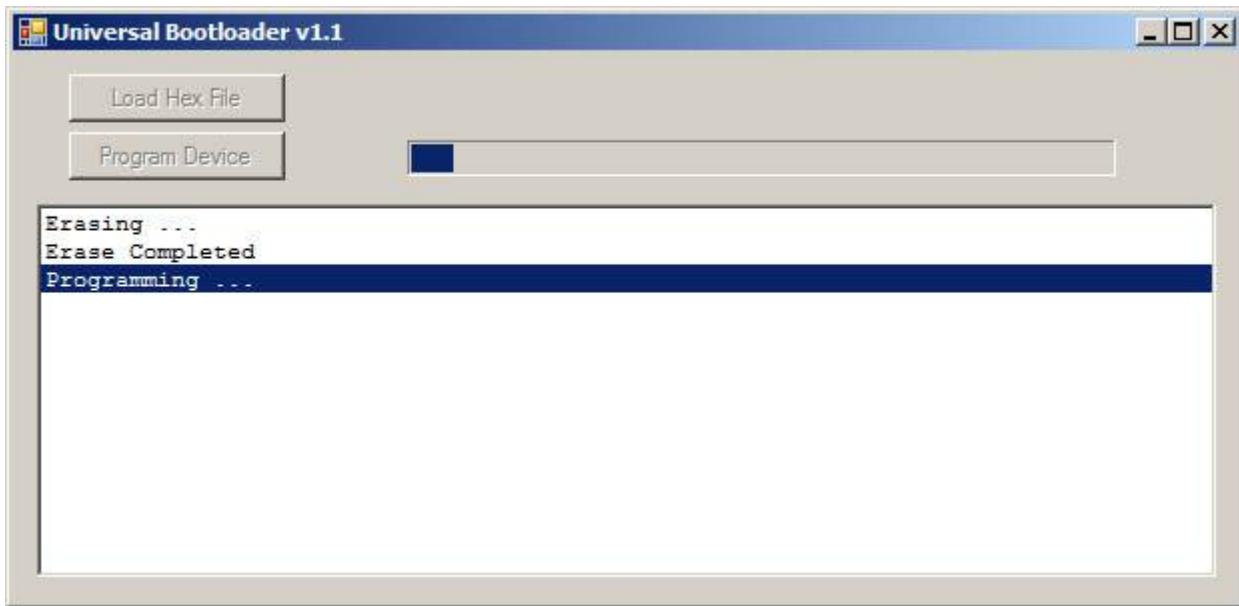


Illustration 107: After erasing, bootloader.exe will program the new firmware to the CGMMSTICK/CGCOLORMAX.

Once erased, bootloader will program the new firmware to the CGCOLORMAX1. There is a progress bar that shows the progress of this programming operation. Immediately after programming bootloader.exe will double-check to make sure that the firmware was correctly loaded with a verification step.

The entire programming and verification process should take only about a minute.

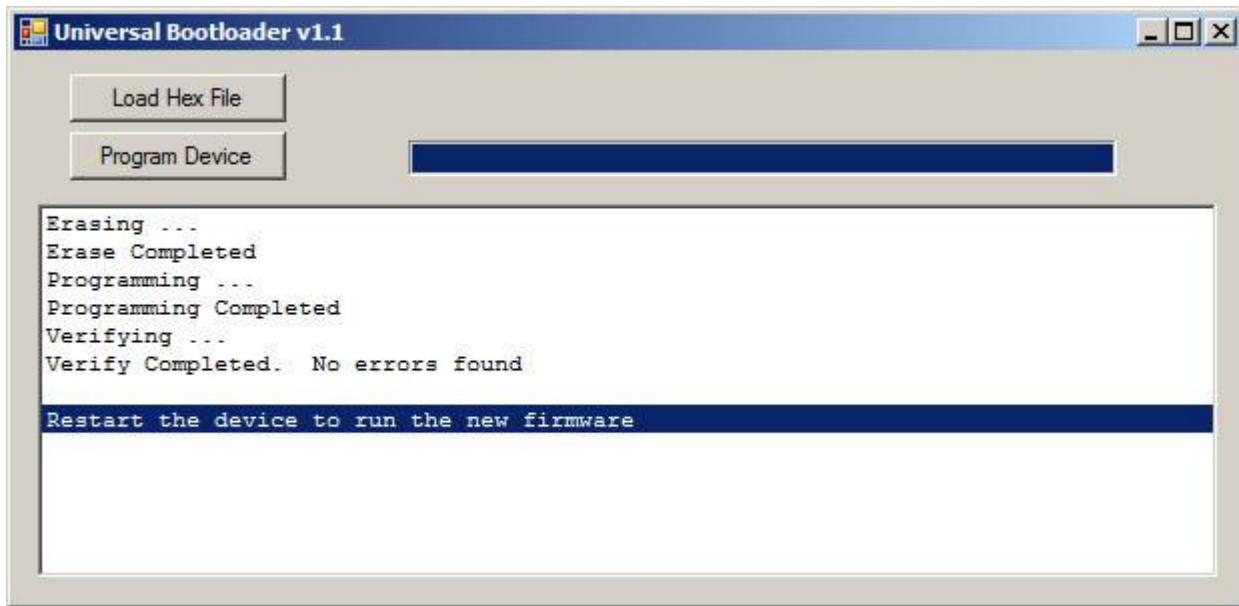


Illustration 108: Programming finished.

After verification the red CGCOLORMAX1 power LED will flash slowly. This indicates that the new firmware on the CGCOLORMAX1 is completely programmed.

Remove the power connection from the CGCOLORMAX1. Make sure that the J7 jumper wire has also been removed.

Power the CGCOLORMAX1 again and the updated MMBasic firmware will run.

CGCOLORMAX2 Technical Information

Newer version of the COLORMAX.

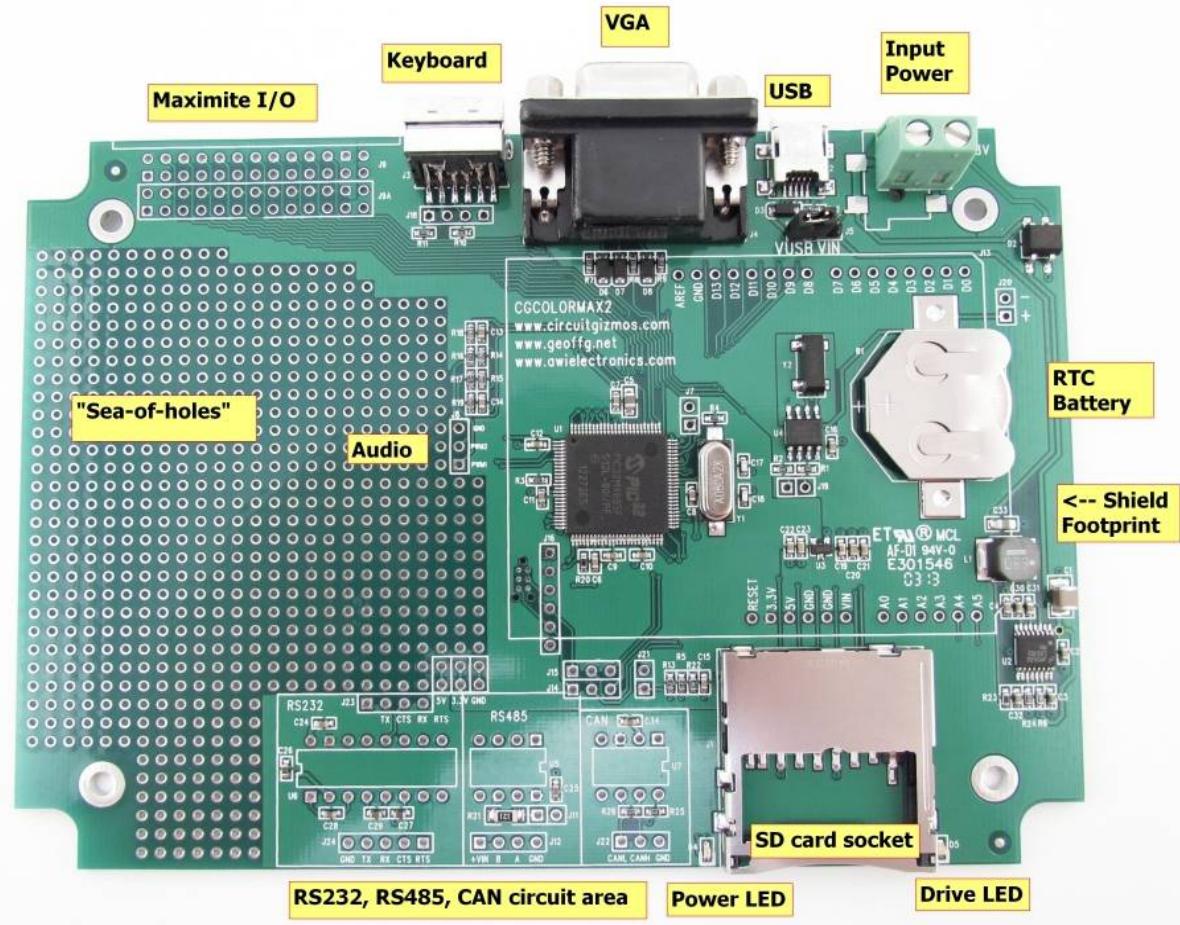


Illustration 109: CGCOLORMAX rev 2 I/O callouts.

Power Supply (CGCOLORMAX2)

The power select jumper (J5) near to the USB jack and to input power selects powering the board from either the USB connector or from the input power jack. Place a header on VUSB to power the board from USB. Move the header to VIN to power the board from the input power connector.

The USB connection provides 5V to the board. The USB hub that you use with this board should supply at least 250mA. Typical powered hubs provide 500mA. 3.3V is regulated internally to power the internal circuitry.

Input voltage (J6) is 8-18VDC. The power supply should be rated at being able to supply a minimum of 250 millamps. 5V is regulated from this, and 3.3V is then regulated from 5V.

The 5V regulator can regulate 700mA for on-board circuits. The 3.3V supply can regulate 200mA (subtracted from the 5V 700mA). Existing 3.3V circuits consume 150mA on the 3.3V line. This leaves 3.3V 50mA, and 5V 500mA for user circuitry.



Illustration 110: Barrel jack and screw terminal.

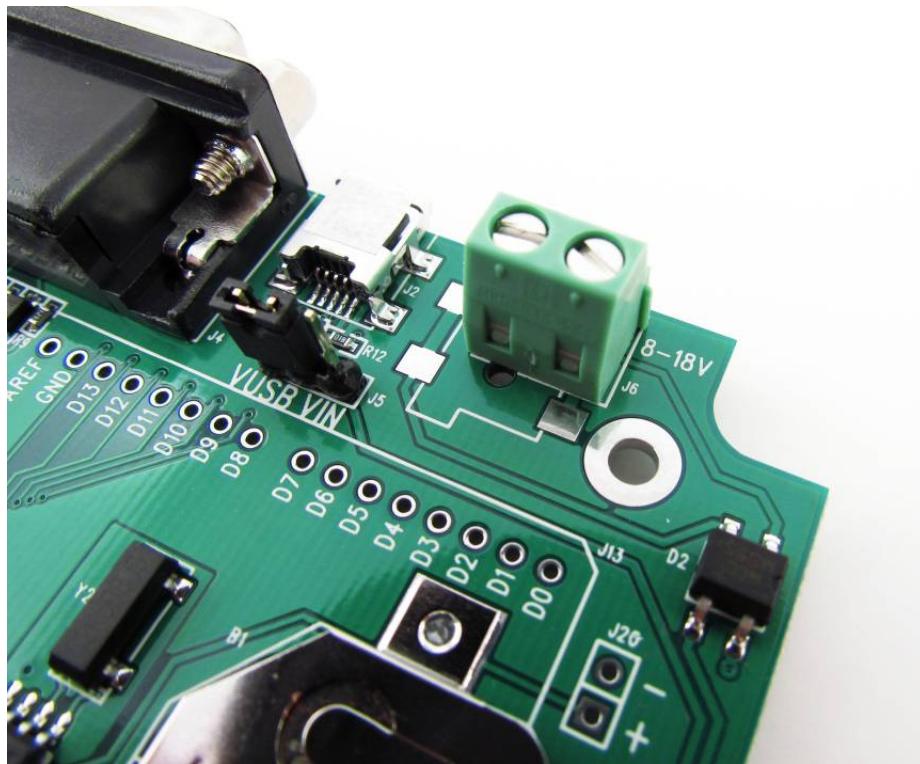


Illustration 111: The CGCOLORMAX2 power input populated with the screw terminal connector.

J6 can be populated with either the supplied screw terminal, or the 5.5mm/2.1mm barrel jack. When using a DC supply with the CGCOLORMAX2, the board circuitry takes care of polarity. The barrel connector used can be center positive or center negative. Positive/negative can be exchanged on the screw connections, too.

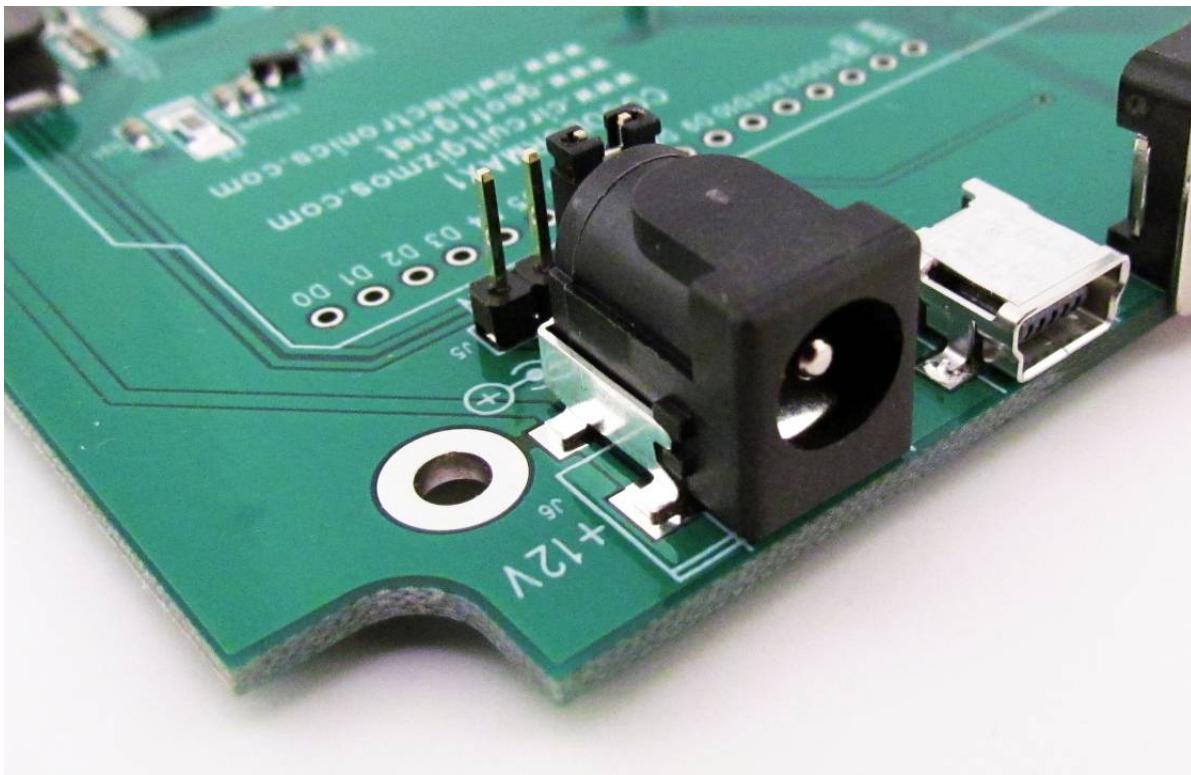


Illustration 112: The barrel jack installed on a CGCOLORMAX.

The CGCOLORMAX2 can be powered by a PC USB connection, but it can also be powered through the USB connector by using a wall-wart or car power adapter that provides +5V via a USB mini-B plug. Things such as cell phone chargers for AC (wall) use, or car cigarette lighter chargers will work if they provide enough current and a regulated +5V.

Rear Connectors (CGCOLORMAX2)

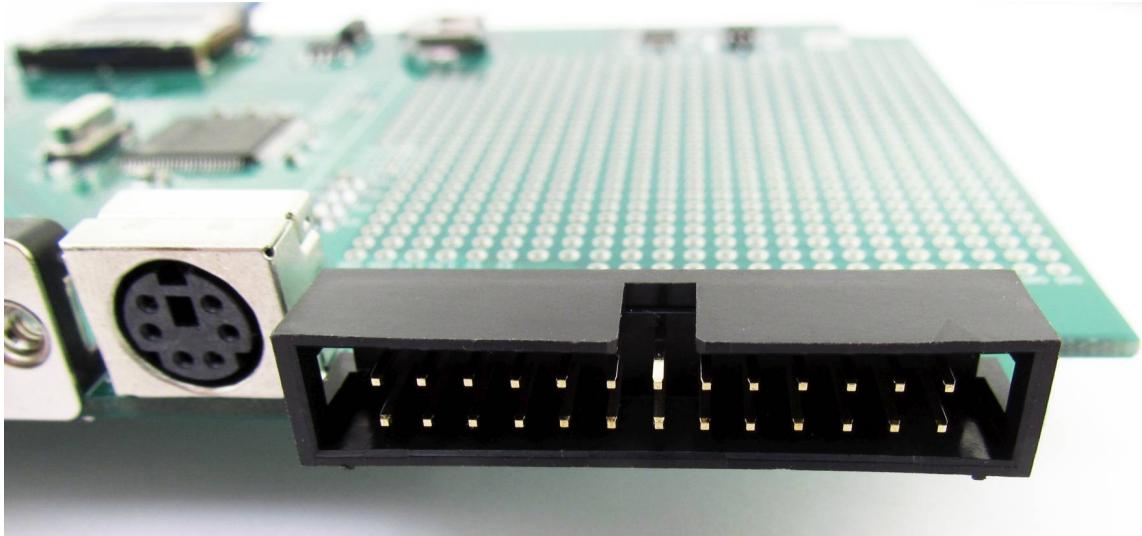


Illustration 113: View of 2x13 "Maximite connector" from back of board. Pin 1 is in the upper right, pin 2 is below it.

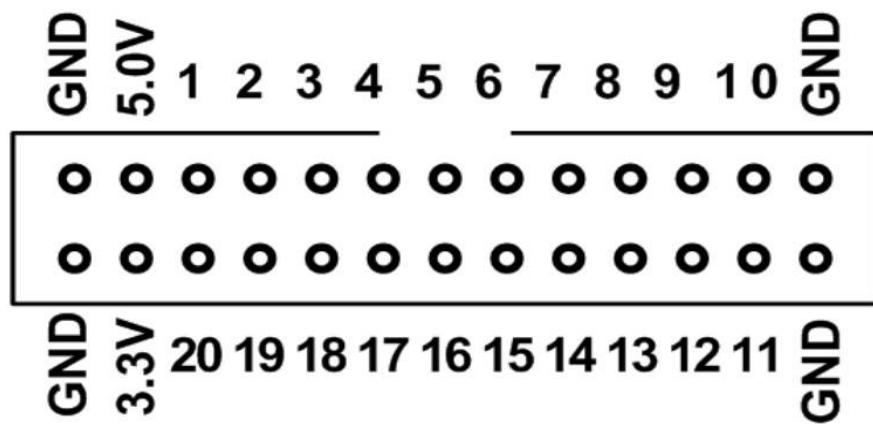
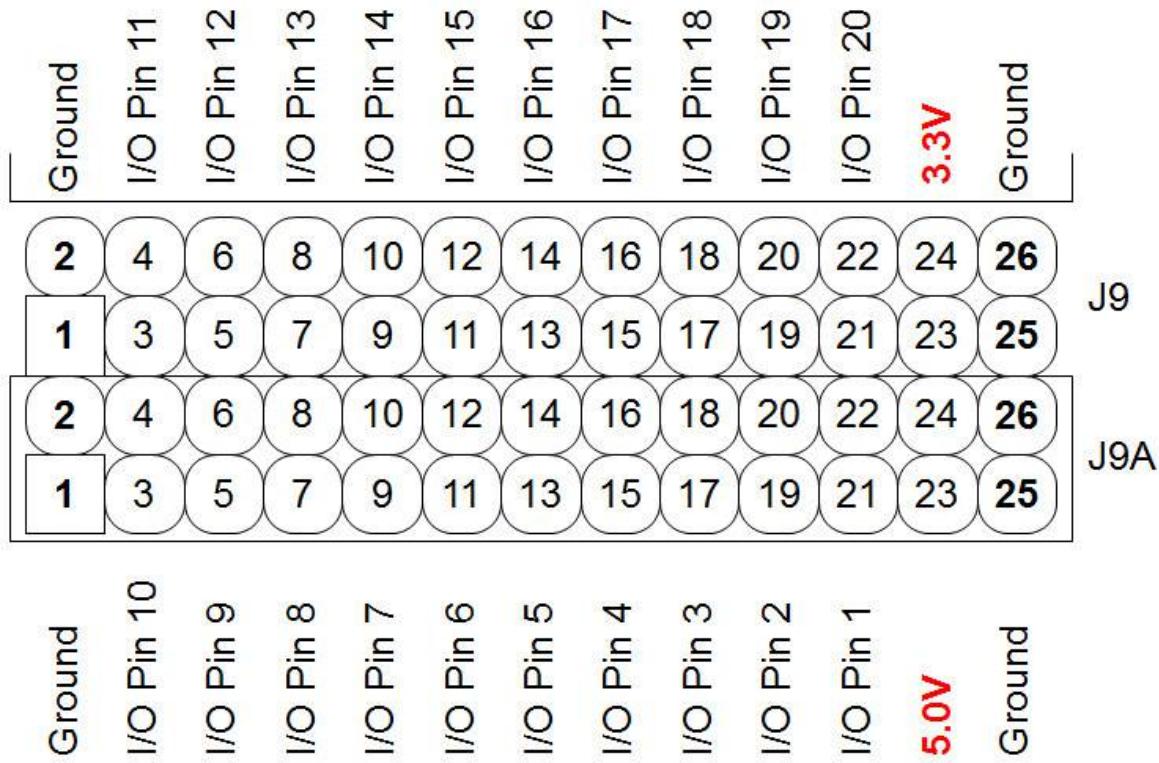


Figure 114: View of 2x13 right-angle "Maximite connector" (if populated) from back of board. This is J9 for the CGCOLORMAX2. Pin 1 is the GND in the upper right, pin 2 is the GND below it.



Top view of J9/J9A for CGCOLORMAX2

Illustration 115: View of the unpopulated J9/J9A connector from the top of the board.

J9 (duplicated on J9A)– CGCOLORMAX2 “Maximite I/O connector:

<i>Connector Pin</i>	<i>Use</i>
1	Ground
2	Ground
3	I/O Pin 10
4	I/O Pin 11
5	I/O Pin 9
6	I/O Pin 12
7	I/O Pin 8
8	I/O Pin 13
9	I/O Pin 7
10	I/O Pin 14
11	I/O Pin 6
12	I/O Pin 15
13	I/O Pin 5
14	I/O Pin 16
15	I/O Pin 4
16	I/O Pin 17
17	I/O Pin 3
18	I/O Pin 18
19	I/O Pin 2
20	I/O Pin 19
21	I/O Pin 1
22	I/O Pin 20
23	5.0V
24	3.3V
25	Ground
26	Ground



Illustration 116: The rear connections on the CGCOLORMAX2: The green screw terminal for power, the USB jack, VGA, and PS/2 keyboard.

Shield Connections (CGCOLORMAX2)

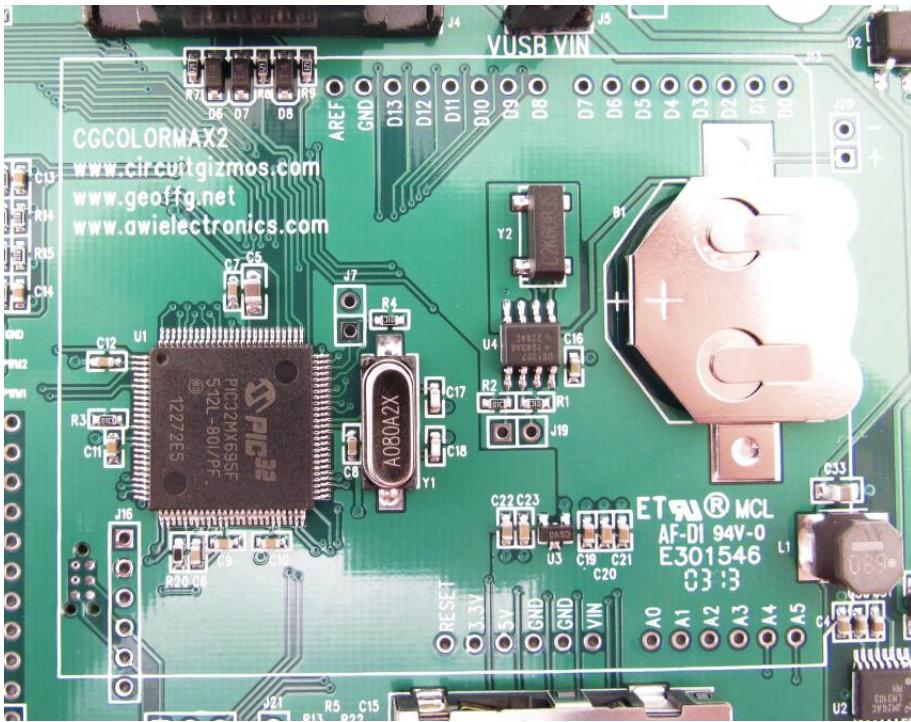


Illustration 117: The "Shield" footprint on a CGCOLORMAX2.

Arduino Shield footprint J13

Shield Pins	Pin from software perspective
D0	I/O Pin 21
D1	I/O Pin 22
D2	I/O Pin 23
D3	I/O Pin 24
D4	I/O Pin 25
D5	I/O Pin 26
D6	I/O Pin 27
D7	I/O Pin 28
D8	I/O Pin 29
D9	I/O Pin 30
D10	I/O Pin 31
D11	I/O Pin 32

D12	I/O Pin 33
D13	I/O Pin 34
A0	I/O Pin 35
A1	I/O Pin 36
A2	I/O Pin 37
A3	I/O Pin 38
A4	I/O Pin 39 (see J15 information)
A5	I/O Pin 40 (see J14 information)

J15 – Shield A4 connection. Connect pins 1-2 to have an I2C connection to SDA on the programming header. Connect 2-3 to connect A4 to I/O Pin 12.

J14 – Shield A5 connection. Connect pins 1-2 to have an I2C connection to SCL on the programming header. Connect 2-3 to connect A5 to I/O Pin 13.

A4/A5 on the Shield connection lead to J15/J14 respectively. A4/A5 can be analog inputs or I2C connections depending on how you may want to use them for the shields that you choose. They are not connected to anything to begin with on the CGCOLORMAX2.

If you wish to use A4/A5 for the I2C interface that MMBasic supports, you will want to connect A4 to I/O Pin 12 (J15 pins 2-3) and connect A5 to I/O Pin 13 (J14 pins 2-3).

If you wish to use A4/A5 for the analog interface that MMBasic supports, you will want to connect J15 pins 1-2 and connect J14 pins 1-2.

Interface Circuits (CGCOLORMAX2)

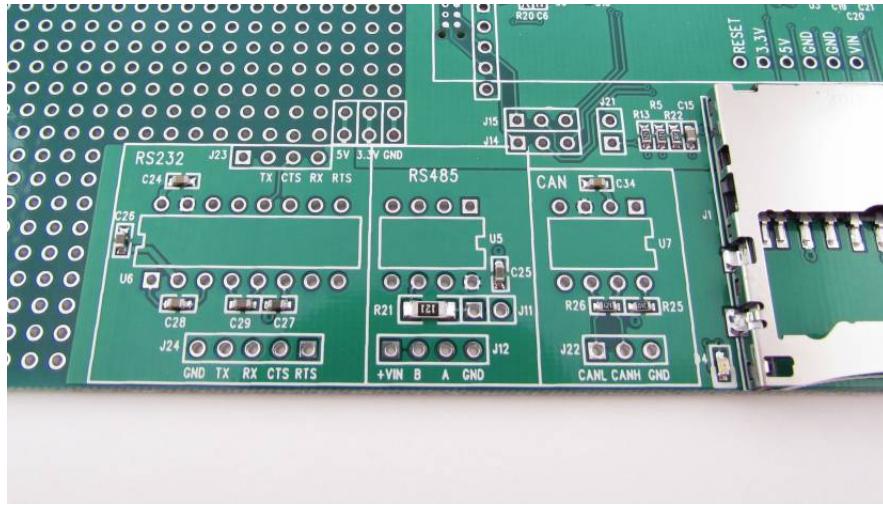


Illustration 118: CGCOLORMAX2 has support for interface circuits by just adding DIP sockets and chips.

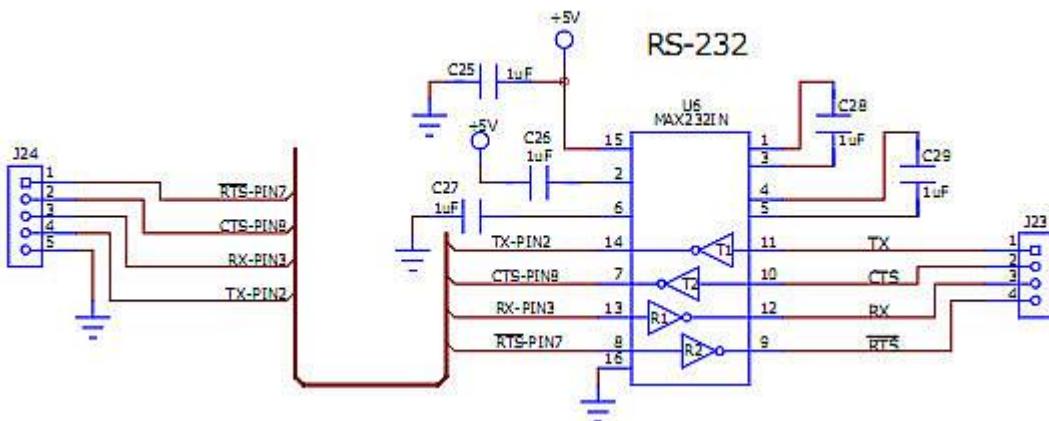


Illustration 119: The RS232 circuit of the CGCOLORMAX2. The driver chip is not populated on the CGCOLORMAX2. The chip is through-hole and can be added by soldering it in place.

J23 - RS232 (CGCOLORMAX2 connection side)

Pin	Use
1	TX (CGCOLORMAX2 microcontroller side)
2	CTS (CGCOLORMAX2 microcontroller side)
3	RX (CGCOLORMAX2 microcontroller side)

Note that the silkscreen printed on the circuit board is a little bit off. The labels have been printed offset by .1".

You can select one of the CGCOLORMAX2's two serial ports for connection to this circuit.

J24 - RS232 (line/cable side)

<i>Pin</i>	<i>Use</i>
1	RS232 RTS
2	RS232 CTS
3	RS232 RX
4	RS232 TX
5	Ground

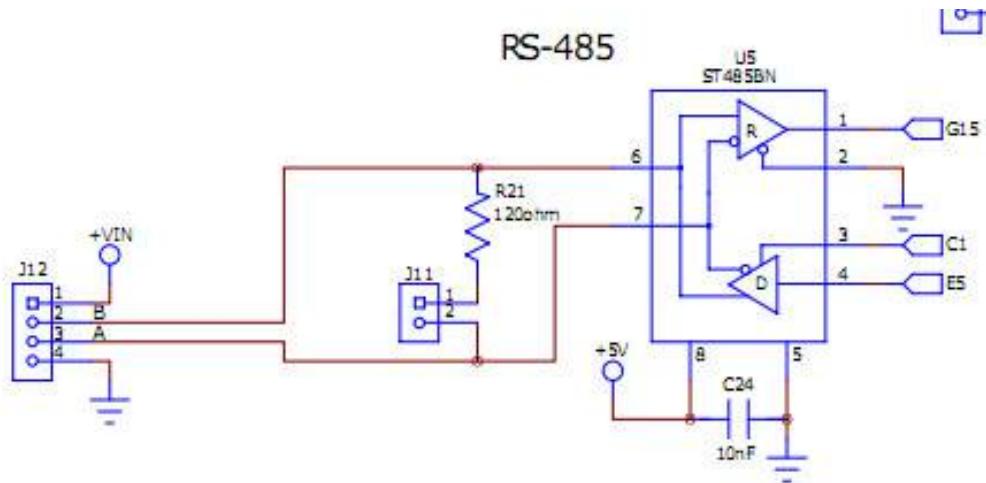


Illustration 120: The RS485 circuit of the CGCOLORMAX2. The driver chip is not populated on the CGCOLORMAX2. The chip is through-hole and can be added by soldering it in place.

J11 – A jumper here will enable RS485 line termination.

J12 – RS485 (line/cable side)

Pin	Use
1	+Vin (Connected to J6)
2	RS485 B
3	RS485 B
4	Ground

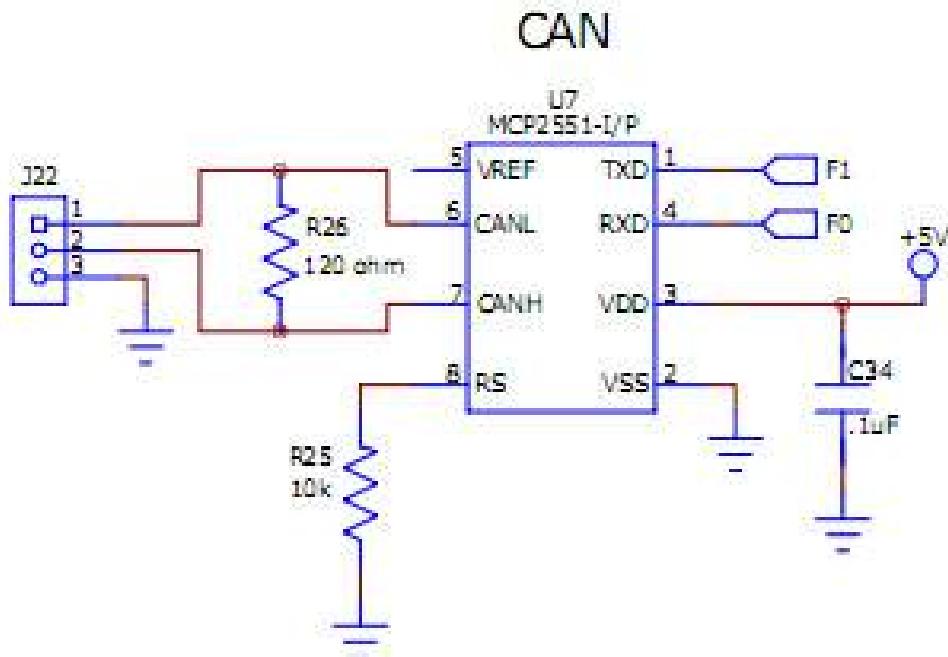


Illustration 121: The CAN circuit of the CGCOLORMAX2. The driver chip is not populated on the CGCOLORMAX2. The chip is through-hole and can be added by soldering it in place.

J22 – CAN (line/cable side)

Pin	Use
1	CANL
2	CANH
3	Ground

Misc Connections (CGCOLORMAX2)

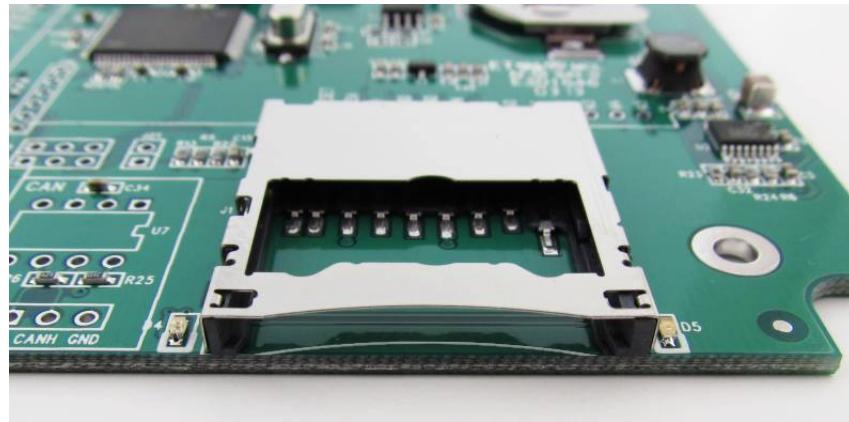


Illustration 122: Full-size SD card holder on the CGCOLORMAX2.

D4 – Green “Power/Firmware” LED, left of SD card holder.

D5 – Blue SD card access LED on the right of the SD card holder.

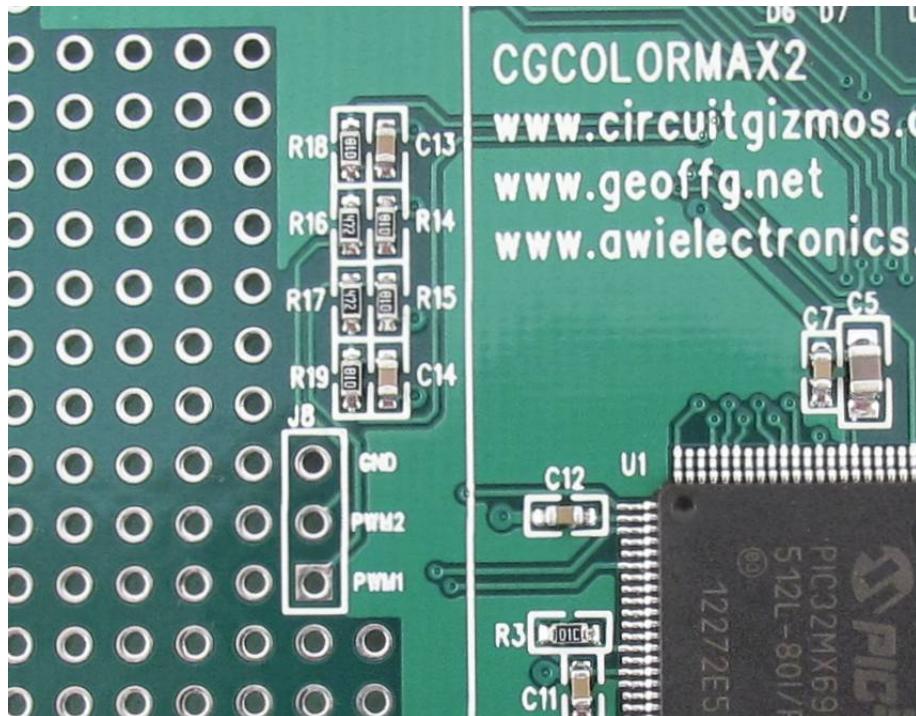


Illustration 123: Audio/PWM header.

J8 – Stereo audio (PWM) out. Pin1 is PWM1, pin2 is PWM2, pin3 is ground.

J16 (1x6) / J17 (2x3 pattern next to J16, no silk screen label) are PIC32 programming headers.

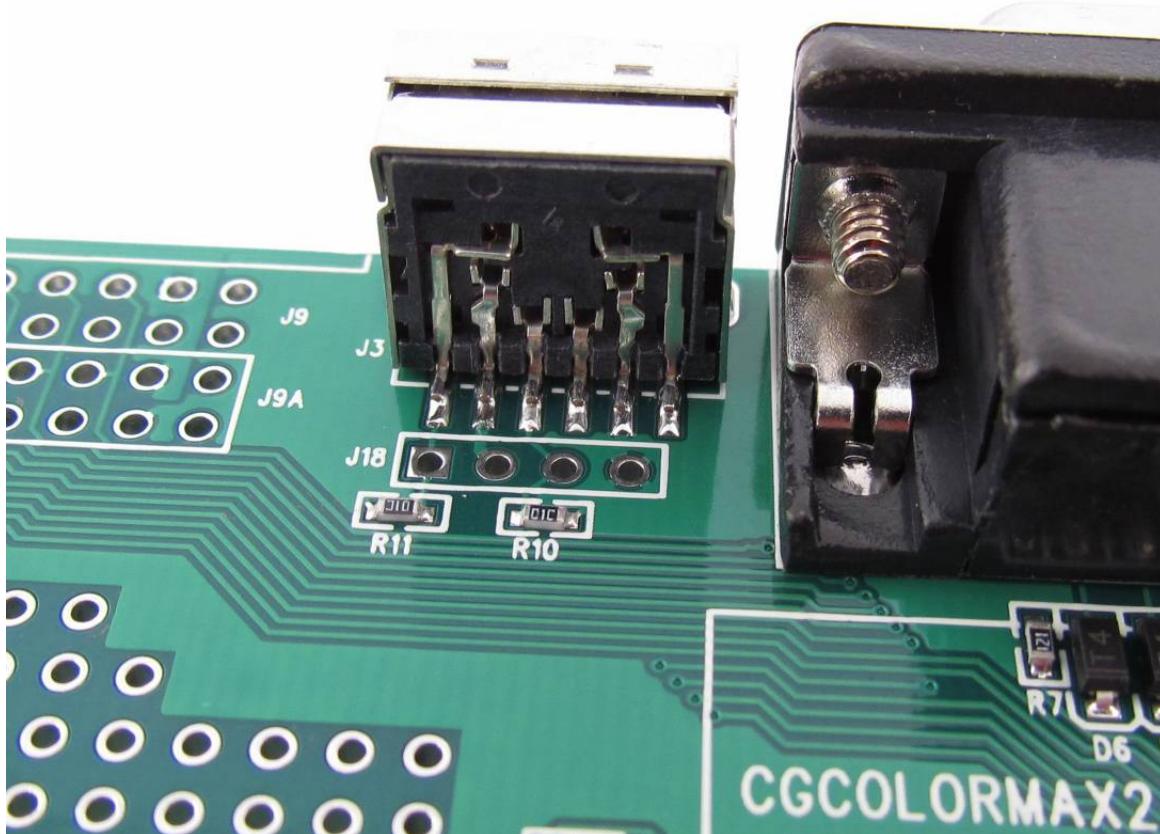


Illustration 124: J18 - extra connections for PS/2 keyboard.

J18 – 1x4 extra keyboard header. A PS/2 keyboard circuit (the CGKEYCHIP1, for example) can be wired to this extra connector.

J18-1	PS/2 clock
J18-2	Ground
J18-3	PS/2 data
J18-4	+5V

J19 – I2C lines for system (RTC) use. J19-1 is SDA, J19-2 is SCL.

J20 – This header is in parallel with the RTC battery if you choose to not use the coin cell but instead use an external clock battery. J20- 1 is positive. J20-2 is ground. The replacement coin cell is a CR2016.

J21 – This is in parallel with the “Power/Firmware” LED and resistor circuit. You can control the LED via MMBasic commands and this connection will let you control an external circuit in parallel with the LED.

There are connections to ground, +5V, and +3.3V in the “sea-of-holes” prototype area for connecting the custom circuits to power.

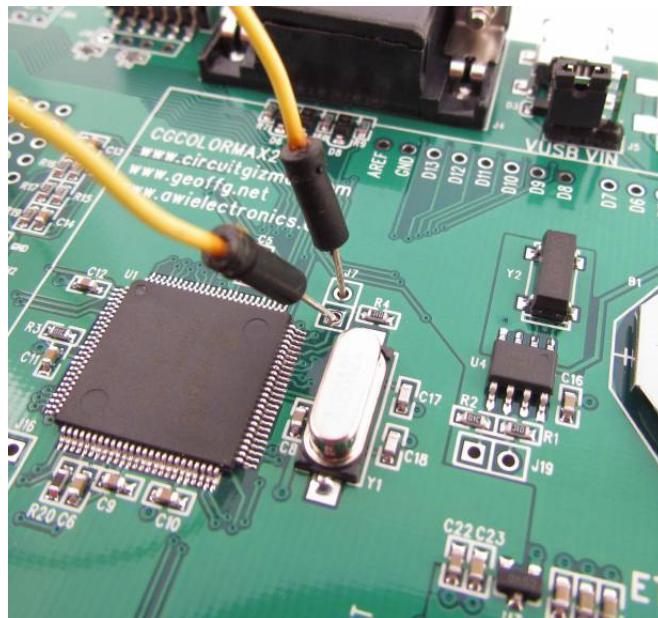


Illustration 125: A bootloader wire jumper in place on a CGCOLORMAX2.

J7 – A jumper in place is for firmware reload. The two connections can be temporarily shorted together with a short piece of solid wire while applying USB power to the board to put the board on bootloader mode.

Pin Outs and Function (CGCOLORMAX2)

J9 Pin / J13 Pin	Function	Analog	Digital In	Freq In	Period In	Count In	Digital Out	Open Collector	5V Tolerant	Additional Function
21	I/O 1	Analog In	Digital In				Digital Out			
19	I/O 2	Analog In	Digital In				Digital Out			
17	I/O 3	Analog In	Digital In				Digital Out			
15	I/O 4	Analog In	Digital In				Digital Out			
13	I/O 5	Analog In	Digital In				Digital Out			
11	I/O 6	Analog In	Digital In				Digital Out			
9	I/O 7	Analog In	Digital In				Digital Out			
7	I/O 8	Analog In	Digital In				Digital Out			
5	I/O 9	Analog In	Digital In				Digital Out			
3	I/O 10	Analog In	Digital In				Digital Out			
4	I/O 11		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
6	I/O 12		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
8	I/O 13		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
10	I/O 14		Digital In	Freq In	Period In	Count In	Digital Out	O.C.	5V Tolerant	
12	I/O 15		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 Receive
14	I/O 16		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 Transmit
16	I/O 17		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 RTS
18	I/O 18		Digital In				Digital Out	O.C.	5V Tolerant	Serial 1 CTS
20	I/O 19		Digital In				Digital Out	O.C.	5V Tolerant	
22	I/O 20		Digital In				Digital Out	O.C.	5V Tolerant	
D0	I/O 21		Digital In				Digital Out	O.C.	5V Tolerant	Serial 2 Receive
D1	I/O 22		Digital In				Digital Out	O.C.	5V Tolerant	Serial 2 Transmit
D2	I/O 23		Digital In				Digital Out	O.C.	5V Tolerant	
D3	I/O 24		Digital In				Digital Out	O.C.	5V Tolerant	
D4	I/O 25		Digital In				Digital Out	O.C.	5V Tolerant	
D5	I/O 26		Digital In				Digital Out	O.C.	5V Tolerant	
D6	I/O 27		Digital In				Digital Out	O.C.	5V Tolerant	
D7	I/O 28		Digital In				Digital Out	O.C.	5V Tolerant	
D8	I/O 29		Digital In				Digital Out	O.C.	5V Tolerant	
D9	I/O 30		Digital In				Digital Out	O.C.	5V Tolerant	
D10	I/O 31		Digital In				Digital Out	O.C.	5V Tolerant	
D11	I/O 32		Digital In				Digital Out	O.C.	5V Tolerant	
D12	I/O 33		Digital In				Digital Out	O.C.	5V Tolerant	
D13	I/O 34		Digital In				Digital Out	O.C.	5V Tolerant	
A0	I/O 35	Analog In	Digital In				Digital Out			
A1	I/O 36	Analog In	Digital In				Digital Out			
A2	I/O 37	Analog In	Digital In				Digital Out			
A3	I/O 38	Analog In	Digital In				Digital Out			
A4	I/O 39	Analog In	Digital In				Digital Out			
A5	I/O 40	Analog In	Digital In				Digital Out			

Firmware Update (CGCOLORMAX2)

The CGCOLORMAX2 is loaded at the factory with a bootloader and with the version of MMBasic that is current at the time of board production. New versions of MMBasic are released periodically, and the CGCOLORMAX2 uses a bootloader that allows for MMBasic update.

Upgrades are done via the USB interface when the boot loader is running.

Powering up the CGCOLORMAX2 will normally cause MMBasic to run. If a wire jumper is in place on J7, then on power up the bootloader will run.

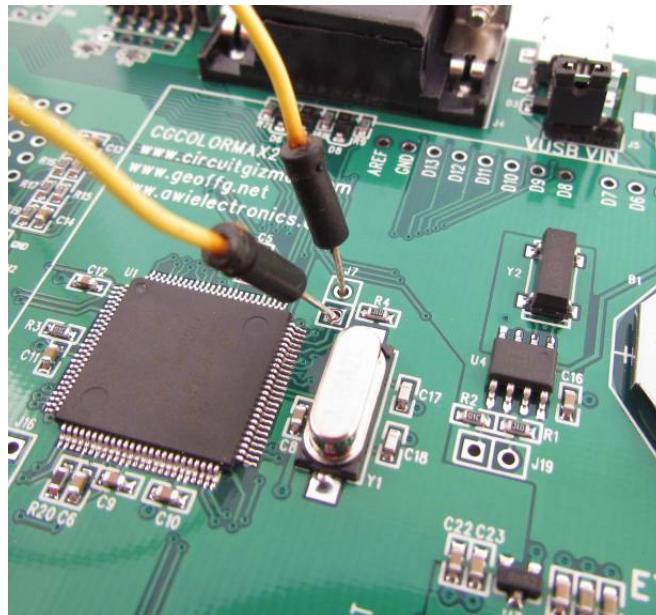


Illustration 126: A bootloader wire jumper in place on a CGCOLORMAX2.

The jumper can be any simple thin wire momentarily held in place to electrically connect both contacts of J7. Even a bare paperclip would work.

When the bootloader is running, the green LED on the CGCOLORMAX2 will flash on and off. During this time, a PC program can connect through USB to the CGCOLORMAX2 for updating firmware. *Bootloader.exe* will load MMBasic onto a CGCOLORMAX2. The program is part of a zip file download from CircuitGizmos. *Bootloader.exe* runs as a stand-alone program without needing installation.

Firmware Upgrade Steps

Start with the CGCOLORMAX2 unpowered.

Have the bootloader jumper wire described above in place on the device while you apply power by connecting the USB cable to your PC. The green CGCOLORMAX2 power LED will rapidly flash to indicate that the boot loader is in control. (At this point the bootloader is running and the jumper can be removed.)

The computer should automatically recognize the device and load the appropriate driver. (The CGCOLORMAX2 will show up in the Windows Device Manager as a Human Interface Device when connected to the bootloader.)

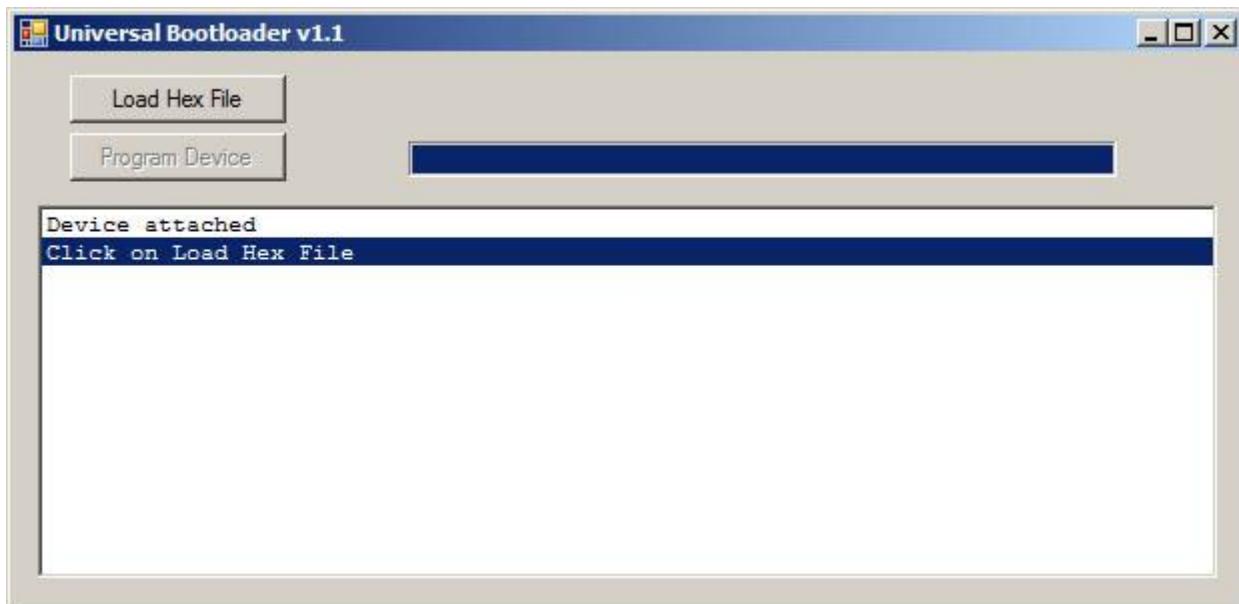


Illustration 127: Bootloader.exe has detected and connected to a CGMMSTICK/CGCOLORMAX in bootloader mode.

Run BootLoader.exe it will automatically detect the device and show the message "Device attached".

If the **Load Hex File** button in bootLoader.exe is grayed out it means that the CGMMSTICK1 is not connected or not in boot load mode. Check the USB cable and that the green CGCOLORMAX2 LED is flashing.

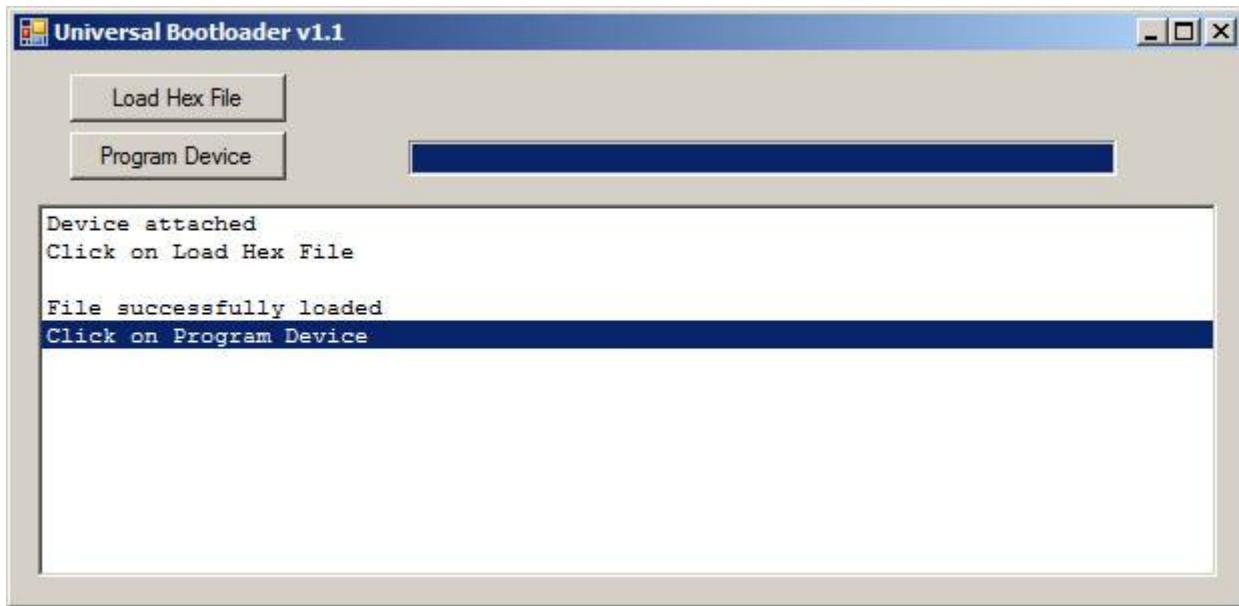


Illustration 128: Bootloader.exe with a firmware hex file loaded.

Click on the **Load Hex File** button and load the firmware upgrade file. The firmware file will have a .hex extension.

An example hex file name might be *Maximite_MMBasic_V4.3.hex*.

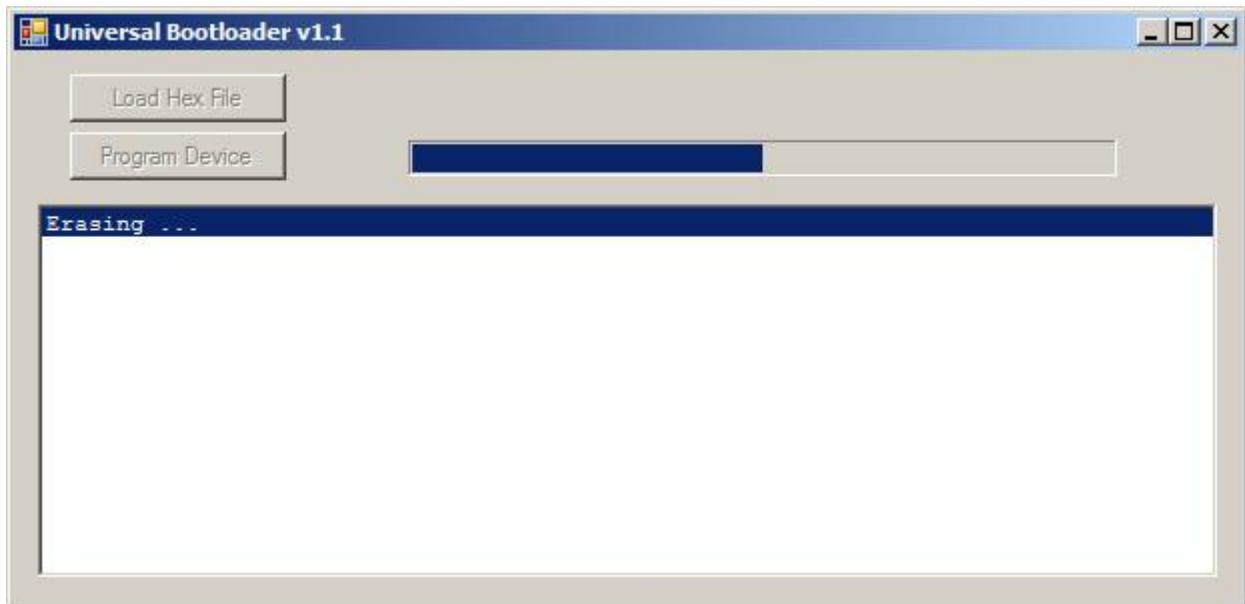


Illustration 129: Bootloader.exe erasing the CGMMSTICK/CGCOLORMAX firmware.

After the firmware hex file is loaded into bootloader.exe, press the **Program Device** button.

Bootloader.exe will erase the old firmware first. This isn't a full erase of the CGCOLORMAX2 chip, as a full erase would also erase the bootloader and that needs to stay on the chip. Everything BUT the bootloader is erased, which means that the contents of the on-chip A: drive is also completely cleared.

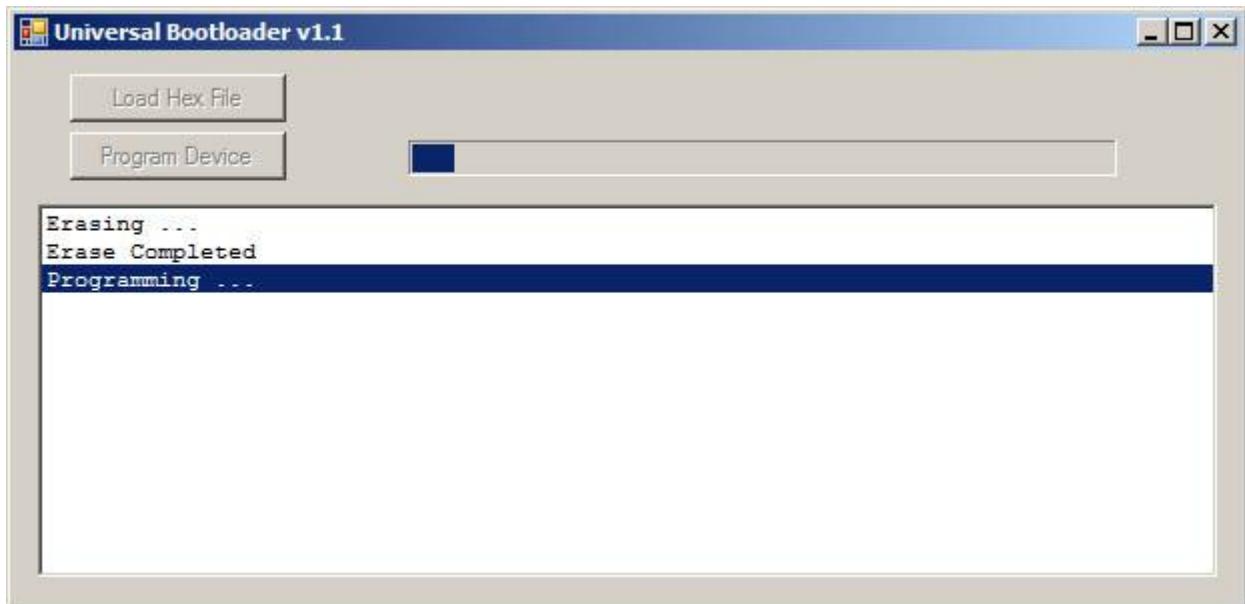


Illustration 130: After erasing, bootloader.exe will program the new firmware to the CGMMSTICK/CGCOLORMAX.

Once erased, bootloader will program the new firmware to the CGCOLORMAX2. There is a progress bar that shows the progress of this programming operation. Immediately after programming bootloader.exe will double-check to make sure that the firmware was correctly loaded with a verification step.

The entire programming and verification process should take only about a minute.

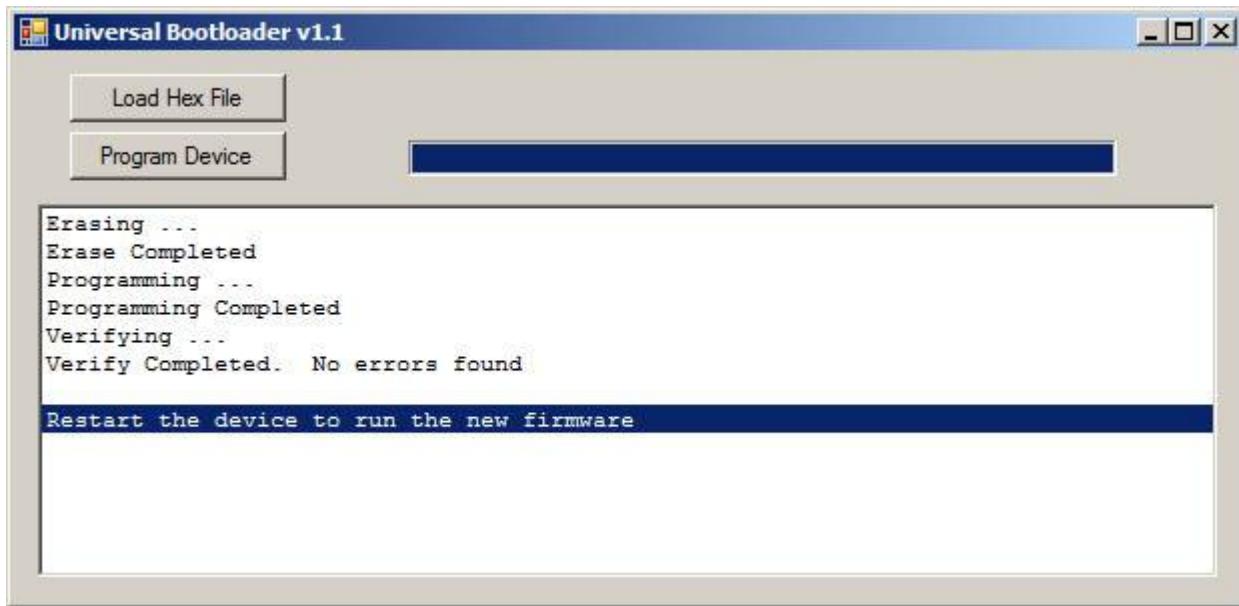


Illustration 131: Programming finished.

After verification the green CGCOLORMAX2 power LED will flash slowly. This indicates that the new firmware on the CGCOLORMAX2 is completely programmed.

Remove the power connection from the CGCOLORMAX2. Make sure that the J7 jumper wire has also been removed.

Power the CGCOLORMAX2 again and the updated MMBasic firmware will run.

CGMMSTICK and CGCOLORMAX I/O Characteristics

Typical Digital Output Electrical Characteristics

Output Low Voltage: 0.0 to 0.4V

Output High Voltage: 2.4 to 3.3V

Maximum current draw/sink on any I/O pin: 25mA

Maximum cumulative current draw/sink for all I/O pins: 200mA (150mA derated)

Maximum voltage for 5V tolerant open collector pins: 5.5V

Typical Digital Input Electrical Characteristics

Maximum input voltage, normal pins: 3.6V

Maximum voltage for 5V tolerant input pins: 5.5V

Voltage range for a logic low input: 0.0 to 0.66V

Voltage range for a logic high input: 2.64 to 3.3V

Voltage range for a logic low input – 5V tolerant inputs: 2.64 to 5.5V

Analog input voltage range: 0.0 to 3.3V

CGCOLORMAX Composite Output

On the CGCOLORMAX Maximite composite video (in monochrome not color) can be accessed via the VGA connector using the following connections:

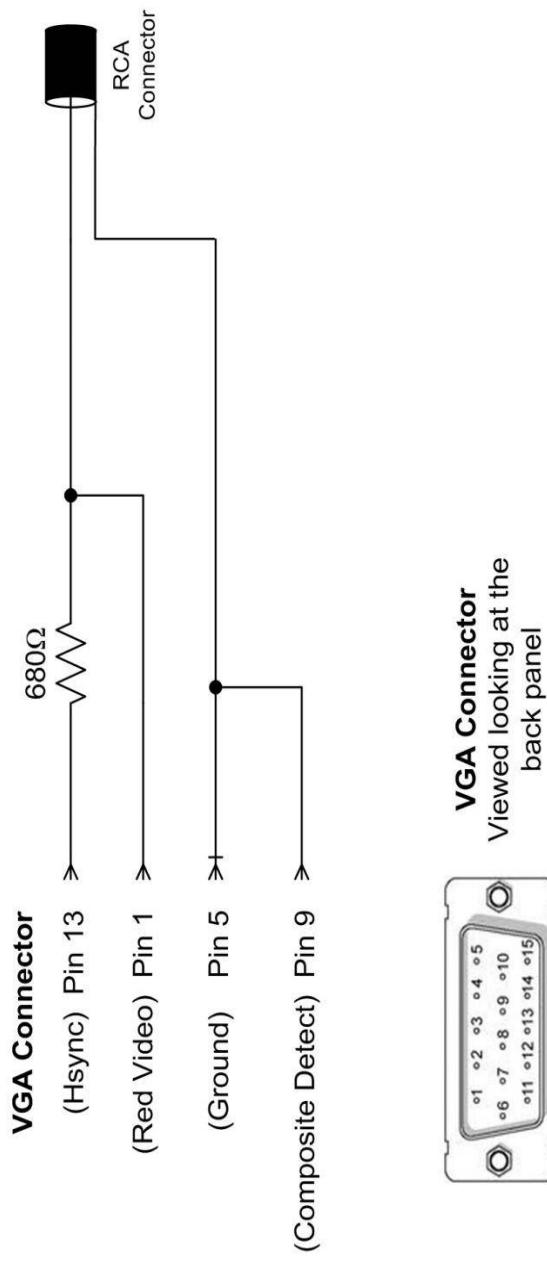


Illustration 132: Composite display circuit.

The screen resolution for composite display in PAL (default) mode is 304 horizontal, and 216 vertical. Screen resolution in NTSC is 304 horizontal, and 180 vertical.

Use CONFIG COMPOSITE to set either PAL or NTSC. NTSC is used for composite video in the United States.

Joystick Connections

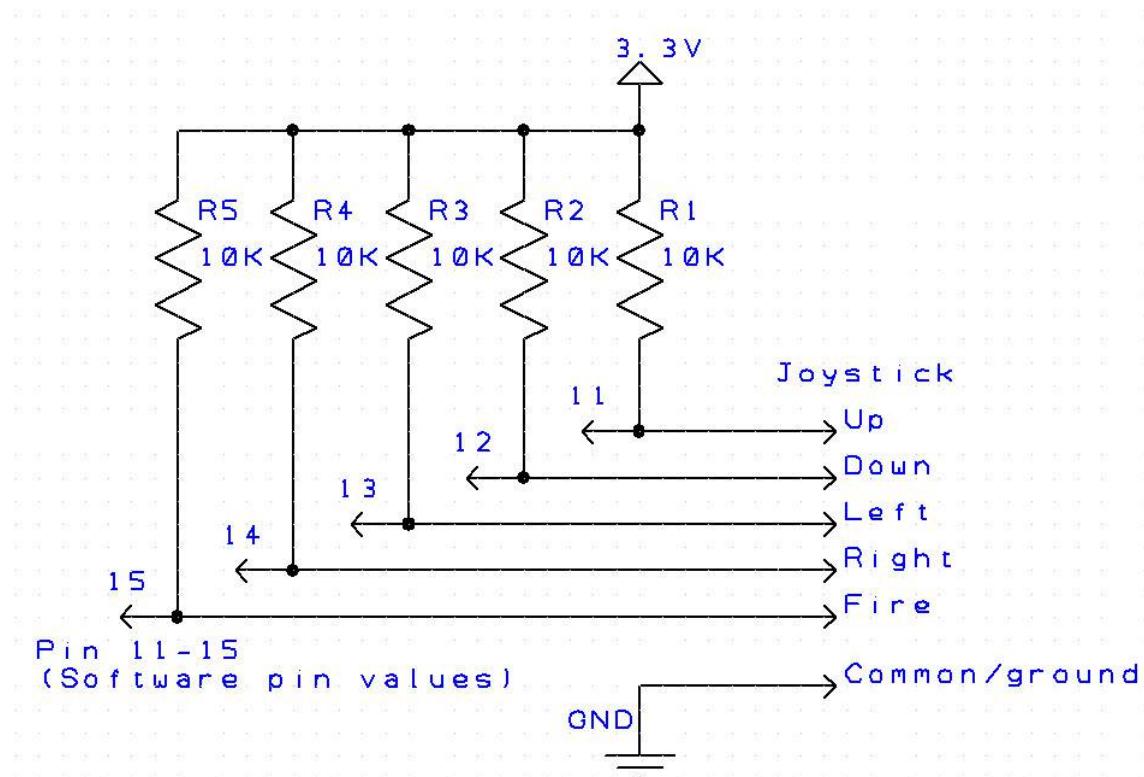


Illustration 133: Schematic for adding a single joystick to a CGMMSTICK or CGCOLORMAX.

Adding a single joystick to either the CGMMSTICK or CGCOLORMAX for playing games can be done by wiring a joystick to the signals that software can read Pin 9 through Pin 15.

Software pin	Joystick signal	J1 on CGMMSTICK	J9 on CGCOLORMAX
Pin(11)	Up	21	4
Pin(12)	Down	22	6
Pin(13)	Left	23	8
Pin(14)	Right	24	10
Pin(15)	Fire Button	25	12
Pin(10)	Pot Y	20	3
Pin(9)	Pot X	19	5

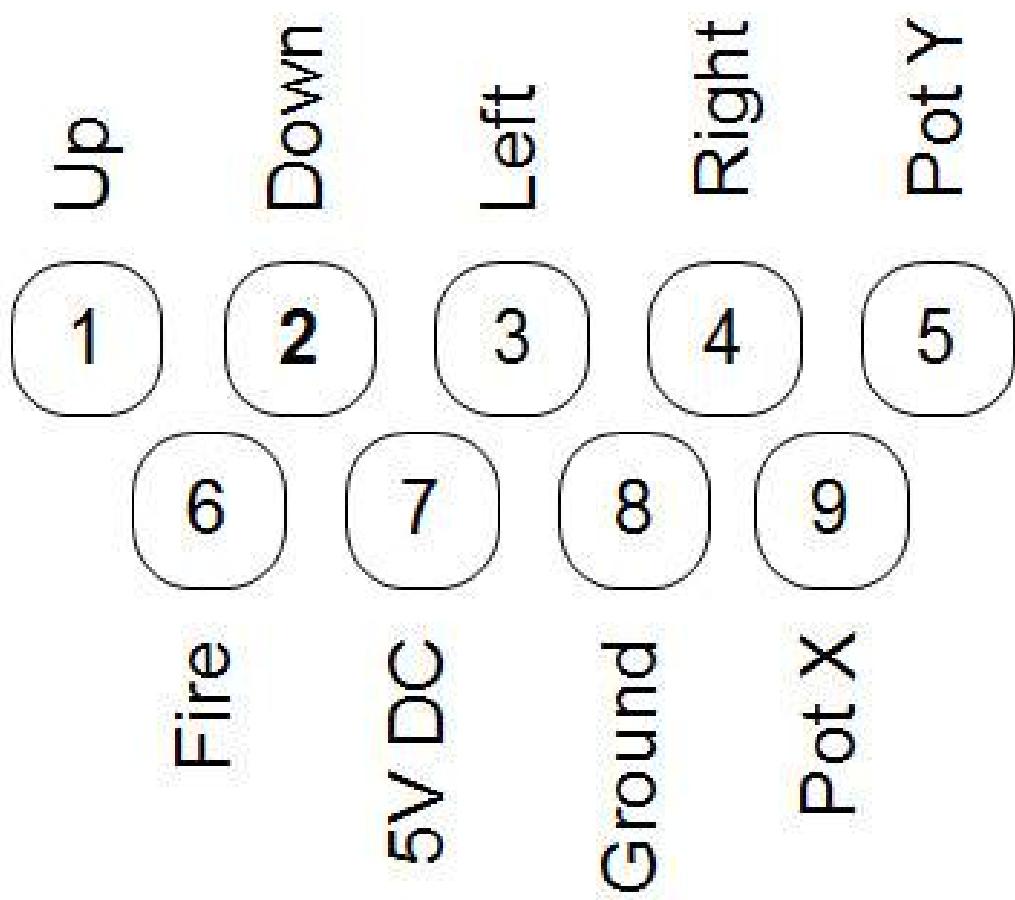


Illustration 134: Example 9-pin joystick connector from the computer perspective. View from the joystick is a right/left mirror image. Your joystick wiring may be different.

Example Shields

Some example shields that can be used with the CGCOLORMAX:



Illustration 135: CGCOLORMAX with relay shield.



Illustration 136: CGCOLORMAX with game shield.



Illustration 137: CGCOLORMAX with ethernet shield.



Illustration 138: CGCOLORMAX with GPRS shield.

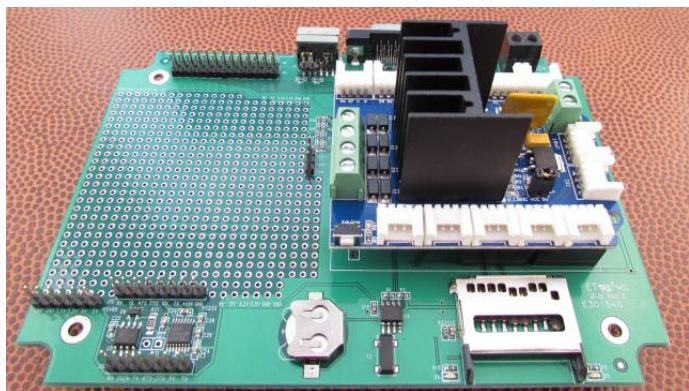


Illustration 139: CGCOLORMAX with motor control shield.

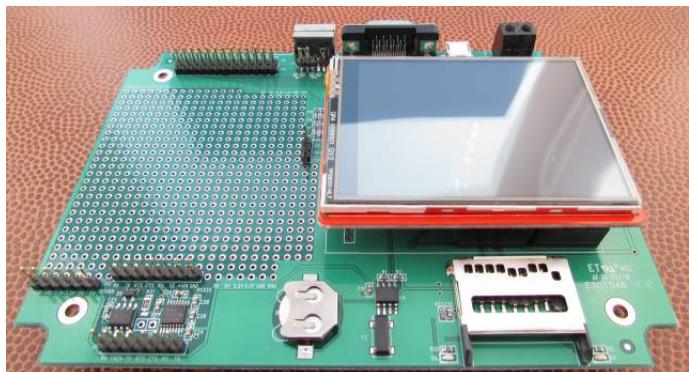


Illustration 140: CGCOLORMAX with graphic LCD touch screen.

MMBasic Reference

The MMBasic language can be broken into several major segments.

Language Constructs control the flow of the program.

Editing, Programming, and Debugging are commands used during program development.

Screen, Graphics, and Keyboard cover the functions of input and output that exclude the electrical I/O of the Maximite.

Math Functions are the functions that convert values and perform helpful math.

Character and String Functions assist in manipulating ASCII characters and text strings.

File System covers the access to the “drives” and transfer of data to/from them.

Input and Output cover all of the gritty hardware interface details

Language Constructs

CLEAR	Delete all variables and recover the memory used by them.
CONTINUE	Resume running a program that has been stopped by an END statement, an error, or CTRL-C.
DATA	Stores numerical and string constants to be accessed by READ.
DIM	Specifies variables that have more than one element in a single dimension.
DO LOOP	This structure will loop forever.
DO LOOP UNTIL	Loops until the expression following UNTIL is true.
DO WHILE LOOP	Loops while "expression" is true.
ELSE	Introduces a default condition in a multi-line IF statement.
ELSEIF THEN	Introduces a secondary condition in a multiline IF statement.
END	End the running program and return to the command prompt.
END FUNCTION	Marks the end of a user defined function.
END SUB	Marks the end of a user defined subroutine.
ENDIF	Terminates a multiline IF statement.
ERASE	Deletes arrayed variables and frees up the memory.
EXIT	EXIT by itself will terminate a DO...LOOP.
EXIT FOR	EXIT FOR will terminate a FOR...NEXT loop.
EXIT FUNCTION	EXIT FUNCTION provides an early exit from a defined function.
EXIT SUB	EXIT SUB provides an early exit from a defined subroutine.
FOR TO STEP	Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' equals 'finish'.
FUNCTION	Defines a callable function.
GOSUB	Initiates a subroutine call to the target which can be a line number or a label.
GOTO	Branches program execution to the target which can be a line number or a label.
IF THEN ELSE	Evaluates the expression 'expr' and performs the THEN statement if it is true or skips to the next line if false.
IF THEN GOTO	Evaluates the expression 'expr' and performs the THEN statement if it is true or skips to the next line if false.
IF THEN	Multiline IF statement with optional ELSE and ELSEIF cases and ending

ELSE	with
ELSEIF	ENDIF. Each component is on a separate line.
IRETURN	Returns from an interrupt.
LET	Assigns the value of 'expression' to the variable.
LOCAL	Defines a list of variable names as local to the subroutine or function.
LOOP UNTIL	Terminates a program loop.
NEXT	NEXT comes at the end of a FOR-NEXT loop.
ON nbr GOTO GOSUB	ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of variable.
OPTION BASE	Sets the lowest value for array subscripts to either 0 or 1.
OPTION Fnn	Sets the programmable function key 'Fnn' to the contents of 'string\$'.
OPTION PROMPT	Sets the command prompt to the contents of 'string\$'.
OPTION USB	Turn the USB output off and on.
PAUSE	Halt execution of the running program for 'delay' mS.
RANDOMIZE	Seeds the random number generator with 'nbr'.
READ	Reads values from DATA statements and assigns these values to the named variables.
REM or '	REM allows remarks to be included in a program.
RESTORE	Resets the line and position counters for DATA and READ statements to the top of the program file.
RETURN	RETURN concludes a subroutine called by GOSUB and returns to the statement after the GOSUB.
SETTICK	This will setup a periodic interrupt (or "tick").
SUB	Defines a callable subroutine.
TIMER	Resets the timer to a number of milliseconds.
WHILE WEND	WHILE initiates a WHILE-WEND loop.

Editing, Programming, and Debugging

AUTO	Enter automatic line entry mode.
CHAIN	Clear the current program from memory and run the new program.
CONFIG CASE	The CASE setting will change the case used for listing command and function names when using the LIST command.
CONFIG KEYBOARD	The KEYBOARD setting will change the keyboard layout to suit standard keyboards (US), United Kingdom (UK), French (FR), German (GR), Belgium (BE) or Italian (IT) keyboards.
CONFIG TAB	The TAB setting will set the spacing for the tab key.
COPYRIGHT	List all contributors to the MMBasic and summarize the copyright statement.
DATE\$	Set the date of the internal clock/calendar. Returns the current date based on MMBasic's internal clock as a string in the form "DD-MM-YYYY".
DELETE	Deletes a program line or a range of lines.
EDIT	Invoke the full screen editor.
ERROR	Forces an error and terminate the program.
LIST	Lists all lines in a program line or a range of lines.
LOAD	Loads a program called 'file\$' from the SD card into working memory.
MEMORY	List the amount of memory currently in use.
MERGE	Adds program lines from 'file\$' to the program in memory.
MM.DEVICE\$	A string representing the device or platform that MMBasic is running on.
MM.FNAME\$	The name of the file that will be used as the default for the SAVE command.
MM.VER	The version number of the firmware.
NEW	Deletes the program in memory and clears all variables.
OPTION BREAK	Set the value of the break key.
RENUMBER	Renumber the program currently held in memory.
RUN	Executes the program in memory.
SAVE	Saves the program in the current working directory on the SD card as 'file\$'.
TIME\$	Sets the time of the internal clock. TIME\$ returns the current time based on MMBasic's internal clock as a string in the form "HH:MM:SS" in 24 hour notation.
TROFF	Turns the trace facility off.
TRON	Turns on the trace facility.

Screen, Graphics, and Keyboard

BLIT	Copy one section of the video screen to another.
CIRCLE	Draws a circle on the video output.
CLS	Clears the video display screen and places the cursor in the top left corner.
COLLISION	Tests if a collision has occurred between sprite 'n' and the edge of the screen or another sprite.
COLOR	Sets the default color for commands that display on the screen.
CONFIG COMPOSITE	The COMPOSITE setting will change the timing for the composite video output.
CONFIG VIDEO	The VIDEO setting will switch the video output on or off.
FONT	Selects a font for the video output.
FONT LOAD/ UNLOAD	Will load the font contained in 'file\$'.
INKEY\$	Checks the keyboard and USB input buffers and, if there is one or more characters waiting in the queue, will remove the first character and return it as a single character in a string.
INPUT	Allows input from the keyboard to a list of variables.
KEYDOWN	Return the decimal ASCII value of the PS2 keyboard key that is currently held down or zero if no key is down.
LINE	Draws a line or box on the video screen.
LINE INPUT	Reads entire line from the keyboard into 'string-variable\$'.
LOADBMP	Load a bit-mapped image and display it on the video screen.
LOCATE	Positions the cursor to a location in pixels.
MM.HPOS	The current horizontal position (in pixels) following the last graphic or print command.
MM.HRES	The horizontal resolution of the current video display screen in pixels.
MM.VPOS	The current vertical position (in pixels) following the last graphic or print command.
MM.VRES	The vertical resolution of the current video display screen in pixels.
MODE	Sets the number of colors that can be displayed on the screen.
OPTION VIDEO	VIDEO OFF prevents the output from the PRINT command from being displayed on the video output (VGA or composite). The VIDEO ON option will revert to the normal action.
PIXEL	Set a pixel on the VGA or composite screen to a color. Returns the value of a pixel on the VGA or composite screen.
POS	Returns the current cursor position in the line in characters.
PRESET	Turn off (PRESET) or on (PSET) a pixel on the video screen.
PRINT	Outputs text to the screen.

PRINT @	Same as the PRINT command except that the cursor is positioned at the coordinates x, y.
PSET	Turn off (PRESET) or on (PSET) a pixel on the video screen.
SAVEBMP	Saves the current VGA or composite screen as a BMP file.
SCANLINE	This command can be used to set the color of individual horizontal scan lines on the VGA monitor when in MODE 1,7 (monochrome with white foreground).
SPRITE	Load and manipulate sprites on the screen.
TAB	Outputs spaces until the column indicated by 'number' has been reached.

Math Functions

ABS	Returns the absolute value of the argument 'number'.
ATN	Returns the arc tangent value of the argument 'number' in radians.
CINT	Round numbers with fractional portions up or down to the next whole number or integer.
COS	Returns the cosine of the argument 'number' in radians.
DEG	Converts 'radians' to degrees.
EXP	Returns the exponential value of 'number'.
FIX	Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point.
INT	Truncate an expression to the next whole number less than or equal to the argument.
LOG	Returns the natural logarithm of the argument 'number'.
PI	Returns the value of pi.
RAD	Converts 'degrees' to radians.
RND	Returns a pseudo random number in the range of 0 to 0.99999.
SGN	Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.
SIN	Returns the sine of the argument 'number' in radians.
SQR	Returns the square root of the argument 'number'.
TAN	Returns the tangent of the argument 'number' in radians.

Character and String Functions

ASC	Returns the ASCII code for the first letter in the argument 'string\$'.
BIN\$	Returns a string giving the binary (base 2) value for the 'number'.
CHR\$	Returns a one-character string consisting of the character corresponding to the ASCII code indicated by argument 'number'.
FORMAT\$	Will return a string representing 'nbr' formatted according to the specifications in the string 'fmt\$'.
HEX\$	Returns a string giving the hexadecimal (base 16) value for the 'number'.
INSTR	Returns the position at which string-pattern\$ occurs in string-searched\$, beginning at start-position.
LCASE\$	Returns 'string\$' converted to lowercase characters.
LEFT\$	Returns a substring of 'string\$' with 'number-of-chars' from the left (beginning) of the string.
LEN	Returns the number of characters in string\$.
MID\$	Returns a substring of 'string\$' beginning at 'start-position-in-string' and continuing for 'number-of-chars' bytes.
OCT\$	Returns a string giving the octal (base 8) representation of 'number'.
RIGHT\$	Returns a substring of 'string\$' with 'number-of-chars' from the right (end) of the string.
SPACE\$	Returns a string of blank spaces 'number' bytes long.
SPC	Returns a string of blank spaces 'number' bytes long.
STR\$	Returns a string in the decimal (base 10) representation of 'number'.
STRING\$	Returns a string 'number' bytes long consisting of either the first character of string\$ or the character representing the ASCII value ascii-value.
UCASE\$	Returns 'string\$' converted to uppercase characters.
VAL	Returns the numerical value of the 'string\$'.

File System

CHDIR	Change the current working directory on the SD card to 'dir\$'
CLOSE	Close the file(s) previously opened with the file number 'nbr'.
COPY	Copy the file named 'src\$' to another file named 'dest\$'.
CWD\$	Returns the current working directory on the SD card as a string.
DIR\$	Will search an SD card for files and return the names of entries found.
DRIVE	Change the default drive used for file operations that do not specify a drive to that specified in drivespec\$.
EOF	Will return true if the file previously opened for INPUT with the file number 'nbr' is positioned at the end of the file.
FILES	Lists files in the current directory on the SD card.
INPUT	Allows input from the from a file previously opened to a list of variables.
INPUT\$	Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number 'fnbr'.
KILL	Deletes the file specified by file\$ from the SD card.
LINE INPUT	Reads entire line from the file previously opened for INPUT as 'nbr'.
MKDIR	Make, or create, the directory 'dir\$' on the SD card.
MM.DRIVES\$	The current default drive returned as a string containing either "A:" or "B:".
MM.ERRNO	Is set to the error number if a statement involving the SD card fails or zero if the operation succeeds.
NAME	Rename a file or a directory on the SD card from 'old\$' to 'new\$'
OPEN	Opens a file for reading or writing.
OPTION ERROR	Sets the treatment for errors in file input/output.
PRINT	Same as PRINT except that the output is directed to a file previously opened for OUTPUT or APPEND as 'nbr'. See the OPEN command.
RMDIR	Remove, or delete, the directory 'dir\$' on the SD card.
WRITE	Outputs the value of each 'expression' separated by commas.
XMODEM	Transfers a file to or from a remote computer using the XModem protocol.

Input and Output

BYTE2NUM	Return the number created by storing the four arguments as consecutive bytes.
CAN	Controller Area Network commands.
CLOSE	Close the serial port(s) previously opened with the file number 'nbr'.
CLOSE CONSOLE	Will close a serial port that had been previously opened as the console.
I2CDIS	Disables the slave I2C module and returns the external I/O pins 12 and 13 to a "not configured" state.
I2CEN	Enables the I2C module in master mode.
I2CRCV	Receive data from the I2C slave device with the optional ability to send some data first.
I2CSEND	Send data to the I2C slave device.
I2CSDIS	Disables the slave I2C module and returns the external I/O pins 12 and 13 to a "not configured" state.
I2CSEN	Enables the I2C module in slave mode.
I2CSRCV	Receive data from the I2C master device.
I2CSSEND	Send the data to the I2C master.
INPUT\$	Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number 'fnbr'.
LOC	Will return the number of bytes waiting in the receive buffer of a serial port.
LOF	Will return the space (in bytes) remaining in the transmit buffer of a serial port.
MM.I2C	Is set to indicate the result of an I2C operation.
MM.OW	MM.OW variable is set to 1 = OK (one-wire device presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).
NUM2BYTE	Convert 'number' to four numbers containing the four separate bytes of 'number' (MMBasic numbers are stored as the C float type and are four bytes in length).
OPEN	Will open a serial port for reading and writing.
OPEN AS CONSOLE	A serial port can be opened with "AS CONSOLE".
OWCRC8	Processes the cdata and returns the 8 bit CRC.
OWCRC16	Processes the cdata and returns the 16 bit CRC.
OWREAD	Read from a one-wire device.
OWRESET	Reset a one-wire device.
OWSEARCH	Search for a one-wire device.
OWWRITE	Write to a one-wire device.

PEEK	Will return a byte within the PIC32 virtual memory space.
PIN	For a 'pin' configured as digital output this will set the output to low or high. PIN returns the value on the external I/O 'pin'.
PLAYMOD	Play synthesised music or sound effects.
PLAYMOD STOP	The command PLAYMOD STOP will immediately halt any music or sound effect that is currently playing.
POKE	Will set a byte within the PIC32 virtual memory space.
PULSE	Will generate a pulse on 'pin' with duration of 'width' mS.
PWM	Generate a pulse width modulated (PWM) output for driving analogue circuits.
PWM STOP	Stop PWM output.
SETPIN	Will configure the external I/O 'pin' according to 'cfg'.
SOUND	Generate a single tone of 'freq' (between 20Hz and 1MHz) for 'dur' milliseconds.
SPI	Sends and receives a byte using the SPI protocol with MMBasic as the master.
TONE	Generates a continuous sine wave on the sound output.
TONE STOP	The command TONE STOP will immediately halt the tone output.

Language Implementation Characteristics

Maximum length of a command line is 255 characters.

Maximum length of a variable name or a label is 32 characters.

Maximum number of dimensions to an array is 8.

Maximum number of arguments to commands that accept a variable number of arguments is 50.

Maximum number of user defined subroutines and functions (combined): 64

Numbers are stored and manipulated as single precision floating point numbers. The maximum number that can be represented is 3.40282347e+38 and the minimum is 1.17549435e-38

The range of integers (whole numbers) that can be manipulated without loss of accuracy is ±16777100.

The timer is accessed using a float, so it is limited to 16777100 before there is a loss of accuracy.

Maximum string length is 255 characters.

Maximum line number is 65000.

Maximum length of a file path name (including the directory path) is 255 characters.

Maximum number of files simultaneously open is 10 on the SD card and one on the internal flash drive (A:).

Maximum SD card size is 2GB formatted with FAT16 or 2TB formatted with FAT32.

Size of the internal flash drive (A:) is 190KB on the monochrome Maximite (less with color or CAN).

Maximum size of a loadable video font is 64 pixels high x 255 pixels wide and 256 characters.

MMBasic language constructs

The language constructs of MMBasic include the BASIC elements needed to perform loops (DO) and to perform branching (GOTO/GOSUB) based on tests of data (IF/THEN). These elements also include DATA/READ statements for data stored within the program itself.

Functions and subroutines are also supported in this BASIC implementation. Variables can be local to functions.

Timing routines are supported allowing the programmer to make use of a tick-based interrupt routine.

STRUCTURED STATEMENTS

MMBasic supports a number of modern structured statements.

The DO WHILE ... LOOP command and its variants make it easy to build loops without using the GOTO statement. Defined subroutines and functions make it easy to add your own commands to MMBasic.

The IF... THEN command can span many lines with ELSEIF ... THEN, ELSE and ENDIF statements as required and also spaced over many lines.

```
IF <condition> THEN      ' start a multiline IF
    <statements>
ELSEIF <condition> THEN  ' the ELSEIF is optional
    <statements>
ELSE                      ' the ELSE is optional
    <statements>
ENDIF                     ' must be used to terminate the IF
```

Defined Subroutines and Functions

Defined subroutines and functions are useful features to help in organizing programs so that they are easy to modify and read. A defined subroutine or function is simply a block of programming code that is contained within a module and can be called from anywhere within your program. It is the same as if you have added your own command or function to the language.

For example, assume that you would like to have the command FLASH added to MMBasic, its job would be to flash the power light on the Maximite.

You could define a subroutine like this:

```
Sub FLASH
    Pin(0) = 1
    Pause 100
    Pin(0) = 0
End Sub
```

Then, in your program you just use the command FLASH to flash the power LED.

For example:

```
IF A <= B THEN FLASH
```

If the FLASH subroutine was in program memory you could even try it out at the command prompt, just like any command in MMBasic. The definition of the FLASH subroutine can be anywhere in the program but typically it is at the start or end. If MMBasic runs into the definition while running your program it will simply skip over it.

Subroutine Arguments

Defined subroutines can have arguments (sometimes called parameter lists). In the definition of the subroutine they look like this:

```
Sub MYSUB (arg1, arg2$, arg3)
    <statements>
    <statements>
End Sub
```

And when you call the subroutine you can assign values to the arguments.

For example:

```
MYSUB 23, "Cat", 55
```

Inside the subroutine arg1 will have the value 23, arg2\$ the value of "Cat", and so on. The arguments act like ordinary variables but they exist only within the subroutine and will vanish when the subroutine ends. You can have variables with the same name in the main program and they will be different from arguments defined for the subroutine (at the risk of making debugging harder).

When calling a subroutine you can supply less than the required number of values.

For example:

```
MYSUB 23
```

In that case the missing values will be assumed to be either zero or an empty string. For example, in the above case arg2\$ will be set to "" and arg3 will be set to zero. This allows you to have optional values and, if the value is not supplied by the caller, you can take some special action.

You can also leave out a value in the middle of the list and the same will happen.

For example:

```
MYSUB 23, , 55
```

Will result in arg2\$ being set to "".

Local Variables

Inside a subroutine you will need to use variables for various tasks. In portable code you do not want the name you chose for such a variable to clash with a variable of the same name in the main program. To this end you can define a variable as LOCAL.

For example, this is our FLASH subroutine but this time we have extended it to take an argument (nbr) that specifies how many times to flash the LED.

```
Sub FLASH ( nbr )
    Local count
    For count = 1 To nbr
        Pin(0) = 1
        Pause 100
        Pin(0) = 0
        Pause 150
    Next count
End Sub
```

The counting variable (count) is declared as local which means that (like the argument list) it only exists within the subroutine and will vanish when the subroutine exits. You can have a variable called count in your main program and it will be different from the variable count in your subroutine.

If you do not declare the variable as local it will be created within your program and be visible in your main program and subroutines, just like a normal variable.

You can define multiple items with the one LOCAL command. If an item is an array the LOCAL command will also dimension the array (ie, you do not need the DIM command).

For example:

```
LOCAL NBR, STR$, ARR(10, 10)
```

You can also use local variables in the target for GOSUBs.

For example:

```
GOSUB MySub
    ...
MySub:
    LOCAL X, Y
```

```
FOR X = 1 TO ...
FOR Y = 5 TO ...
<statements>
RETURN
```

The variables X and Y will only be valid until the RETURN statement is reached and will be different from variables with the same name in the main body of the program.

Defined Functions

Defined functions are similar to defined subroutines with the main difference being that the function is used to return a value in an expression.

For example, if you wanted a function to select the maximum of two values you could define:

```
Function Max(a, b)
    If a > b
        Max = a
    Else
        Max = b
    EndIf
End Function
```

Then you could use it in an expression:

```
SetPin 1, 1 : SetPin 2, 1
Print "The highest voltage is" Max(Pin(1), Pin(2))
```

The rules for the argument list in a function are similar to subroutines. The only difference is that brackets are required around the argument list when you are calling a function (they are optional when calling a subroutine).

To return a value from the function you assign a value to the function's name within the function. If the function's name is terminated with a \$ the function will return a string, otherwise it will return a number.

Within the function the function's name acts like a standard variable.

As another example, let us say that you need a function to format time in the AM/PM format:

```
Function MyTime$(hours, minutes)
    Local h
    h = hours
    If hours > 12 Then h = h - 12
    MyTime$ = Str$(h) + ":" + Str$(minutes)
    If hours <= 12 Then
        MyTime$ = MyTime$ + "AM"
    Else
        MyTime$ = MyTime$ + "PM"
```

```
    EndIf  
End Function
```

As you can see, the function name is used as an ordinary local variable inside the subroutine. It is only when the function returns that the value assigned to MyTime\$ is made available to the expression that called it. This example also illustrates that you can use local variables within functions just like subroutines.

Passing Arguments by Reference

If you use an ordinary variable (ie, not an expression) as the value when calling a subroutine or a function, the argument within the subroutine/function will point back to the variable used in the call and any changes to the argument in your routine will also be made to the supplied variable. This is called passing arguments by reference.

For example, you might define a subroutine to swap two values, as follows:

```
Sub Swap a, b
    Local t
    t = a
    a = b
    b = t
End Sub
```

In your calling program you would use variables for both arguments:

```
Swap nbr1, nbr2
```

And the result will be that the values of nbr1 and nbr2 will be swapped.

Unless you need to return a value via the argument you should not use an argument as a general purpose variable inside a subroutine or function. This is because another user of your routine may unwittingly use a variable in their call and that variable will be "magically" changed by your routine. It is much safer to assign the argument to a local variable and manipulate that instead.

ADDITIONAL NOTES

There can be only one END SUB or END FUNCTION for each definition of a subroutine or function. To exit early from a subroutine (ie, before the END SUB command has been reached) you can use the EXIT SUB command. This has the same effect as if the program reached the END SUB statement. Similarly you can use EXIT FUNCTION to exit early from a function.

You cannot use arrays in a subroutine or function's argument list however the caller can use them.

For example, this is a valid way of calling the Swap subroutine (discussed above):

```
Swap dat(i), dat(I + 1)
```

This type of construct is often used in sorting arrays.

The use of defined subroutines and functions should reduce the need to add specialized features to MMBasic. For instance, there have been a few requests to add bit shifting functions to the language. Now you can do that yourself... this is the right shift function:

```
Function RShift(nbr, bits)
  If nbr < 0 or bits < 0 THEN ERROR "Invalid argument"
  RShift = nbr\ (2^bits)
End Function
```

You can now use this function as if it is a part of the language:

```
a = &b11101001
b = RShift(a, 3)
```

After running this fragment of code the variable b would have the binary value of 11101.

The defined subroutine and function is intended to be a portable lump of code that you can insert into any program. This is why the full screen editor has the CTRL-F keys for inserting another file. The idea is that you can keep your defined routines in a file and whenever you need them you can quickly insert them using CTRL-F.

So, it would be easy to create a library of bit manipulation functions like that described above and insert them into any program when needed.

Logical operators	
NOT	Logical inverse of the value on the right
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	Equality
AND OR XOR	Conjunction, disjunction, exclusive OR.

The operators AND, OR and XOR are bitwise operators. For example PRINT 3 AND 6 will output 2.

The other logical operations result in the number 0 (zero) for false and 1 for true. For example the statement PRINT 4 >= 5 will print the number zero on the output and the expression A = 3 > 2 will store +1 in A.

The NOT operator is highest in precedence so it will bind tightly to the next value. For normal use the expression to be negated should be placed in brackets. For example, IF NOT (A = 3 OR A = 8) THEN ...

More than one statement can be on a single program line. For example:

```
' Data lines  
SETPIN 25, 8 : SETPIN 26, 8 : SETPIN 27, 8 : SETPIN 28, 8
```

Each statement must be separated by a colon. In the example above there are four SETPIN statements that each set a different line to be outputs. The program executes the statements from left to right.

Naming Conventions

Command names, function names, labels, variable names, file names, etc are not case sensitive, so that "Run" and "RUN" are equivalent and "dOO" and "Doo" refer to the same variable.

There are two types of variable: numeric which stores a floating point number (eg, 45.386), and string which stores a string of characters (eg, "Tom"). String variable names are terminated with a \$ symbol (eg, name\$) while numeric variables are not.

Variable names and labels can start with an alphabetic character or underscore and can contain any alphabetic or numeric character, the period (.) and the underscore (_). They may be up to 32 characters long. A variable name or a label must not be the same as a function or one of the following keywords: THEN, ELSE, GOTO, GOSUB, TO, STEP, FOR, WHILE, UNTIL, LOAD, MOD, NOT, AND, OR, XOR. Eg, step = 5 is illegal as STEP is a keyword. In addition, a label cannot be the same as a command name.

Timing

You can freeze program execution for a number of milliseconds using PAUSE. MMBasic also maintains an internal stopwatch function (the TIMER function) which counts up in milliseconds. You can reset the timer to zero or any other number by assigning a value to the TIMER.

Using SETTICK in the Maximite versions you can setup a “tick” which will generate a regular interrupt with a period from one millisecond to over a month. See Interrupts.

Interrupts

The timer interrupt will cause an immediate branch to a specified line number or label (similar to a GOSUB). Return from an interrupt is via the IRETURN statement. All statements (including GOSUB/RETURN) can be used within an interrupt.

During processing of an interrupt all other interrupts are disabled until the interrupt routine returns with an IRETURN.

A periodic interrupt (or regular “tick”) with a period specified in milliseconds can be setup using the SETTICK statement. This interrupt has the lowest priority.

Interrupts can occur at any time but they are disabled during INPUT statements. If you need to get input from the keyboard while still accepting interrupts you should use the INKEY\$ function. When using interrupts the main program is completely unaffected by the interrupt activity unless a variable used by the main program is changed during the interrupt.

For most programs MMBasic will respond to an interrupt in under 100µS. To prevent slowing the main program by too much an interrupt should be short and execute the IRETURN statement as soon as possible.

Also remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non-intuitive bugs.

CLEAR

CLEAR

Delete all variables and recover the memory used by them.

See ERASE for deleting specific array variables.

```
forget$ = "I have short term memory"  
remember = 42  
PRINT forget$  
I have short term memory  
PRINT remember  
42  
  
CLEAR  
PRINT forget$  
  
PRINT remember  
0
```

See also:

DATA

ERASE

READ

DIM

LET

RESTORE

CONTINUE

CONTINUE

Resume running a program that has been stopped by an END statement, an error, or CTRL-C (can be redefined by OPTION BREAK). The program will restart with the next statement following the previous stopping point.

See also:

OPTION BREAK

RUN

DATA

DATA constant[,constant]...

Stores numerical and string constants to be accessed by READ.

String constants do not need to be quoted unless they contain significant spaces, the comma or a keyword (such as THEN, WHILE, etc).

Numerical constants can also be expressions such as 5 * 60.

```
10 Data 1, 2, 3, 4, 5, 6, 7, 8, 9
20 Dim var(9)
30 For loop = 1 To 9
40 Read var(loop)
50 Next loop
60 For loop = 1 To 9
70 Print var(loop)
80 Next loop
> run
```

```
1
2
3
4
5
6
7
8
9
```

See also:

CLEAR

ERASE

READ

DIM

LET

RESTORE

DIM

DIM variable(elements...) [variable(elements...)]...

Specifies variables that have more than one element in a single dimension, i.e., arrayed variables.

```
10 Data 1, 2, 3, 4, 5, 6, 7, 8, 9
20 Dim var(9)
30 For loop = 1 To 9
40 Read var(loop)
50 Next loop
60 For loop = 1 To 9
70 Print var(loop)
80 Next loop
> run
```

```
1
2
3
4
5
6
7
8
9
```

See also:

CLEAR

ERASE

READ

DATA

LET

RESTORE

OPTION BASE

DO LOOP

DO

<statements>

LOOP

This structure will loop forever; the EXIT command can be used to terminate the loop or control must be explicitly transferred outside of the loop by commands like GOTO or RETURN (if in a subroutine).

See also:

DO LOOP UNTIL

EXIT FOR

NEXT

DO WHILE LOOP

FOR TO STEP

WHILE WEND

EXIT

LOOP UNTIL

DO LOOP UNTIL

DO

<statements>

LOOP UNTIL expression

Loops until the expression following UNTIL is true. Because the test is made at the end of the loop the statements inside the loop will be executed at least once even if the expression is false.

See also:

DO LOOP

EXIT FOR

NEXT

DO WHILE LOOP

FOR TO STEP

WHILE WEND

EXIT

LOOP UNTIL

DO WHILE LOOP

DO WHILE expression

<statements>

LOOP

Loops while "expression" is true (this is equivalent to the older WHILE-WEND loop, also implemented in MMBasic). If, at the start, the expression is false the statements in the loop will not be executed, even once.

See also:

DO LOOP

EXIT FOR

NEXT

DO LOOP UNTIL

FOR TO STEP

WHILE WEND

EXIT

LOOP UNTIL

ELSE

Introduces a default condition in a multi-line IF statement.

See the multi-line IF statement for more details.

See also:

ELSEIF THEN

ENDIF

IF THEN ELSE

IF THEN GOTO

IF THEN ELSE ELSIF

ENDIF

ELSEIF THEN

ELSEIF expression THEN

Introduces a secondary condition in a multi-line IF statement.

See the multi-line IF statement for more details.

See also:

ELSE

ENDIF

IF THEN ELSE

IF THEN GOTO

IF THEN ELSE ELSIF

ENDIF

END

END

End the running program and return to the command prompt.

See also:

CONTINUE

RUN

END FUNCTION

END FUNCTION

Marks the end of a user defined function.

Each sub must have one and only one matching END FUNCTION statement. Use EXIT FUNCTION if you need to return from a subroutine from within its body.

Only one space is allowed between END and FUNCTION.

See also:

END SUB

EXIT SUB

LOCAL

EXIT FUNCTION

FUNCTION

SUB

END SUB

END SUB

Marks the end of a user defined subroutine. See the SUB command.

Each sub must have one and only one matching END SUB statement. Use EXIT SUB if you need to return from a subroutine from within its body.

Only one space is allowed between END and SUB.

See also:

END FUNCTION

EXIT SUB

LOCAL

EXIT FUNCTION

FUNCTION

SUB

ENDIF

ENDIF

Terminates a multiline IF statement.

See the multiline IF statement for more details.

See also:

ELSE

ELSEIF THEN

IF THEN ELSE

IF THEN GOTO

IF THEN ELSE ELSIF

ENDIF

ERASE

ERASE variable [,variable]...

Deletes arrayed variables and frees up the memory.

Use CLEAR to delete all variables including all arrayed variables.

See also:

CLEAR

DIM

READ

DATA

LET

RESTORE

EXIT

EXIT

EXIT by itself will terminate a DO...LOOP.

See also:

DO LOOP

EXIT FOR

NEXT

DO LOOP UNTIL

FOR TO STEP

WHILE WEND

DO WHILE LOOP

LOOP UNTIL

EXIT FOR

EXIT FOR

EXIT FOR will terminate a FOR...NEXT loop.

Only one space is allowed between the two words.

See also:

DO LOOP

EXIT

NEXT

DO LOOP UNTIL

FOR TO STEP

WHILE WEND

DO WHILE LOOP

LOOP UNTIL

EXIT FUNCTION

EXIT FUNCTION

EXIT FUNCTION provides an early exit from a defined function.

Only one space is allowed between the two words.

See also:

END FUNCTION

EXIT SUB

LOCAL

END SUB

FUNCTION

SUB

EXIT SUB

EXIT SUB

EXIT SUB provides an early exit from a defined subroutine.

Only one space is allowed between the two words.

See also:

END FUNCTION

EXIT FUNCTION

LOCAL

END SUB

FUNCTION

SUB

FOR TO STEP

FOR counter = start TO finish [STEP increment]

Initiates a FOR-NEXT loop with the 'counter' initially set to 'start' and incrementing in 'increment' steps (default is 1) until 'counter' equals 'finish'.

The 'increment' must be an integer but may be negative.

```
10 Data 1, 2, 3, 4, 5, 6, 7, 8, 9
20 Dim var(9)
30 For loop = 1 To 9
40 Read var(loop)
50 Next loop
60 For loop = 1 To 9
70 Print var(loop)
80 Next loop
> run
```

```
1
2
3
4
5
6
7
8
9
```

See also:

DO LOOP

DO WHILE LOOP

LOOP UNTIL

DO LOOP UNTIL

EXIT

NEXT

EXIT FOR

WHILE WEND

FUNCTION

FUNCTION xxx (arg1 [,arg2, ...])

<statements>

xxx = <return value>

Defines a callable function. This is the same as adding a new function to MMBasic while it is running your program.

'xxx' is the function name and it must meet the specifications for naming a variable. 'arg1', 'arg2', etc are the arguments or parameters to the function.

To set the return value of the function you assign the value to the function's name.

```
FUNCTION SQUARE (a)
    SQUARE = a * a
END FUNCTION
```

Every definition must have one END FUNCTION statement. When this is reached the function will return its value to the expression from which it was called. The command EXIT FUNCTION can be used for an early exit.

You use the function by using its name and arguments in a program just as you would a normal MMBasic function.

```
PRINT SQUARE (56.8)
```

When the function is called each argument in the caller is matched to the argument in the function definition. These arguments are available only inside the function.

Functions can be called with a variable number of arguments. Any omitted arguments in the function's list will be set to zero or a null string.

Arguments in the caller's list that are a variable (ie, not an expression or constant) will be passed by reference to the function. This means that any changes to the corresponding argument in the function will also be copied to the caller's variable.

You must not jump into or out of a function using commands like GOTO, GOSUB, interrupts, etc. Doing so will have undefined side effects.

See also:

END FUNCTION

EXIT FUNCTION

LOCAL

END SUB

EXIT SUB

SUB

GOSUB

GOSUB target

Initiates a subroutine call to the target which can be a line number or a label.

The subroutine must end with RETURN.

See also:

GOTO

ON GOTO|GOSUB

RETURN

GOTO

GOTO target

Branches program execution to the target which can be a line number or a label.

See also:

GOSUB

ON GOTO|GOSUB

RETURN

IF THEN ELSE

IF expr THEN statement [ELSE statement]

Evaluates the expression 'expr' and performs the THEN statement if it is true or skips to the next line if false. The optional ELSE statement is the reverse of the THEN test. This type of IF statement is all on one line.

See also:

ELSE

ELSEIF THEN

ENDIF

IF THEN GOTO

**IF THEN ELSE ELSIF
ENDIF**

IF THEN GOTO

IF condition THEN GOTO linenbr | label

Evaluates the expression 'expr' and performs the THEN statement if it is true or skips to the next line if false. The optional ELSE statement is the reverse of the THEN test. This type of IF statement is all on one line.

The 'THEN statement' construct can be also replaced with:

```
GOTO linenumber | label
```

See also:

ELSE

ELSEIF THEN

ENDIF

IF THEN ELSE

IF THEN ELSE ELSIF

ENDIF

IF THEN ELSE ELSEIF ENDIF

```
IF expression THEN
<statements>
[ELSE
<statements>]
[ELSEIF expression THEN
<statements>]
ENDIF
```

Multiline IF statement with optional ELSE and ELSEIF cases and ending with ENDIF. Each component is on a separate line.

Evaluates 'expression' and performs the statement(s) following THEN if the expression is true or optionally the statement(s) following the ELSE statement if false.

The ELSEIF statement (if present) is executed if the previous condition is false and it starts a new IF chain with further ELSE and/or ELSEIF statements as required.

One ENDIF is used to terminate the multiline IF.

See also:

ELSE	ENDIF	IF THEN GOTO
ELSEIF THEN	IF THEN ELSE	

IRETURN

IRETURN

Returns from an interrupt. The next statement to be executed will be the one that was about to be executed when the interrupt was detected.

See also:

SETPIN

SETTICK

LET

LET variable = expression

Assigns the value of 'expression' to the variable. LET is automatically assumed if a statement does not start with a command.

See also:

CLEAR

DIM

READ

DATA

ERASE

RESTORE

LOCAL

LOCAL variable [, variables]

Defines a list of variable names as local to the subroutine or function.

'variable' can be an array and the array will be dimensioned just as if the DIM command had been used.

A local variable will only be visible within the procedure and will be deleted (and the memory reclaimed) when the procedure returns.

If the local variable has the same name as a global variable (used before any subroutines or functions were called) the global variable will be hidden by the local variable while the procedure is executed.

See also:

END FUNCTION

EXIT FUNCTION

FUNCTION

END SUB

EXIT SUB

SUB

LOOP UNTIL

LOOP [UNTIL expression]

Terminates a 'DO' program loop.

See also:

DO LOOP

EXIT

NEXT

DO LOOP UNTIL

EXIT FOR

WHILE WEND

DO WHILE LOOP

FOR TO STEP

NEXT

NEXT [counter-variable] [, counter-variable]...

NEXT comes at the end of a FOR-NEXT loop.

The 'counter-variable' specifies exactly which loop is being operated on.

If no 'counter-variable' is specified the NEXT will default to the innermost loop. It is also possible to specify multiple counter-variables as in:

```
NEXT x, y, z
```

```
10 Data 1, 2, 3, 4, 5, 6, 7, 8, 9
20 Dim var(9)
30 For loop = 1 To 9
40 Read var(loop)
50 Next loop
60 For loop = 1 To 9
70 Print var(loop)
80 Next loop
> run
```

```
1
2
3
4
5
6
7
8
9
```

See also:

DO LOOP

EXIT

LOOP UNTIL

DO LOOP UNTIL

EXIT FOR

WHILE WEND

DO WHILE LOOP

FOR TO STEP

ON nbr GOTO | GOSUB

ON nbr GOTO | GOSUB target[,target, target,...]

ON either branches (GOTO) or calls a subroutine (GOSUB) based on the rounded value of variable; if it is 1, the first target is called, if 2, the second target is called, etc.

Target can be a line number or a label.

See also:

GOSUB

GOTO

RETURN

OPTION BASE

OPTION BASE 0/1

Sets the lowest value for array subscripts to either 0 or 1. The default is 0. This must be used before any arrays are declared.

See also:

DIM

OPTION Fnn

OPTION Fnn string\$

Sets the programmable function key 'Fnn' to the contents of 'string\$'. 'Fnn' is the function key F1 to F12. Maximum string length is 12 characters. 'string\$' can also be an expression which will be evaluated at the time of running the OPTION command. This is most often used to append the ENTER key (chr\$(13)), or double quotes (chr\$(34)).

For example:

```
OPTION F1 "RUN" + CHR$(13)
OPTION F6 "SAVE " + CHR$(34)
OPTION F10 "ENDIF"
```

Normally these commands are included in an AUTORUN.BAS file (see OPTION PROMPT for an example).

OPTION PROMPT

OPTION PROMPT string\$

Sets the command prompt to the contents of 'string\$' (which can also be an expression which will be evaluated when the prompt is printed).

For example:

```
OPTION PROMPT "Ok "
OPTION PROMPT TIME$ + ":" "
OPTION PROMPT CWD$ + ":" "
```

Maximum length of the prompt string is 48 characters. The prompt is reset to the default ("> ") on power up but you can automatically set it by saving the following example program as "AUTORUN.BAS" on the internal flash drive A:.

```
OPTION PROMPT "My prompt: "
My prompt:
```

OPTION USB

OPTION USB ON | OFF

Turn the USB output off and on. This disables/enables the output from commands like PRINT from being sent out on the USB interface. It does not affect the reception of characters from the USB interface.

Normally this is used when a program wants to separately display data on the USB and video interfaces. This option is always reset to ON at the command prompt.

PAUSE

PAUSE delay

Halt execution of the running program for 'delay' mS.

This can be a fraction. For example, 0.2 is equal to 200 µS.

The maximum delay is 4294967295 mS (about 49 days).

See also:

TIMER

SETTICK

RANDOMIZE

RANDOMIZE nbr

Seeds the random number generator with 'nbr'. To generate a different random sequence each time you must use a different value for 'nbr'. One good way to do this is use the TIMER function.

```
RANDOMIZE TIMER
```

See also:

RND

READ

READ variable[, variable]...

Reads values from DATA statements and assigns these values to the named variables. Variable types in a READ statement must match the data types in DATA statements as they are read.

```
10 Data 1, 2, 3, 4, 5, 6, 7, 8, 9
20 Dim var(9)
30 For loop = 1 To 9
40 Read var(loop)
50 Next loop
60 For loop = 1 To 9
70 Print var(loop)
80 Next loop
> run
```

```
1
2
3
4
5
6
7
8
9
```

See also:

CLEAR

DIM

LET

DATA

ERASE

RESTORE

REM or '

REM or '

REM allows remarks to be included in a program.

Note the Microsoft style use of the single quotation mark to denote remarks is also supported and is preferred.

Starts a comment and any text following it will be ignored. Comments can be placed anywhere on a line.

RESTORE

RESTORE

Resets the line and position counters for DATA and READ statements to the top of the program file.

See also:

CLEAR

DIM

LET

DATA

ERASE

READ

RETURN

RETURN

RETURN concludes a subroutine called by GOSUB and returns to the statement after the GOSUB.

See also:

GOSUB

GOTO

ON GOTO|GOSUB

SETTICK

SETTICK period, target

This will setup a periodic interrupt (or "tick"). The time between interrupts is 'period' milliseconds and 'target' is the line number or label of the interrupt routine.

See also IRETURN to return from the interrupt.

The period can range from 1 to 4294967295 mSec (about 49 days).

This interrupt can be disabled by setting 'line' to zero (ie, SETTICK 0, 0).

See also:

IRETURN

PAUSE

TIMER

SUB

SUB xxx (arg1 [,arg2, ...])

<statements>

<statements>

END SUB

Defines a callable subroutine. This is the same as adding a new command to MMBasic while it is running your program.

'xxx' is the subroutine name and it must meet the specifications for naming a variable. 'arg1', 'arg2', etc are the arguments or parameters to the subroutine.

Every definition must have one END SUB statement. When this is reached the program will return to the next statement after the call to the subroutine. The command EXIT SUB can be used for an early exit.

You use the subroutine by using its name and arguments in a program just as you would a normal command.

```
MySub a1, a2
```

When the subroutine is called each argument in the caller is matched to the argument in the subroutine definition. These arguments are available only inside the subroutine. Subroutines can be called with a variable number of arguments. Any omitted arguments in the subroutine's list will be set to zero or a null string.

Arguments in the caller's list that are a variable (ie, not an expression or constant) will be passed by reference to the subroutine. This means that any changes to the corresponding argument in the subroutine will also be copied to the caller's variable and therefore may be accessed after the subroutine has ended.

Brackets around the argument list in both the caller and the definition are optional.

See also:

END FUNCTION

EXIT FUNCTION

FUNCTION

END SUB

EXIT SUB

LOCAL

TIMER (command/function)

TIMER = msec

Resets the timer to a number of milliseconds. Normally this is just used to reset the timer to zero but you can set it to any positive integer.

TIMER

TIMER returns the elapsed time in milliseconds (eg, 1/1000 of a second) since reset. If not specifically reset this count will wrap around to zero after 49 days.

The timer is reset to zero on power up and you can also reset it by using TIMER as a command.

See also:

PAUSE

SETTICK

WHILE WEND

WHILE expression
<statements>
WEND

WHILE initiates a WHILE-WEND loop. The loop ends with WEND, and execution reiterates through the loop as long as the 'expression' is true.

This construct is included for Microsoft compatibility. New programs should use the DO ... LOOP construct.

See also:

DO LOOP	EXIT	LOOP UNTIL
DO LOOP UNTIL	EXIT FOR	NEXT
DO WHILE LOOP	FOR TO STEP	

MMBasic editing, programming, and debugging

MMBasic supports program entry and editing commands, as well as commands to save/load programs from the two "drives" that the hardware supports.

COMMAND AND PROGRAM INPUT

At the command prompt (the greater than symbol, ie, >) you can enter a command line followed by the enter key and it will be immediately run. This is useful for testing commands and their effects.

Line numbers are optional. If you use them you can enter a program at the command line by preceding each program line with a line number however; it is recommended that the full screen editor (the EDIT command) be used to enter and edit programs.

When entering a line at the command prompt the line can be edited using the arrow keys to move along the line, the Delete key to delete a character and the Insert key to switch between insert and overwrite. The up arrow key will move through a list of previously entered commands which can be edited and reused.

A program held in memory can be listed with LIST, run using the RUN command and cleared with the NEW command. You can interrupt MMBasic at any time by typing CTRL C (can be redefined with OPTION BREAK) and control will be returned to the prompt.

KEYBOARD/DISPLAY

Input can come from either a keyboard or from a computer using a terminal emulator via the USB or serial interfaces. Both the keyboard and the USB interface can be used simultaneously and can be detached or attached at any time without affecting a running program.

Output will be simultaneously sent to the USB interface and the video display (VGA or composite). Either can be attached or removed at any time.

LINE NUMBERS, PROGRAM STRUCTURE, AND EDITING

In version 3.0 of MMBasic and later versions the use of line numbers is optional. MMBasic will still run programs written using line numbers, but it is recommended that new programs avoid them.

The structure of a program line is:

```
[line-number] [label:] command arguments [: command arguments] ...
```

Instead of using a line number a label can be used to mark a line of code. A label has the same specifications (length, character set, etc) as a variable name but it

cannot be the same as a command name. When used to label a line the label must appear at the beginning of a line but after a line number (if used), and be terminated with a colon character (:). Commands such as GOTO can use labels instead of line numbers to identify the destination (in that case the label does not need to be followed by the colon character). For example:

```
GOTO xxxx
-
-
xxxx: PRINT "We have jumped to here"
```

MMBasic finds a label much faster than a line number so labels are recommended for new programs.

Multiple commands separated by a colon can be entered on the one line (as in INPUT A : PRINT B).

Long programs (with or without line numbers) can be sent via USB to MMBasic using the XMODEM command or the AUTO command.

STORAGE COMMANDS AND FUNCTIONS

A program can be saved to either drive using the SAVE command. It can be reloaded using LOAD or merged with the current program using MERGE. A saved program can also be loaded and run using the RUN command. The RUN command can also be used within a running program, which enables one program to load and transfer control to another.

You can list the programs stored on a drive with the FILES command, delete them using KILL and rename them using NAME. On an SD card the current working directory can be changed using CHDIR. A new directory can be created with MKDIR or an old one deleted with RMDIR.

Whenever specified a file name can be a string constant (ie, enclosed in double quotes) or a string variable. This means you must use double quotes if you are directly specifying a file name. Eg, RUN "TEST.BAS"

On the SD card programs are stored using standard text and can be read and edited in Windows, Apple Mac, Linux, etc.

REAL-TIME CALENDAR/CLOCK

You can get the current date and time using the DATE\$ and TIME\$ functions and you can set them by assigning the new date and time to them. Hardware (CGCOLORMAX) with the battery backed clock will never lose the time. On Maximites without RTC hardware (CGMMSTICK) the calendar will start from midnight 1st Jan 2000 on power up.

Full Screen Editor

An important productivity feature of MMBasic is the full screen editor. It will work using an attached video screen (VGA or composite) and over USB with a VT100 compatible terminal emulator.

The full screen editor is invoked with the EDIT command. If you just type EDIT without anything else the editor will automatically start editing whatever is in program memory. If program memory is empty you will be presented with an empty screen.

The cursor will be automatically positioned at the last place that you were editing at and, if your program had just been stopped by an error, the cursor will be positioned at the line that caused the error.

You can also run the editor with a file name (eg, EDIT "file.ext") and the editor will edit that file while leaving program memory untouched. This is handy for examining or changing files on the disk without disturbing your program.

If you are used to an editor like Notepad you will find that the operation of the full screen editor is familiar. The arrow keys will move your cursor around in the text, home and end will take you to the beginning or end of the line. Page up and page down will do what their titles suggest. The delete key will delete the character at the cursor and backspace will delete the character before the cursor. The insert key will toggle between insert and overtype modes.

About the only unusual key combination is that two home key presses will take you to the start of the program and two end key presses will take you to the end.

At the bottom of the screen the status line will list the various function keys used by the editor and their action.

In more details these are:

ESC	This will cause the editor to abandon all changes and return to the command prompt with the program memory unchanged. If you have changed the text you will be asked if this is really what you want to do.
F1: SAVE	This will save the program to program memory and return to the command prompt. If you are editing a disk file it will save that file to the disk.
F2: RUN	This will save the program to program memory and immediately run it.
F3: FIND	This will prompt for the text that you want to search for. When you press enter the cursor will be placed at the start of the first entry found.
SHIFT-F3	Once you have used the search function you can repeatedly search for the same text by pressing SHIFT-F3.
F4: MARK	This is described in detail below.
F5: PASTE	This will insert (at the current cursor position) the text that had been previously cut or copied (see below).

If you pressed the mark key (F4) the editor will change to the mark mode. In this mode you can use the arrow keys to mark a section of text which will be highlighted in reverse video. You can then delete, cut or copy the marked text. In this mode the status line will change to show the functions of the function keys in the mark mode. These keys are:

ESC	Will exit mark mode without changing anything.
F4: CUT	Will copy the marked text to the clipboard and remove it from the program.
F5: COPY	Will just copy the marked text to the clipboard.
DELETE	Will delete the marked text leaving the clipboard unchanged.

The best way to learn the full screen editor is to simply fire it up and experiment.

The editor is a very productive method of writing a program. Using the OPTION Fnn command you can program a function key to generate the command "EDIT" when pressed. So, with one key press you can jump into the editor where you can make a change. Then by pressing the F3 key you can save and run the program.

If your program stops with an error you can press the edit function key and be back in the editor with the cursor positioned at the line that caused the error. This edit/run/edit cycle is very fast.

If you are using the full screen editor over USB with Terra Term you must set Terra Term to a screen size of 80 characters by 36 lines.

Note that a terminal emulator can lose its position in the text with multiple fast keystrokes (like the up and down arrows). If this happens you can press the HOME key twice which will force the editor to jump to the start of the program and redraw the display.

AUTO

AUTO [start] [, increment]

Enter automatic program line entry mode.

To terminate this mode use Control-C.

With no arguments this command will take lines of text from the keyboard or USB and append them to program memory without modification. This is useful for adding lines that do not have line numbers.

If 'start' is provided the lines will be prefixed with an automatically generated line number. 'start' is the starting line number and 'increment' is the step size (default 10). If the automatically generated number is the same as an existing line in memory it will be preceded by an asterisk (*). In this case pressing Enter without entering any text will preserve the line in memory and generate the next number.

AUTO provides a way to load program source through a serial connection, if console is redirected through the serial port. Transmit AUTO, followed by line end. Then transmit the text of each program line. When finished, terminate AUTO mode with a Control-C. At this point the program is in MMBasic and can be run.

See also:

DELETE

LIST

RENUMBER

NEW

CHAIN

CHAIN file\$

Clear the current program from memory and run the new program ('file\$') starting with the first line.

Unlike the RUN command this command retains the current state of the program (ie, the value of variables, open files, loaded fonts, open COM ports, etc). The only exception is that any open interrupts will be automatically closed.

One program can CHAIN to another which can then chain to another (or back to the original) an unlimited number of times.

As long as a program can be broken down into modules this command allows programs of almost unlimited size to be run, even with limited memory.

Communication between the modules can be accomplished by assigning values to one or more variables which then can be examined by the new chained program.

See also:

LOAD

NEW

SAVE

RUN

CONFIG CASE

CONFIG CASE UPPER|LOWER|TITLE

The CASE setting will change the case used for listing command and function names when using the LIST command. The default is TITLE but the old standard of MMBasic can be restored using CONFIG CASE UPPER.

The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off).

The power must be cycled after changing a setting for it to take effect.

See also:

LIST

CONFIG KEYBOARD

CONFIG KEYBOARD US | UK | FR | GR | BE | IT

The KEYBOARD setting will change the keyboard layout to suit standard keyboards (US), United Kingdom (UK), French (FR), German (GR), Belgium (BE) or Italian (IT) keyboards. Default is US.

The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off).

The power must be cycled after changing a setting for it to take effect.

CONFIG TAB

CONFIG TAB 2 | 4 | 8

The TAB setting will set the spacing for the tab key. Default is 2.

The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off).

The power must be cycled after changing a setting for it to take effect.

COPYRIGHT

COPYRIGHT

List all contributors to the MMBasic and summarize the copyright statement.

See also:

MM.DEVICE\$

MM.VER

DATE\$ (command/function)

DATE\$ = "DD-MM-YY"

DATE\$ = "DD/MM/YY"

Set the date of the internal clock/calendar.

DD, MM and YY are numbers, for example: DATE\$ = "28-2-2011" The date is set to "1-1-2000" on power up.

With a CGCOLORMAX the current date will be automatically set on power up because of the RTC hardware.

DATE\$

Returns the current date based on MMBasic's internal clock as a string in the form "DD-MM-YYYY". For example, "28-02-2011".

The internal clock/calendar will keep track of the time and date including leap years.

See also:

DATE\$

TIME\$

DELETE

DELETE line

DELETE -lastline

DELETE firstline -

DELETE firstline - lastline

Deletes a program line or a range of lines.

If –lastline is used it will start with the first line in the program. If startline- is used it will delete to the end of the program.

See also:

AUTO

LIST

RENUMBER

NEW

EDIT

EDIT [filename] | [line-number]

Invoke the full screen editor. This can be used to edit either the program currently loaded in memory or a file on either drive A: or B:. It can also be used to view and edit text data files.

If EDIT is used on its own it will edit the program memory. If 'filename' is supplied the file will be edited leaving the program memory untouched.

On entry the cursor will be automatically positioned at the last line edited or, if there was an error when running the program, the line that caused the error. If 'line-number' is specified on the command line the program in memory will be edited and cursor will be placed on the line specified.

The editing keys are:

Left/right arrows Moves the cursor within the line.

Up/Down arrows Moves the cursor up or down a line.

Page Up/Down Move up or down a page of the program.

Home/End Moves the cursor to the start or end of the line. A second Home/End will move to the start or end of the program.

Delete Delete the character over the cursor. This can be the line separator character and thus join two lines.

Backspace Delete the character before the cursor.

Insert Will switch between insert and overtype mode.

Escape Key Will close the editor without saving (confirms first).

Function Key 1 Will save the edited text and exit.

Function Key 2 Will save, exit and run the program.

Function Key 3 Will invoke the search function.

SHIFT F3 Will repeat the search using the text entered at F3.

Function Key 4 Will mark text for cut or copy (see below).

Function Key 5 Will paste text previously cut or copied.

CTRL-F Will insert a file into the program being edited.

When in the mark text mode (entered with F4) the editor will allow you to use the arrow keys to highlight text which can be deleted, cut to the clipboard or simply

copied to the clipboard. The status line will change to indicate the new functions of the function keys.

While the full screen editor is running it will override the programmable function keys F1 to F5. When the editor exits all programmable functions will be restored.

The editor will work with lines wider than the screen but characters beyond the screen edge will not be visible. You can split such a line by inserting a new line character and the two lines can be later rejoined by deleting the inserted new line character.

All the editing keys work with a VT100 terminal emulator so editing can also be accomplished over a USB or serial link. The editor has been tested with Tera Term and this is the recommended software. Note that Tera Term MUST be configured for an 80 column by 36 line display.

See also:

LOAD

NEW

SAVE

RUN

ERROR

ERROR [error_msg\$]

Forces an error and terminate the program. This is normally used in debugging or to trap events that should not occur.

LIST

LIST

LIST line

LIST -lastline

LIST firstline -

LIST firstline - lastline

Lists all lines in a program line or a range of lines.

If –lastline is used it will start with the first line in the program. If startline- is used it will list to the end of the program.

See also:

AUTO

DELETE

RENUMBER

CONFIG CASE

NEW

LOAD

LOAD file\$

Loads a program called 'file\$' from the SD card into working memory. Quote marks are required around a string constant. Example: LOAD "TEST.BAS" If an extension is not specified ".BAS" will be added to the file name.

See also:

CHAIN

MERGE

RUN

EDIT

MM.FNAME\$

SAVE

LOAD

NEW

MEMORY

MEMORY

List the amount of memory currently in use.

```
15kB (18%) Program (528 lines)
23kB (28%) 52 Variables
17kB (21%) General
28kB (33%) Free
```

Program memory is cleared by the NEW command. Variable and the general memory spaces are cleared by many commands (eg, NEW, RUN, LOAD, etc) as well as the specific commands CLEAR and ERASE.

General memory is used for fonts, file I/O buffers, etc.

See also:

NEW

MERGE

MERGE file\$

Adds program lines from 'file\$' to the program in memory. Unlike LOAD, it does not clear the program currently in memory.

See also:

CHAIN

LOAD

SAVE

MM.CMDLINE\$

MM.CMDLINE\$

The contents of the command line from the implied RUN command.

See also:

RUN

MM.DEVICE\$

MM.DEVICE\$

A string representing the device or platform that MMBasic is running on. Currently this variable will contain one of the following:

"Maxomite" on the standard Maximite (CGMMSTICK).
"Colour Maximite" on the Color Maximite (CGCOLORMAX).

See also:

COPYRIGHT

MM.VER

MM.FNAME\$

MM.FNAME\$

The name of the file that will be used as the default for the SAVE command. This is set by LOAD, RUN and SAVE.

See also:

LOAD

RUN

SAVE

MM.VER

MM.VER

The version number of the firmware in the form aa.bbcc where aa is the major version number, bb is the minor version number and cc is the revision number (normally zero but A = 01, B = 02, etc).

See also:

COPYRIGHT

MM.DEVICE\$

NEW

NEW

Deletes the program in memory and clears all variables.

See also:

AUTO	LIST	RUN
CHAIN	LOAD	SAVE
DELETE	NEW	
EDIT	RENUMBER	

OPTION BREAK

OPTION BREAK nn

Set the value of the break key to 'nn'. This key is used to interrupt a running program.

The value of the break key is set to Ctrl-C key at start up but it can be set to any keyboard key using this command (for example, OPTION BREAK 156 will set the break key to the F12 key). Setting this option to an invalid key value (for example, 255) will disable the break function entirely.

See the list of Special Keyboard Keys.

See also:

CONTINUE

RUN

RENUMBER

RENUMBER

RENUMBER first

RENUMBER first, incr

RENUMBER first, incr, start

Renumber the program currently held in memory including all references to line numbers in commands such as GOTO, GOSUB, ON, etc.

'first' is the first number to be used in the new sequence. Default is 10.

'incr' is the increment for each line. Default is 10.

'start' is the line number in the old program where renumbering should commence from. The default is the first line of the program.

This command will first check for errors that may disrupt the renumbering process and it will only change the program in memory if no errors are found. However, it is prudent to save the program before running this command in case there are some errors that are not caught.

Note: RENumber is only in 4.3 and earlier. RENumber was removed in 4.3A.

See also:

AUTO	LIST
DELETE	NEW

RUN

RUN [line] [file\$]

Executes the program in memory. If a line number is supplied then execution will begin at that line, otherwise it will start at the beginning of the program. Or, if a file name (file\$) is supplied, the current program will be erased and that program will be loaded from the current drive and executed. This enables one program to load and run another.

```
RUN "TEST.BAS"

RUN "DAVE\TEST.BAS"

' Run starting from root dir:
RUN "\DAVE\TEST.BAS"

' Up a directory from where you are:
RUN "..\DAVE\TEST.BAS"
```

If an extension is not specified ".BAS" will be added to the file name.

The length of the path plus the file name (including punctuation) must be less than 127 characters.

Implied RUN command

At the command prompt it is possible to omit the RUN keyword and MMBasic will search the default drive and directory for a matching program and if found, it will be run.

'program' is the program name. If there is no extension the extension .BAS will be automatically appended.

'command-line' is an arbitrary string of characters which can be retrieved by the new program from the read only variable MM.CMDLINE\$.

The implied RUN command is valid only at the command prompt (not within a running program).

If 'program' is the same as an internal MMBasic command the internal command will be run instead. To avoid this 'program' can be surrounded by quote marks:

```
"PRINT" command line
```

An error will be generated if the program currently in memory has been edited and not saved. If this happens (and you do not want to save the program) use the NEW command to clear the memory.

Example:

```
SORT INFILE.TXT OUTFILE.TXT
```

Will run the program SORT.BAS and the variable MM.CMDLINE\$ will hold the string: "INFILE.TXT OUTFILE.TXT"

Create an MMBasic file named CMDLINE.BAS with this code:

```
PRINT MM.CMDLINE$
```

Place it on the B: drive (SD card). With B: as the current drive, the following is a test of the command line:

```
> cmdline 1 2 3 4  
1 2 3 4  
>
```

See also:

CHAIN	MM.CMDLINE\$	OPTION BREAK
EDIT	MM.FNAME\$	SAVE
LOAD	NEW	

SAVE

SAVE [file\$]

Saves the program in the current working directory on the SD card as 'file\$'. The file name is optional and if omitted the last file name used in SAVE, LOAD or RUN will be automatically used. Example: SAVE "TEST.BAS" If an extension is not specified ".BAS" will be added to the file name.

Example:

```
SAVE "TEST.BAS"
```

If an extension is not specified ".BAS" will be added to the file name.

See also:

CHAIN	MERGE	RUN
EDIT	MM.FNAME\$	
LOAD	NEW	

TIME\$ (command/function)

TIME\$ = "HH:MM:SS"

TIME\$ = "HH:MM"

TIME\$ = "HH"

Sets the time of the internal clock. MM and SS are optional and will default to zero if not specified.

For example TIME\$ = "14:30" will set the clock to 14:30 with zero seconds. The time is set to "0:0:0" (midnight) on power up.

Normally the time is set to "00:00:00" on power up. If the real time clock option is fitted to the Color Maximite the current time will be automatically set using that facility.

TIME\$

TIME\$ returns the current time based on MMBasic 's internal clock as a string in the form "HH:MM:SS" in 24 hour notation. For example, "14:30:00". The internal clock/calendar will keep track of the time and date including leap years.

See also:

DATE\$

TIME\$

TROFF

TROFF

Turns the trace facility off; see TRON.

TRON

TRON

Turns on the trace facility. This facility will print each line number in square brackets as the program is executed. This is useful in debugging programs.

MMBasic Screen, Graphics, and Keyboard

Screen and graphics commands affect both the output on the VGA connector (or composite output) and the output to the serial console.

The keyboard input can come from a physically attached keyboard, or through the serial console.

The PC/USB interface is most frequently the serial console, but the console can be redirected to a hardware serial port.

Graphics

Graphics commands operate on the video output only (not USB). Coordinates are measured in pixels with x being the horizontal coordinate and y the vertical coordinate. The top left of the screen is at location x = 0 and y = 0, and the bottom right of the screen defined by the read-only variables x = MM.HRES and y = MM.VRES which change depending on the video mode selected (VGA or composite). Increasing positive numbers represent movement down the screen and to the right.

You can clear the screen with CLS and an individual pixel can be turned on or off and its color set with PIXEL(x,y) = . You can draw lines and boxes with LINE, and circles using CIRCLE. You can also set the screen location (in pixels) of the PRINT output using @(x,y) and the SAVEBMP command will save the current screen as a BMP file. LOADBMP will load and display a bitmap image stored on the SD card.

The origin of graphics commands is in the upper left corner of the screen.
Resolution for mode 1, 2, and 3 is 480 horizontal pixels and 432 vertical pixels.

Resolution for mode 4 is 240 horizontal pixels and 216 vertical pixels.

Resolution for composite PAL is 304 horizontal pixels and 216 vertical pixels.

Resolution for composite NTSC is 304 horizontal pixels and 180 vertical pixels.

In mode 1, 2, and 3 the display is 80 characters wide by 36 lines using the internal default font.

In mode 4 the display is 40 characters wide by 18 lines.

In composite PAL the display is 50 characters wide by 18 lines.

In composite NTSC the display is 50 characters wide by 15 lines.

VGA mode 1, 2, or 3:

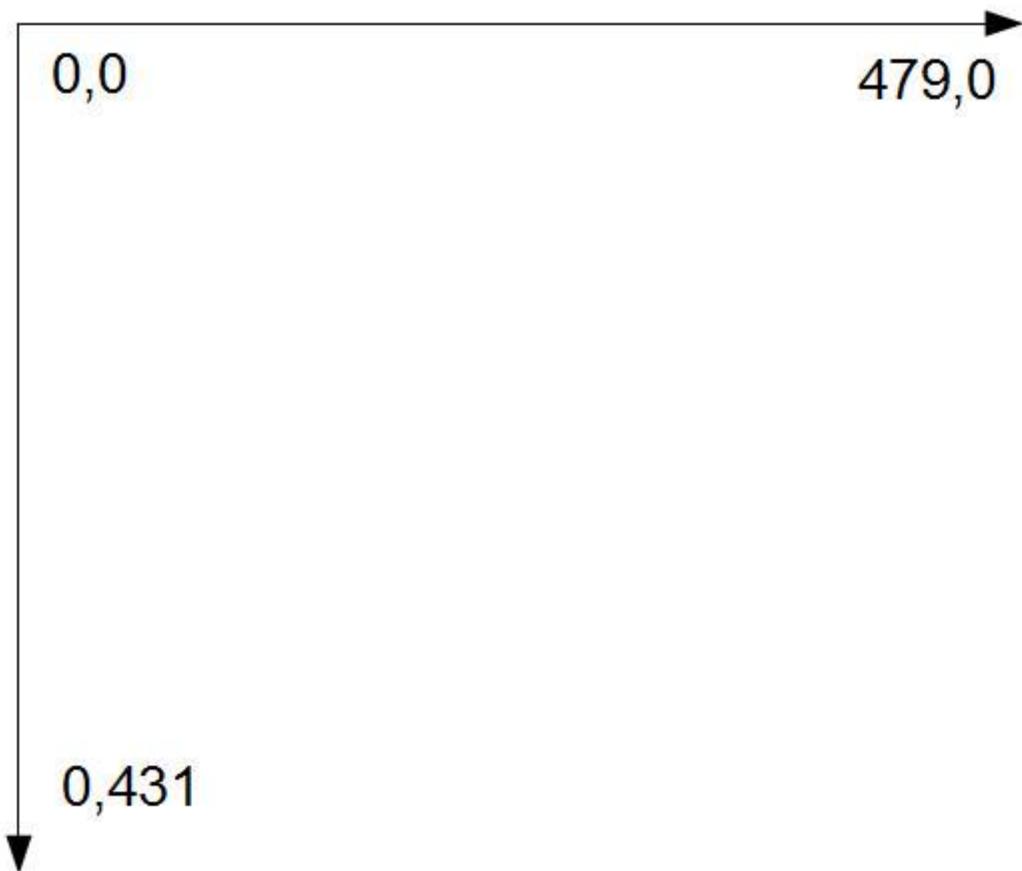


Illustration 141: VGA screen resolution for modes 1, 2, and 3.

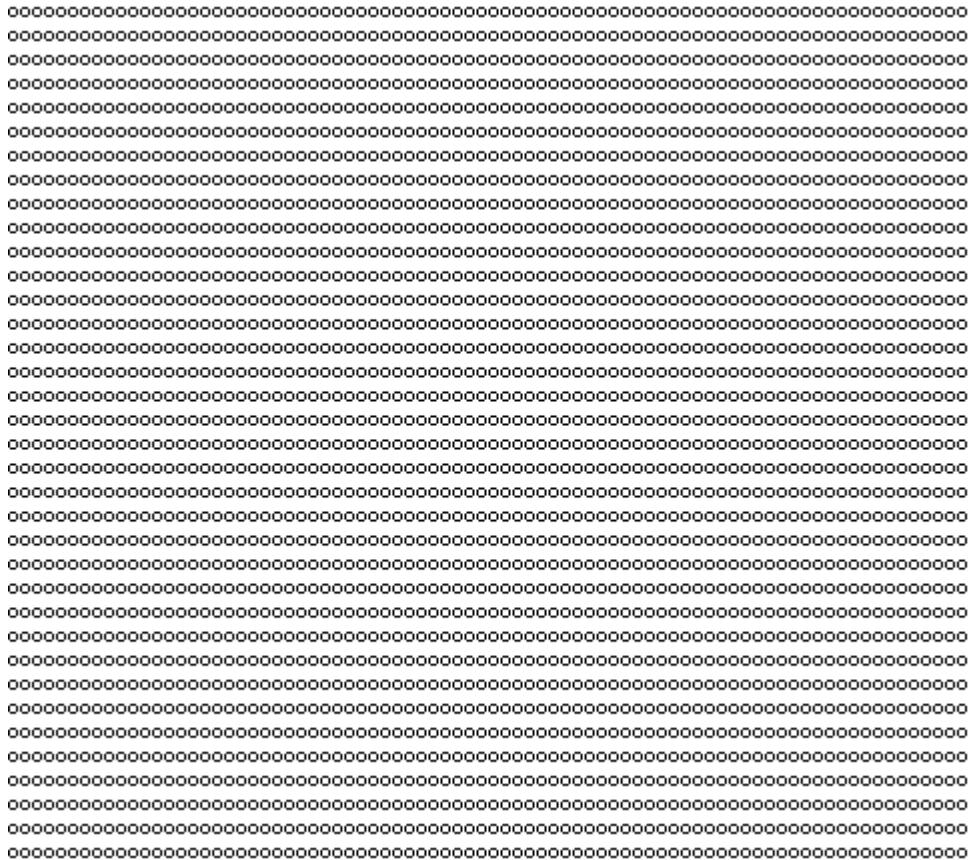


Illustration 142: In mode 1, 2, and 3 the display is 80 characters wide by 36 lines using the internal default font.

VGA mode 4:

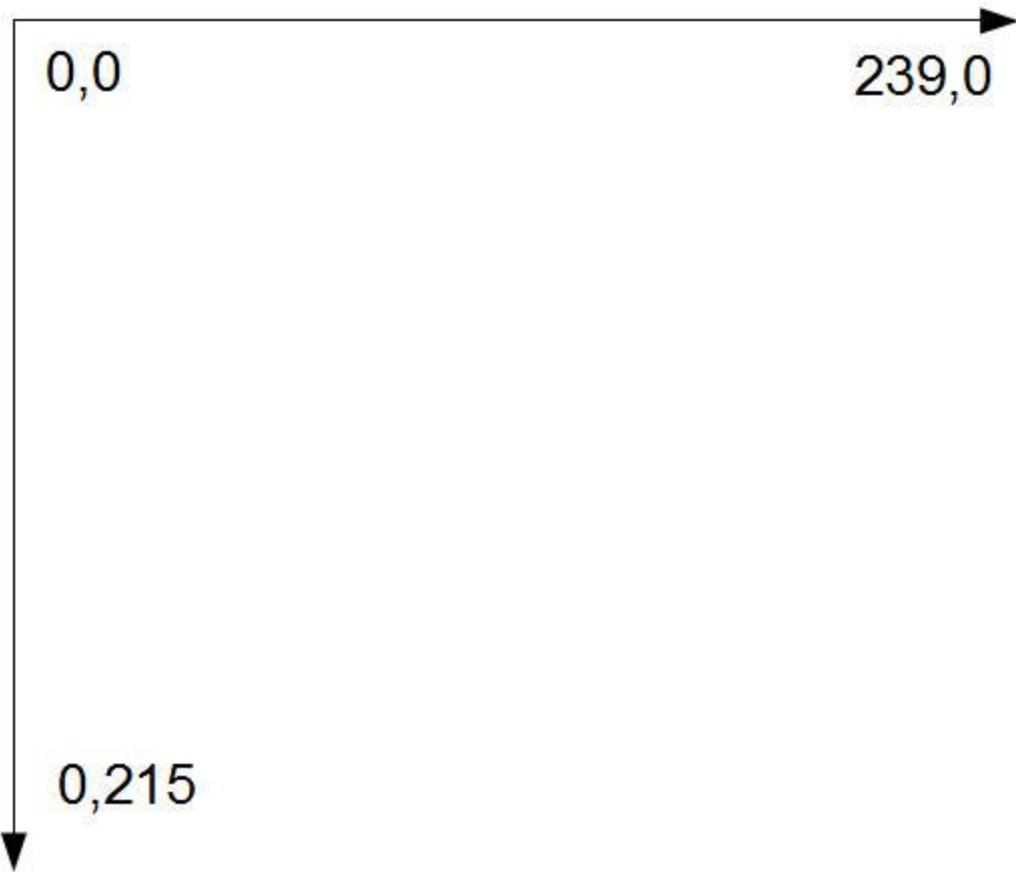


Illustration 143: VGA screen resolution for mode 4.

```
oooooooooooooooooooooooooooooo  
oooooooooooooooooooooooooooooo
```

Illustration 144: In mode 4 the display is 40 characters wide by 18 lines – default font.

Composite PAL:

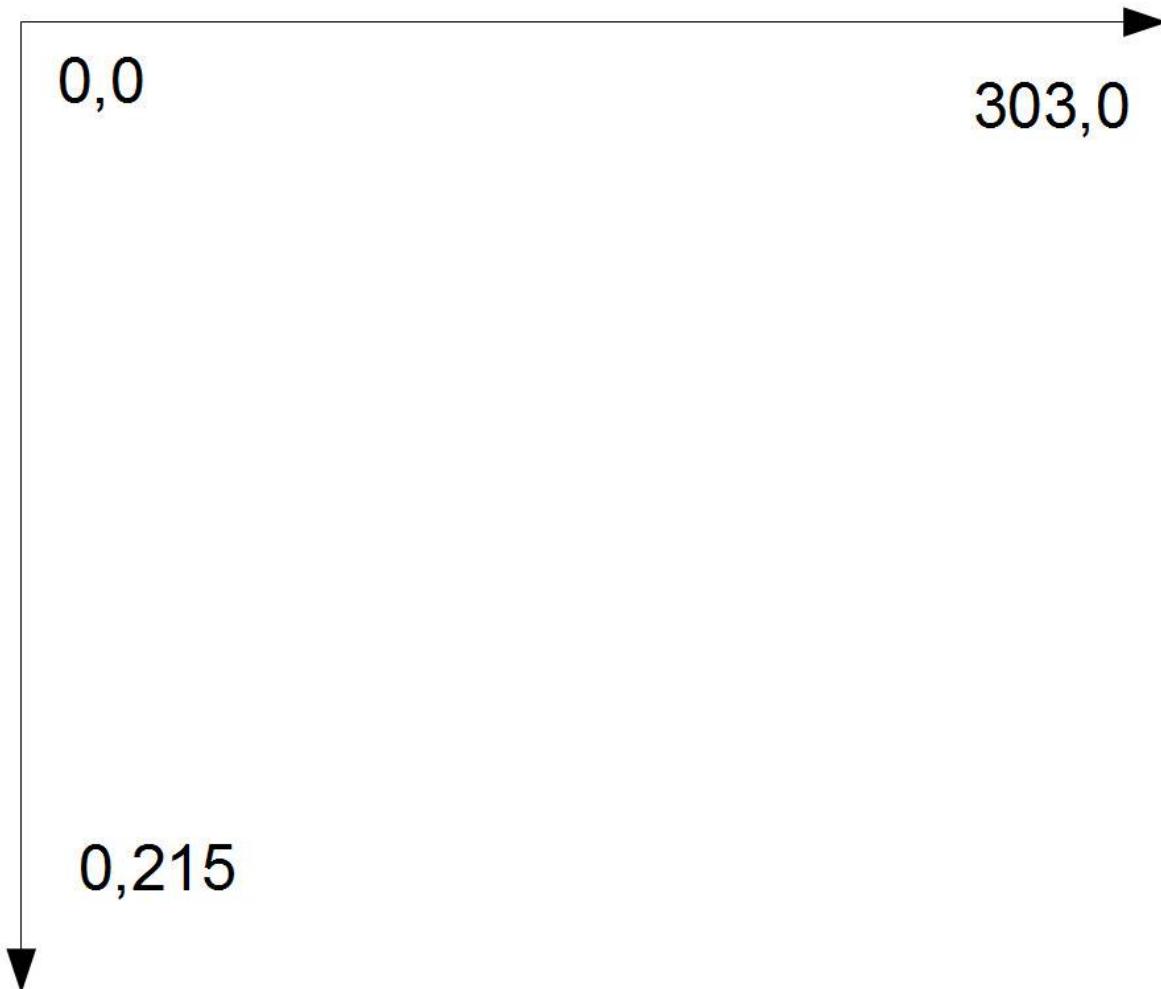


Illustration 145: Composite screen resolution for PAL.

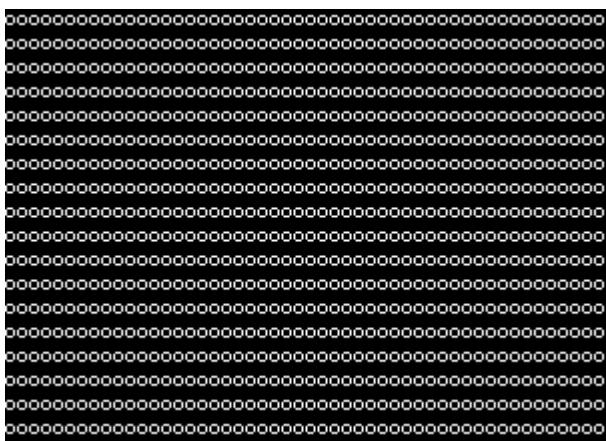


Illustration 146: In composite PAL the display is 50 characters wide by 18 lines – default font.

Composite NTSC:

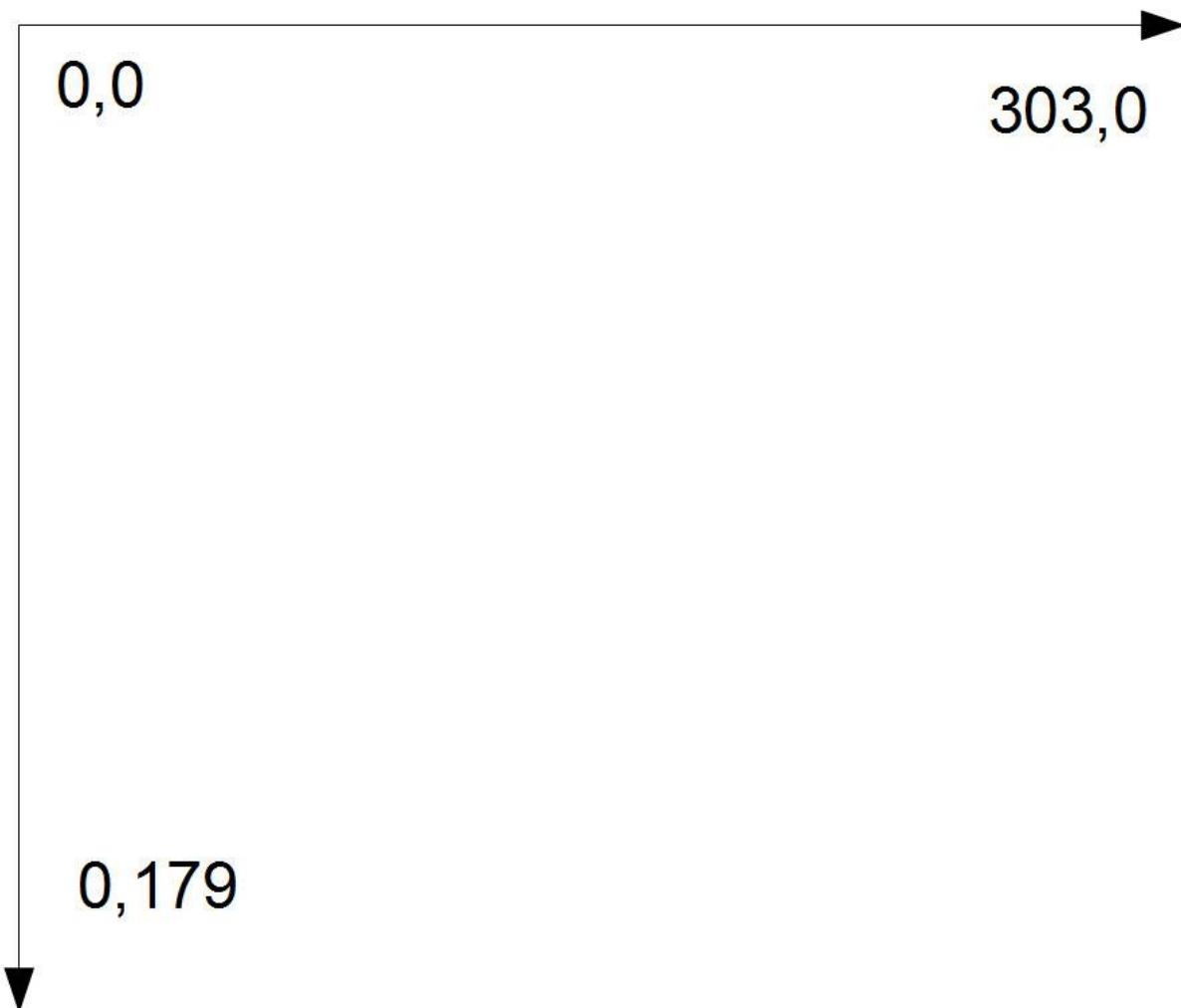


Illustration 147: Composite screen resolution for NTSC.

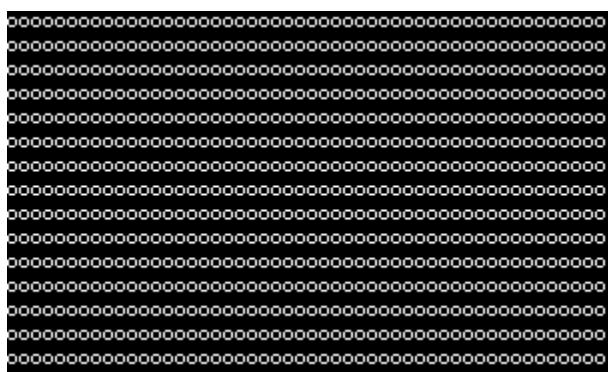


Illustration 148: In composite NTSC the display is 50 characters wide by 15 lines – default font.

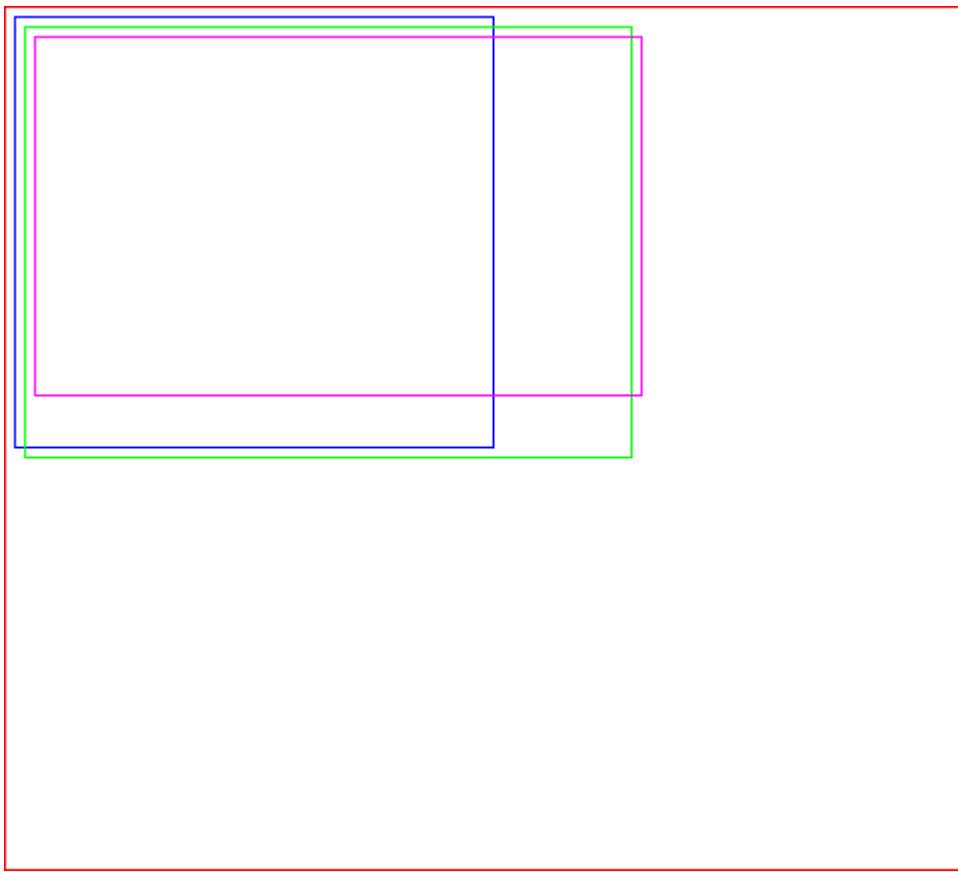


Illustration 149: Comparisons of the screen resolutions.

The image above compares the screen resolutions. The red box is VGA mode 1, 2, or 3. The blue box is VGA mode 4. Composite PAL is green and NTSC is purple. The boxes are offset from each other by 5 pixels to make it easier to see lines that would otherwise overlap. Here is the code:

```
' Square for VGA mode 1, 2, 3
LINE (0,0) - (479,431), RED, B

' Square for VGA mode 4
LINE (5,5) - (239+5,215+5), BLUE, B

' Square for composite PAL
LINE (10,10) - (303+10,215+10), GREEN, B

' Square for composite NTSC
LINE (15,15) - (303+15,179+15), PURPLE, B
```


Working with Color

The CGCOLORMAX supports eight colors (black, blue, green, cyan, red, purple, yellow and white). The monochrome CGMMSTICK Maximite support just two (black and white). In most places you can also specify the color as -1 (negative 1) to invert a pixel (this is useful in animation).

Throughout MMBasic you can refer to the colors by their name or their corresponding numbers where black = 0, blue = 1, green = 2, etc through to white = 7. Commands such as LINE and CIRCLE use this color or number to specify the color to draw. For example:

```
CIRCLE (100, 100), 50, CYAN      ' will draw a circle in cyan.  
CIRCLE (100, 100), 50, 3        ' same
```

You can also specify a default color that will be used for all screen output with the COLOR command. For example: COLOR PURPLE will set the color of text to purple (and any other output where the color is not specified). The COLOR command also takes a second parameter for the background color. So, COLOR YELLOW, BLUE specifies that text will be displayed in yellow on a blue background.

In addition to the COLOR command you can change the color of text by embedding color codes into strings using the CLR\$() function. For example, the following will display each word in a different color:

```
Txt$ = "This is " + CLR$(RED) "red " + CLR$(YELLOW) + "yellow"  
PRINT Txt$
```

You can also use this function to set the background color by supplying a second parameter. For example:

```
PRINT CLR$(YELLOW,RED) " ALARM "
```

If the function is used without any parameters (eg, CLR\$()) it will reset the colors to the defaults set by the last COLOR command. The colors are also reset when the print command terminates.

This function simply generates a two character string where the first character is the number 128 plus the foreground color number and the second character is the number 192 plus the background color number.

You can use this trick to embed color commands in any text, even text read from a text file on the SD card.

COLORS

Colors can be selected by number, or by a reserved keyword:

0	BLACK
1	BLUE
2	GREEN
3	CYAN
4	RED
5	PURPLE
6	YELLOW
7	WHITE

Color Modes

The video system can be configured into one of four modes using the MODE command. This enables the programmer to trade off the number of colors used on the screen and the graphic resolution against the amount of memory required by the video driver. Modes 1 and 4 use the least amount of memory while mode 3 uses the most. The syntax of the MODE command is:

```
MODE color-mode, palette
```

The 'color-mode' can be one of four numbers:

1 Monochrome mode. In this mode the Color MMBasic operates the same as the monochrome MMBasic and has the maximum amount of free memory available for programs and data. The second argument of the MODE command ('palette') selects the color to be used for all output. It can be any color number from black to white.

2 Four color mode. In this mode four colors (including black) are available. The actual colors are selected by a number (1 to 6) used in the second argument of the MODE command ('palette'). See the following image or the MODE command for a listing of the actual colors available.

3 Eight color mode. In this mode all eight colors are available and can be used simultaneously anywhere on the screen. The 'palette' argument is not required and will be ignored if specified. MODE 3 uses the most memory but there still is plenty left for programs and data. This is the default when the CGCOLORMAX is first powered up.

4 240x216 pixel mode. In this mode all eight colors are available and the video resolution is halved (meaning that characters and graphics are doubled in size). This mode is most suitable for games as all colors are available, it has the maximum amount of free memory and drawing of graphics is very fast. The 'palette' argument is not required and will be ignored if specified.

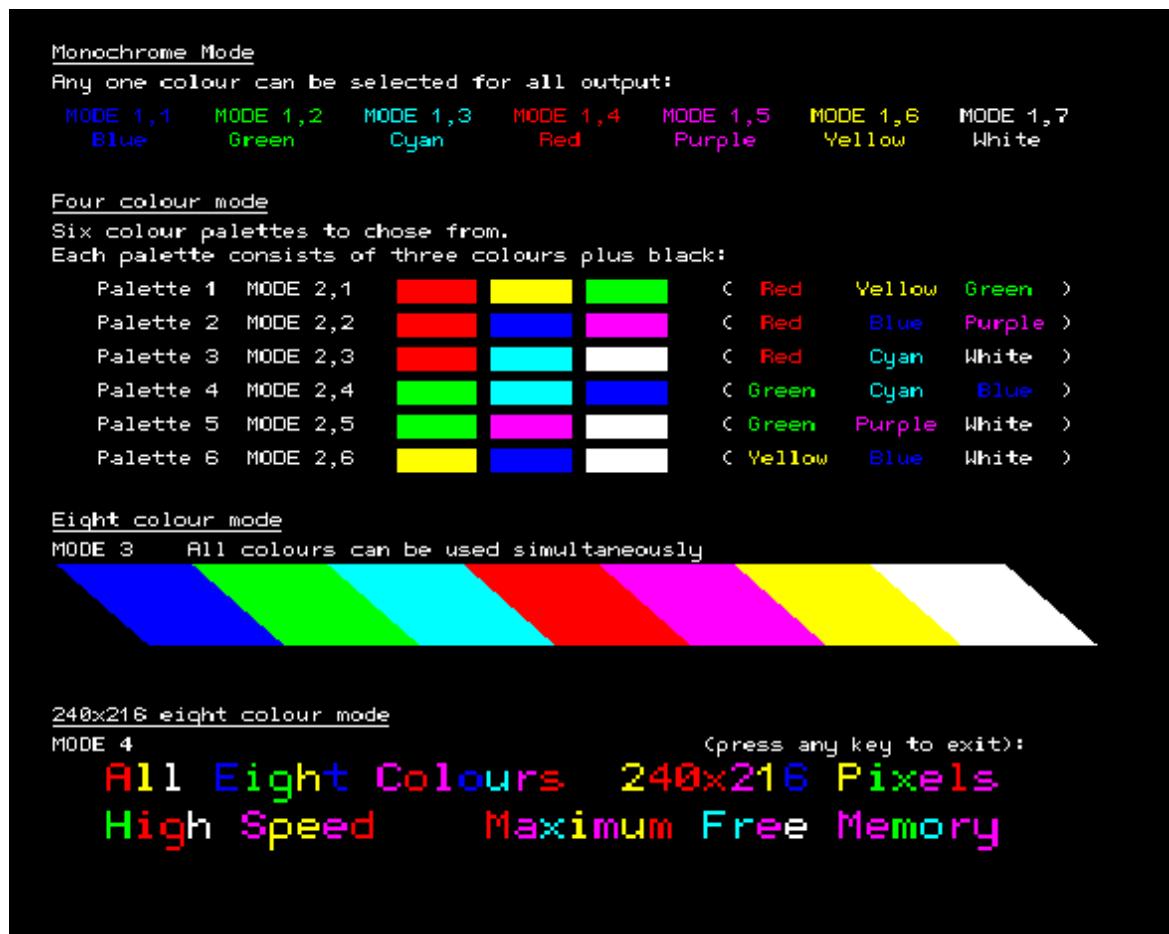


Illustration 150: Color graphics. Examples of the MODE command. (Image actually generated completely in mode 3.)

You can change the mode and the palette at any time and as often as you need, even within a running program.

Scan Line Color Override

In mode 1 (monochrome) there is an additional facility to change the color of each horizontal line of pixels on the screen using the SCANLINE command. This is intended mostly for programmers writing games and provides limited control over color while still providing the maximum amount of free memory. The syntax is:

```
SCANLINE color, startline, endline
```

This command can only be used in MODE 1, 7 (monochrome with the color set to white) and is used to set the color for each horizontal scan line of pixels on the screen.

'color' is the color to be used and can be any one of the eight colors, 'startline' is the starting scan line to be set to that color and 'endline' is the ending line. The scan lines are numbered from 0 at the top of the screen to 431 at the bottom of the screen. The numbering is the same as that used when specifying the vertical coordinates of a pixel.

You can use multiple SCANLINE commands to set multiple scan lines to different colors. For example:

```
SCANLINE RED, 0, 9      ' set the top 10 lines to red
SCANLINE YELLOW, 120    ' and set only line 120 to yellow
SCANLINE BLUE, 200, 219  ' and set a band of 20 to blue
```

To turn off the override imposed by the use of SCANLINE commands you can use the MODE command to reselect mode 1 or a change to a different mode. It is also automatically turned off when control is returned to the command prompt.

Game Playing Features

MMBasic 4.x introduces a number of features that are intended to make it easier to write games on the Maximite.

MODE 4

The color MODE 4 described in the previous section is mostly intended for games. It provides eight colors and leaves plenty of free memory for the other aspects of an animated game (the program, sprites, arrays, and so on).

Because this color mode has only one quarter of the pixels the graphics operations are much faster due to the fact that there are fewer pixels that need to be manipulated by MMBasic when drawing on the screen.

BLIT

This command will move an area of the video screen from one location to another. The destination can overlap the source area and the BLIT command will copy the video data correctly to avoid corruption. On the CGCOLORMAX1 you can also independently specify what color planes to copy.

This method of moving video data is much faster than copying pixels one by one and allows for rapid animation on the screen. It can also be used to replicate a pattern like a border or a brick wall to build a complete image.

SPRITE

A sprite is a 16x16 bit graphic image that can be moved about on the screen independently of the background. When the sprite is displayed MMBasic will automatically save the background text and graphics under the sprite and when the sprite is turned off or moved MMBasic will restore the background.

The sprites are defined in a file which is loaded into memory using the SPRITE LOAD command, the number of sprites contained in the file is only limited by the amount of available memory. Each sprite in the file can contain pixels of any color (on the CGCOLORMAX1 Maximite) and can also have transparent pixels which allow the background to show through.

To manipulate the sprites you can use the command SPRITE ON which will display a specific sprite at a specified location on the screen. SPRITE MOVE will move a sprite to a new location and restore the background. SPRITE OFF will remove a sprite from the screen and restore the background.

Sprites should not overlap but if they do you should turn them off in the reverse sequence that you turned them on before you turn them on again at their new location. This will enable the background image to be correctly maintained.

For example, the following two sprites overlap:

```
SPRITE ON 1, 100, 150  ' sprite 1 is drawn at x = 100, y = 150
SPRITE ON 2, 110, 160  ' sprite 2 overlaps
```

To move the sprites they need to be turned off in the reverse sequence:

```
SPRITE OFF 2
SPRITE OFF 1
```

Then they can be redrawn at their new location:

```
SPRITE ON 1, 104, 154  ' sprite 1 is drawn at x = 104, y = 154
SPRITE ON 2, 116, 166  ' sprite 2 still overlaps
```

Because sprites are drawn so fast the user is unaware that the sprite has been turned off then redrawn.

Specifying the Background Color

An alternative to turning sprites off in sequence is to specify the background color when using the SPRITE ON or SPRITE MOVE commands. The background color is optional and is specified at the end of the command.

For example:

```
SPRITE ON 1, 100, 100, BLUE
```

This results in a much faster operation when using a solid background color because MMBasic does not have to copy the background to a buffer. It also means the MMBasic will always restore the correct color, even if sprites overlap.

Collision Detection

You can use the COLLISION() function to detect if a sprite has collided with another sprite or the edges of the screen. A collision is reported if the non transparent portion of the sprite is just touching (ie, the non transparent pixels are adjacent) or overlapping the non transparent portion of another sprite or the edge of the screen.

To detect if a sprite has collided with another sprite you use:

```
R = COLLISION (n, SPRITE)
```

And to detect if it has collided with the screen edge you use:

```
R = COLLISION (n, EDGE)
```

Where 'n' is the number of the sprite to test.

In both cases the value returned by the function indicates if the collision was on the left of the sprite, the right, the top, etc.

Following a collision COLLISION () will return:

&B0001 Indicating a collision with something on the left of the sprite

&B0010 Collision on the right

&B0100 Collision on the top

&B1000 Collision on the bottom

Note that it is possible for these results to be combined. For example; a result of &B0101 indicates that the sprite has collided with something both at the top and left of the sprite (for example the top left corner of the screen).

When testing for collisions with other sprites it is possible for the function to return &B1111 indicating that there are collisions on all sides. This can happen if the sprite is surrounded on all sides by other sprites.

If the sprite is overlapping another (ie, one or more non transparent pixels are on top of another sprite's non transparent pixels) bit &B10000 will be set in the value returned by COLLISION () in addition to the bits for left, right, etc as described above.

LOADBMP and FONTS

The LOADBMP command will load a color or monochrome bitmap image and display it at a specified location on the screen. This is handy for loading background images for games.

The FONT command can also be used to load custom designed graphic images and display them on the screen.

Loadable Fonts

This section describes the format of a font file that can be loaded using the FONT LOAD command.

A font file is just a text file containing ordinary characters which are loaded line by line to build the bitmap of each character in the font. Each character can be up to 64 pixels high and 255 pixels wide. Up to 255 characters can be defined.

The first non-comment line in the file must be the specifications for the font as follows:

height, width, start, end

Where 'height' and 'width' are the size of each character in pixels, 'start' is the number in the ASCII chart where the first character sits and 'end' is the last character. Each number is separated by a comma. So, for example, 16, 11, 48, 57 means that the font is 16 pixels high and 11 wide. The first character is decimal 48 (the zero character) and the last is 57 (number nine character).

The remainder of the lines specify the bitmap for each character.

Each line represents a horizontal row of pixels. A space means the pixel is not illuminated and any other character will turn the pixel on. If the font is 11 pixels wide there must be 11 characters in the line although trailing spaces can be omitted. The first line is the top row of pixels in the character, the next is the second and so on. If the character is 16 pixels high there must be 16 lines to define the character. This repeats until each character is drawn. Using the above example of a font 16x11 with 10 characters there must be a total of 160 lines with each line 11 characters wide. This is in addition to the specification line at the top.

A comment line has an apostrophe ('') as the first character and can occur anywhere. A comment line is completely ignored; all other lines are significant.

The following example creates two small icons; a smiley face and a frowning face. Each is 11x11 pixels with the first (the smiley face) in the position of the zero character (0) and the frowning face in the position of number one (1). To display a smiley face your program would contain this:

```
40 FONT LOAD "FACES.FNT" AS #6      ' load the font
50 FONT #6                           ' select the font
60 PRINT "0"                         ' print a smiley face
```

```
' example
' FACES.FNT
11,11,48,49
```

```
    XXX
    XX    XX
    XX      XX
XX  X  X  XX
X          X
XX X    X XX
X  XXX   X
XX    XX
    XXX
```

```
    XXX
    XX    XX
    XX      XX
XX  X  X  XX
X          X
XX  XXX   XX
X  X    X  X
XX    XX
    XXX
```

Sprite Definition Files

This section describes the format of a sprite file that can be loaded using the SPRITE LOAD command.

A sprite file is similar to a font file except that it contains the definition of sprites which are 16x16 bit graphical objects that can be moved about on the video screen without disturbing background text or graphics. The sprite file is just a text file containing ordinary characters which are loaded line by line to build the bitmap of each sprite. Currently the dimensions of each sprite are fixed at 16x16 bits although alternative sizes may be allowed in the future.

The first non-comment line in the file must be the specifications for the sprite file as follows:

dimension, number

Where 'dimension' is the height and width of the sprites in pixels. At this time it must be the number 16. 'number' is the number of sprites in the file and is limited only by the amount of free memory available. The remainder of the lines specify the bitmap for each sprite.

Each line represents a horizontal row of pixels with each character in the line defining the color of the pixel. The character can be a single numeric digit in the range of 0 to 7 representing the colors black to white or it can be a space which means that that particular pixel will be transparent (ie, the background will show through).

Each sprite must immediately follow the preceding sprite in the file and be defined by 16 lines each of 16 characters wide (although trailing spaces can be omitted and will be assumed to be transparent pixels).

A comment line has an apostrophe ('') as the first character and can occur anywhere. A comment line is completely ignored; all other lines are significant.

The following example is of a file that contains a single sprite consisting of a red ball with a white border and a blue center dot:

Special Keyboard Keys

MMBasic generates a single unique character for the function keys and other special keys on the keyboard.

These are shown in the table as hexadecimal and decimal numbers:

Keyboard Key	Key Code (Hex)	Key Code (Decimal)
Up Arrow	80	128
Down Arrow	81	129
Left Arrow	82	130
Right Arrow	83	131
Insert	84	132
Home	86	134
End	87	135
Page Up	88	136
Page Down	89	137
Alt	8B	139
Num Lock	8C	140
F1	91	145
F2	92	146
F3	93	147
F4	94	148
F5	95	149
F6	96	150
F7	97	151
F8	98	152
F9	99	153
F10	9A	154
F11	9B	155
F12	9C	156

If the control key is simultaneously pressed then 20 (hex) is added to the code (this is the equivalent of setting bit 5). If the shift key is simultaneously pressed then 40 (hex) is added to the code (this is the equivalent of setting bit 6). If both are pressed 60 (hex) is added. For example Control-PageDown will generate A9 (hex).

The shift modifier only works with the function keys F1 to F12; it is ignored for the other keys.

MMBasic will translate most VT100 escape codes generated by terminal emulators such as Tera Term and Putty to these codes (excluding the shift and control modifiers). This means that a terminal emulator operating over a USB or a serial port opened as console will generate the same key codes as a directly attached keyboard. This is particularly useful when using the EDIT command.

KEYDOWN FUNCTION

The KEYDOWN function makes it easy to tell if the user is holding down a key like an arrow key.

BLIT

BLIT x1, y1, x2, y2, w, h

Copy one rectangular section of the video screen to another. The source coordinate is 'x1' and 'y1'. The destination coordinate is 'x2' and 'y2'. The width of the screen area to copy is 'w' and the height is 'h'.

The rectangle is defined with the upper left pixel as the origin for the rectangle.

All arguments are in pixels. The source and destination can overlap.

BLIT x1, y1, x2, y2, w, h, RGB

CGCOLORMAX only: If the optional argument 'RGB' is specified then only the specified color planes will be copied. For example, 'GB' will copy only the green and blue color planes.



Illustration 151: BLIT copy and paste examples.

Code for BLIT example:

```
' Move to the B drive and load in a BMP file
DRIVE "B:"
LOADBMP "SCREEN.BMP"

' Copy logo once
BLIT 0, 0, 0, 100, 479, 100

' Copy logo again, without green
BLIT 0, 0, 0, 200, 479, 100, RB

' Copy logo again, green only
BLIT 0, 0, 0, 300, 479, 100, G
```

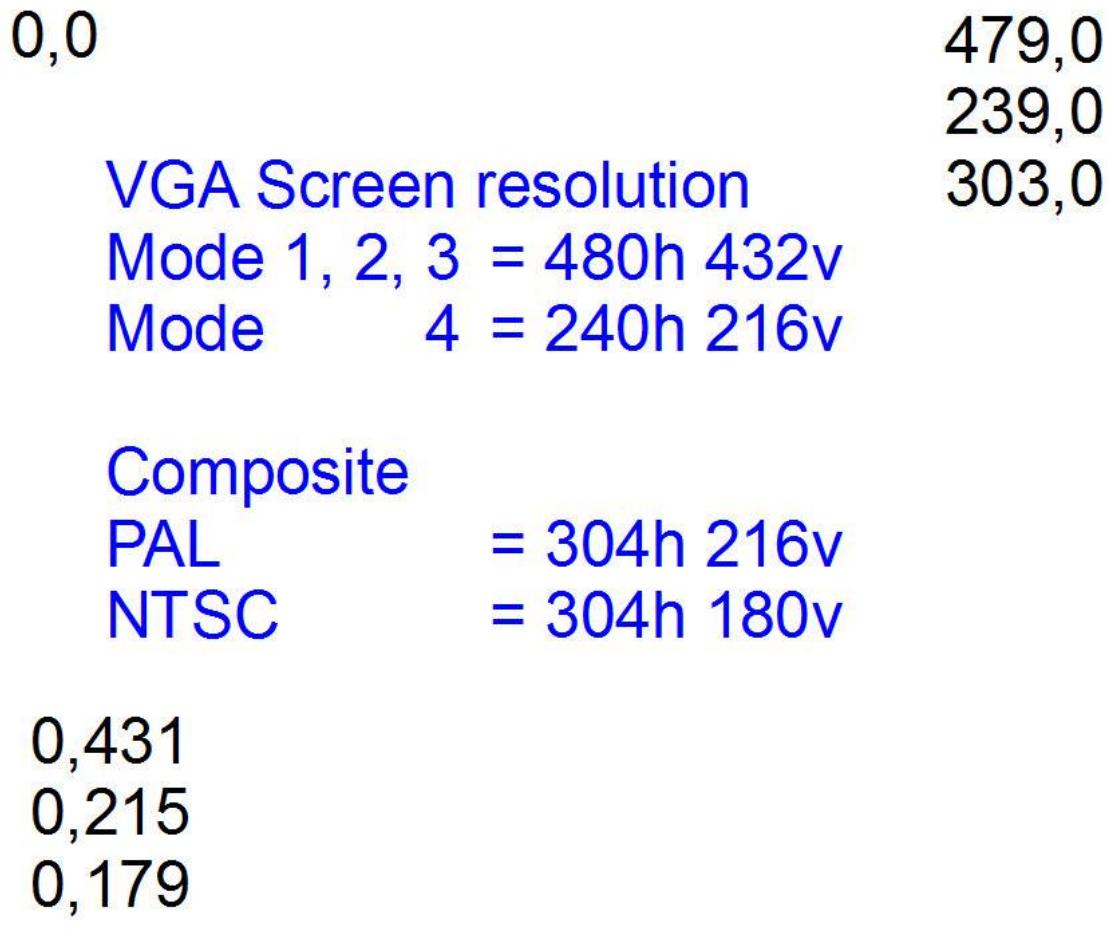


Illustration 152: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

CIRCLE	MM.HRES	PRINT
CLR\$	MM.VPOS	PRINT @
CLS	MM.VRES	PSET
COLOR	MODE	SCANLINE
LINE	PIXEL	SPRITE
LOCATE	POS	
MM.HPOS	PRESET	

CIRCLE

CIRCLE (x, y),r [,c [,aspect [,F]]]

Draws a circle on the video output centered at 'x' and 'y' with a radius of 'r'.

'c' is the color and defaults to the current foreground color if not specified. 'c' can also be -1 which will invert the pixels.

The optional 'aspect' will define the aspect ratio. Because the pixels displayed on the VGA output are rectangular an aspect ratio of 0.83 will result in a near perfect circle on most monitors. The default if 'aspect' is not specified is 1.0 which is backwards compatible with early versions of MMBasic and will result in a slightly oval circle.

The F option can be appended to the end of the argument list and will cause the circle to be filled according to the 'c' parameter.

Note that without using 'aspect' to compensate the pixels on a monitor are not exactly square and the displayed circle will be oval to some degree. When the screen is saved with SAVEBMP, the shape will be a circle.

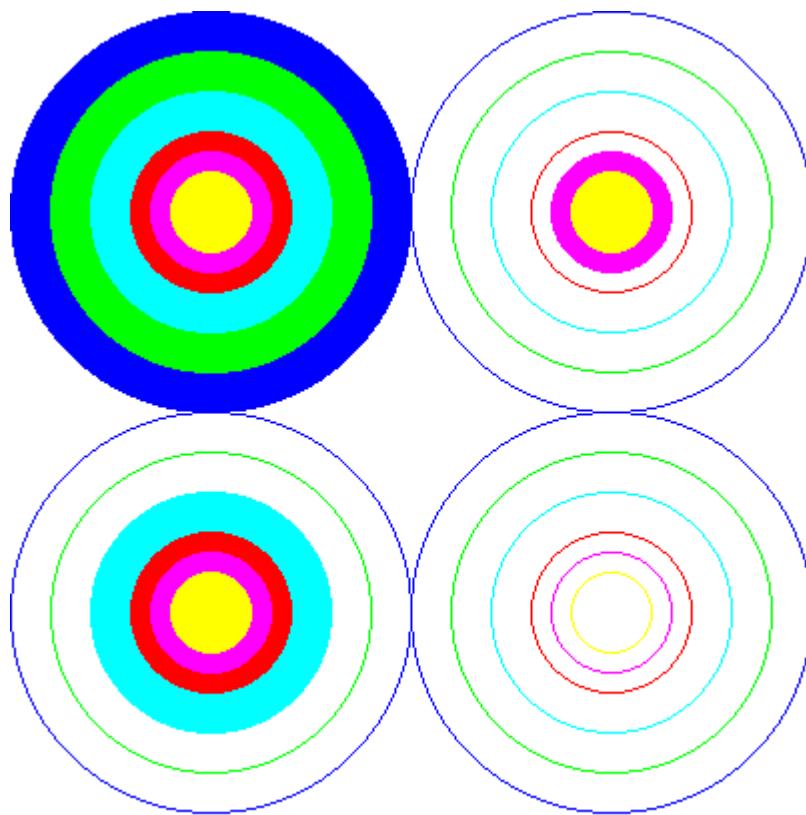


Illustration 153: Example circles.

Example circles code:

```
COLOR 0, 7

CLS
CIRCLE (100,100), 100, 1, F
CIRCLE (100,100), 80, 2, F
CIRCLE (100,100), 60, 3, F
CIRCLE (100,100), 40, 4, F
CIRCLE (100,100), 30, 5, F
CIRCLE (100,100), 20, 6, F

CIRCLE (100,300), 100, 1
CIRCLE (100,300), 80, 2
CIRCLE (100,300), 60, 3, F
CIRCLE (100,300), 40, 4, F
CIRCLE (100,300), 30, 5, F
CIRCLE (100,300), 20, 6, F

CIRCLE (300,100), 100, 1
CIRCLE (300,100), 80, 2
CIRCLE (300,100), 60, 3
CIRCLE (300,100), 40, 4
CIRCLE (300,100), 30, 5, F
CIRCLE (300,100), 20, 6, F

CIRCLE (300,300), 100, 1
CIRCLE (300,300), 80, 2
CIRCLE (300,300), 60, 3
CIRCLE (300,300), 40, 4
CIRCLE (300,300), 30, 5
CIRCLE (300,300), 20, 6
```

0,0		479,0
		239,0
VGA Screen resolution		303,0
Mode 1, 2, 3	= 480h 432v	
Mode 4	= 240h 216v	
Composite		
PAL	= 304h 216v	
NTSC	= 304h 180v	
0,431		
0,215		
0,179		

Illustration 154: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HRES	PRINT
CLR\$	MM.VPOS	PRINT @
CLS	MM.VRES	PSET
COLOR	MODE	SCANLINE
LINE	PIXEL	SPRITE
LOCATE	POS	
MM.HPOS	PRESET	

CLR\$

CLR\$()

CLR\$(fg)

CLR\$(fg, bg)

Returns a string containing embedded codes to select colors in a string.

'fg' is the foreground color and 'bg' is the background color. If no parameters are specified both the foreground and background colors will be reset to the defaults set by the last COLOR command.

Example, this will display yellow letters on a red background:

```
COLOR 0, 7  
CLS  
PRINT CLR$(YELLOW,RED) " ALARM "
```

This function simply generates a two character string where the first character is the number 128 plus the foreground color number and the second character is the number 192 plus the background color number.



ALARM

Illustration 155: Example of CLR\$.

See also:

BLIT	MM.HRES	PRESET
CLS	MM.VPOS	PRINT
COLOR	MM.VRES	PRINT @
LINE	MODE	PSET
LOCATE	PIXEL	SCANLINE
MM.HPOS	POS	SPRITE

CLS

CLS [color]

Clears the video display screen and places the cursor in the top left corner. Optionally 'color' can be specified which will be used for the background when clearing the screen.

If 'color' is not specified, then the previously set color is used, such as that set by COLOR.

See also:

BLIT	MM.HRES	PRINT
CIRCLE	MM.VPOS	PRINT @
CLR\$	MM.VRES	PSET
COLOR	MODE	SCANLINE
LINE	PIXEL	SPRITE
LOCATE	POS	
MM.HPOS	PRESET	

COLLISION

COLLISION(n, EDGE)

COLLISION(n, SPRITE)

Tests if a collision has occurred between sprite 'n' and the edge of the screen or another sprite depending on the form used. Returns:

&B0001 Indicating a collision to the left of the sprite

&B0010 Collision on the right

&B0100 Collision on the top

&B1000 Collision on the bottom

Note that it is possible for these results to be combined. For example; a result of &B0101 indicates that the sprite has collided with something both at the top and left of the sprite (for example the top left corner of the screen).

See also:

BLIT	MM.HRES	PRINT
CIRCLE	MM.VPOS	PRINT @
CLR\$	MM.VRES	PSET
COLOR	MODE	SCANLINE
LINE	PIXEL	SPRITE
LOCATE	POS	
MM.HPOS	PRESET	

COLOR

COLOR fore [, back]

Sets the default color for commands that display on the screen (PRINT, LINE, etc).

'fore' is the foreground color, 'back' is the background color. The background is optional and if not specified will default to black.

The actual color displayed will depend on the current color mode (see the MODE command).

CLS will use the most recent fore/back colors from the last run COLOR command.

```
CircuitGizmos
```

Illustration 156: COLOR command example.

Example COLOR code:

```
' Set foreground to black ( 0 )
' and background to white ( 7 )
COLOR 0, 7

' Entire screen cleared with last colors
' chosen by the COLOR command
CLS
```

```
PRINT "CircuitGizmos"

' Color names can be used instead of numbers
COLOR BLUE, WHITE
PRINT "CircuitGizmos"

COLOR GREEN, WHITE
PRINT "CircuitGizmos"

COLOR CYAN, WHITE
PRINT "CircuitGizmos"

COLOR RED, WHITE
PRINT "CircuitGizmos"

COLOR PURPLE, WHITE
PRINT "CircuitGizmos"

COLOR YELLOW, WHITE
PRINT "CircuitGizmos"

' Black background as default
COLOR WHITE
PRINT "CircuitGizmos"

COLOR BLUE
PRINT "CircuitGizmos"

COLOR GREEN
PRINT "CircuitGizmos"

COLOR CYAN
PRINT "CircuitGizmos"

COLOR RED
PRINT "CircuitGizmos"

COLOR PURPLE
PRINT "CircuitGizmos"

COLOR YELLOW
PRINT "CircuitGizmos"
```

See also:

BLIT	MM.HRES	PRESET
CIRCLE	MM.VPOS	PRINT
CLR\$	MM.VRES	PRINT @
LINE	MODE	PSET
LOCATE	PIXEL	SCANLINE
MM.HPOS	POS	SPRITE

CONFIG COMPOSITE

CONFIG COMPOSITE NTSC | PAL

The COMPOSITE setting will change the timing for the composite video output. Default is PAL. NTSC is used for composite in the United States. See the hardware for the CGMMSTICK or CGCOLORMAX for information on setting your device to output composite rather than VGA.

The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off).

The power must be cycled after changing a setting for it to take effect.

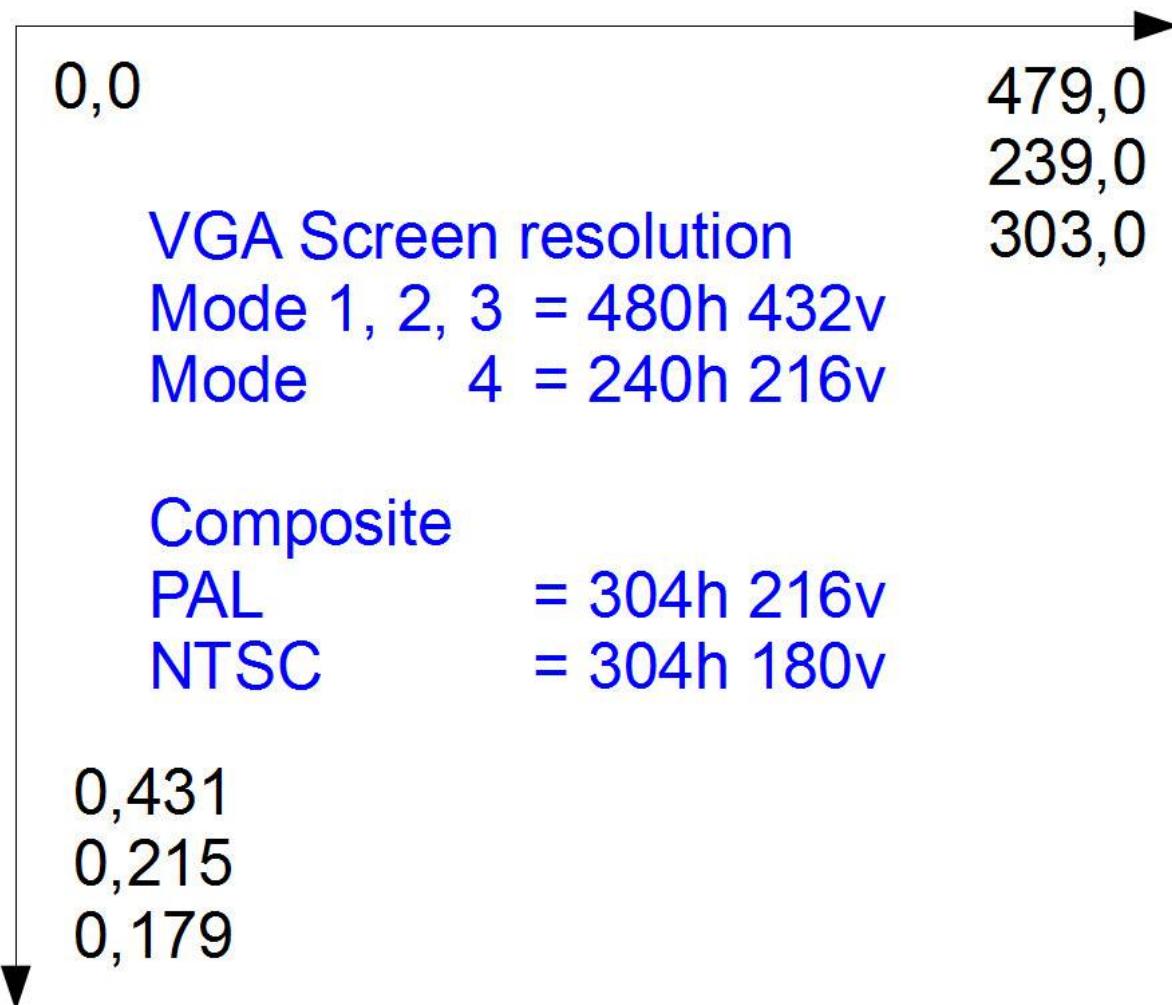


Illustration 157: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

CONFIG VIDEO

OPTION VIDEO

CONFIG VIDEO

CONFIG VIDEO OFF | ON

The VIDEO setting will switch the video output on or off. There is a slight performance improvement with the video off but the biggest benefit with VIDEO OFF is that the unused memory is returned to the general memory pool used by strings and arrays. Default is VIDEO ON.

The CONFIG command differs from other options. It permanently reconfigures MMBasic and it only needs to be run once (ie, the setting will be remembered even with the power turned off).

The power must be cycled after changing a setting for it to take effect.

See also:

CONFIG COMPOSITE OPTION VIDEO

FONT

FONT #nbr [, scale [, reverse]]

Selects a font for the video output. 'nbr' is the font number in the range of 1 to 10, the # symbol is optional.

'scale' is the multiply factor in the range of 1 to 8 (eg, a scale of 2 will double the size of a pixel in both the vertical and horizontal). Default is 1.

If 'reverse' is a number other than zero the font will be displayed in reverse video. Default is no reverse.

There are three fonts built into MMBasic:

#1 is the standard font of 10 x 5 pixels containing the full ASCII set.

#2 is a larger font of 16 x 11 pixels also with the full ASCII set.

#3 is a jumbo font of 30 x 22 pixels consisting of the numbers zero to nine and the characters plus, minus, comma and full stop.

Examples:

```
FONT #3, 2, 1  ' double scale and reverse video
```

```
FONT #2  ' just select font #2
```

Font #1 with a scale of one and no reverse is the default on power up and will be reinstated whenever control returns to the input prompt.

Other fonts can be loaded into memory, see the FONT LOAD command.



Illustration 158: Example of fonts.

Code for font example:

```
COLOR 0, 7
CLS

FONT #1
PRINT "CGMMSTICK"

FONT #1, 2
PRINT CLR$(BLUE, WHITE) "CGMMSTICK"

FONT #1, 3, 1
PRINT "CGMMSTICK"

FONT #2
PRINT "CGCOLORMAX"
```

```
FONT #2, 2
PRINT CLR$(BLACK, RED) "CGCOLORMAX"

FONT #2, 3, 1
PRINT "CGCOLORMAX"

FONT #3, 1
PRINT "123"

FONT #3, 1
PRINT CLR$(RED, YELLOW) "123"

FONT #3, 2, 1
PRINT "123"
```

See also:

FONT LOAD/UNLOAD

FONT LOAD/UNLOAD

FONT LOAD file\$ AS #nbr

Will load the font contained in 'file\$' and install it as font 'nbr' which can be any number between 3 and 10, the # symbol is optional. The font is loaded into the memory area used by arrays and strings, use the MEMORY command to check the usage of this area.

FONT UNLOAD #nbr

Removes font 'nbr' and free the memory used, the # symbol is optional. You cannot unload the built in fonts.

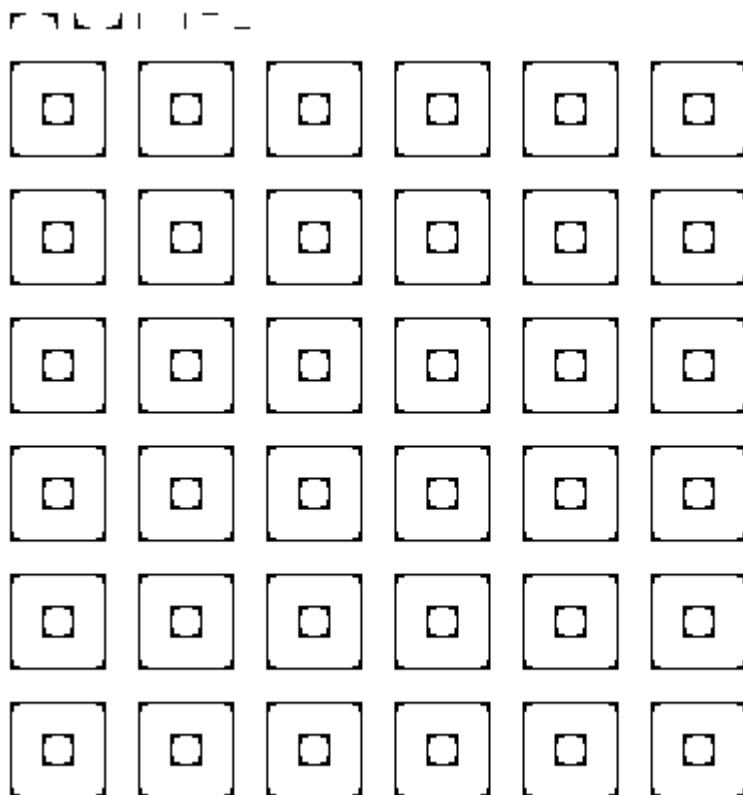


Illustration 159: Example of using a font file to create block graphics.

Block graphic font example code:

```
DRIVE "B:"  
FONT LOAD "parts.fnt" AS #4  
FONT #4  
  
PRINT "I"  
PRINT "IAIBICIDIEIFIGIH"  
PRINT "I"  
  
FOR A = 1 to 6  
PRINT "I"  
PRINT "IAGGGGBIIAGGGGBIIAGGGBIIAGGGBIAGGGBIIAGGGBI"  
PRINT "IEIIIIIFIIEIIIFIIEIIIFIIEIIIFIIEIIIFIIEIIIFI"  
PRINT "IEIABIFIIEIABIFIIEIABIFIIEIABIFIIEIABIFI"  
PRINT "IEICDIFIIEICDIFIIEICDIFIIEICDIFIIEICDIFI"  
PRINT "IEIIIIIFIIEIIIFIIEIIIFIIEIIIFIIEIIIFIIEIIIFI"  
PRINT "ICHHHHDIICHHHHDIICHHHHDIICHHHHDIICHHHHDIICHHHHDI"  
PRINT "I"  
NEXT A
```

Font definition file “parts.fnt” for example:

```
8,8,65,73  
'A  
*****  
****  
**  
**  
*  
*  
*  
*  
'B  
*****  
***  
**  
**  
**  
*
```

```
*  
*  
' C  
*  
*  
*  
*  
**  
**  
**  
***  
*****  
' D  
*  
*  
*  
**  
**  
**  
***  
*****  
' E  
*  
*  
*  
*  
*  
*  
*  
*  
*  
*  
* F  
*  
*  
*  
*  
*  
*  
*  
*  
* G  
*****
```

```
' H
```

```
*****
```

```
' I
```

```
' End of font file
```

See also:

FONT

INKEY\$

INKEY\$

Checks the keyboard and USB input buffers and, if there is one or more characters waiting in the queue, will remove the first character and return it as a single character in a string.

If the input buffer is empty this function will immediately return with an empty string (ie, "").

See also:

INPUT (from keyboard) **LINE INPUT (from keyboard)**
KEYDOWN

INPUT (from keyboard)

INPUT ["prompt string\$";] list of variables

Allows input from the keyboard to a list of variables. The input command will prompt with a question mark (?).

The input must contain commas to separate each data item if there is more than one variable.

For example, if the command is:

```
INPUT a, b, c
```

And the following is typed on the keyboard: 23, 87, 66

Then a = 23 and b = 87 and c = 66

If the "prompt string\$" is specified it will be printed before the question mark. If the prompt string is terminated with a comma (,) rather than the semicolon (;) the question mark will be suppressed.

See also:

INKEY\$
KEYDOWN

**LINE INPUT (from
keyboard)**

KEYDOWN (keyboard)

KEYDOWN

Return the decimal ASCII value of the PS2 keyboard key that is currently held down or zero if no key is down.

Note that this function will only work with the attached PS2 keyboard and the character will still be entered into the input keyboard buffer (including repeated characters if the key is held down long enough).

KEYDOWN clears the keyboard buffer.

See also:

INKEY\$

**LINE INPUT (from
keyboard)**

INPUT (from keyboard)

LINE

LINE [(x1 , y1)] - (x2, y2) [,c [,B[F]]]

Draws a line or box on the video screen.

x1,y1 and x2,y2 specify the beginning and end points of a line.

'c' specifies the color and defaults to the default foreground color if not specified. It can also be -1 to invert the pixels.

(x1, y1) is optional and if omitted the last drawing point will be used.

The optional B will draw a box with the points (x1,y1) and (x2,y2) at opposite corners. The optional BF will draw a box and fill the interior.

Example code that uses the box capabilities of the LINE command:

```
MODE 3
CLS

FOR a = 1 to 40
    LINE (RND()*480 , RND()*432) - (RND()*480, RND()*432), RND()*6+1, B
    ' Version with fill
    'LINE (RND()*480 , RND()*432) - (RND()*480, RND()*432), RND()*6+1,
BF
NEXT a
```

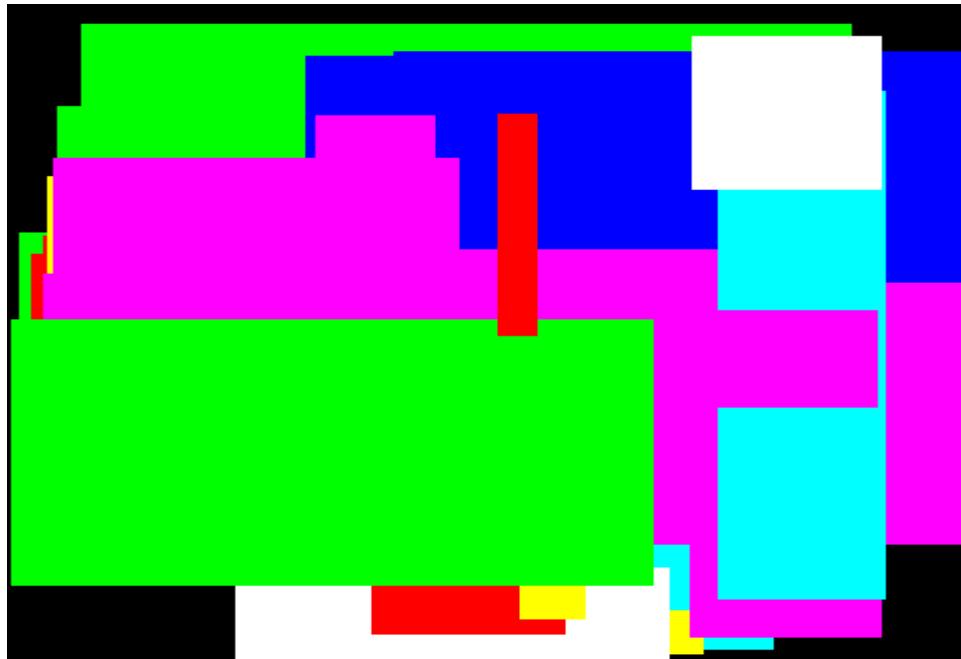
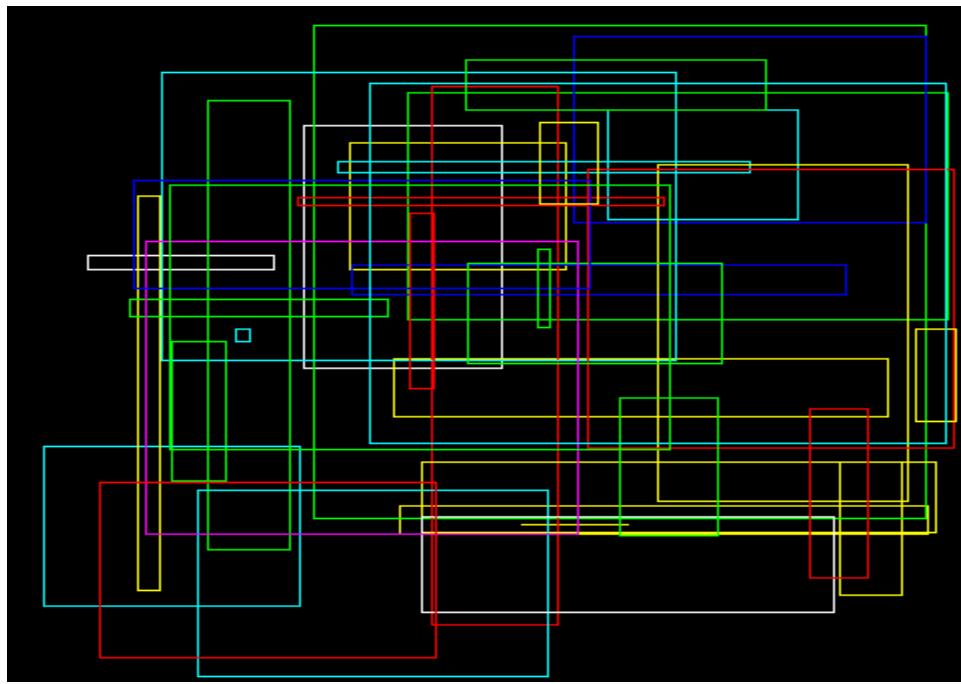


Illustration 160: Two outputs of program that draws boxes on screen.



Illustration 161: Using lines to write a name in cursive.

Code for cursive name example:

```
COLOR 0, 7
CLS

For loop = 1 to 50
    ' Jos point 1 and 2
    LINE (96+ RND()*2 , 208+ RND()*2) - (90+ RND()*2, 224+ RND()*2)

    ' Jos 3-50
    For a = 1 to 48
        if loop = 1 then pause 80
        READ x : READ y
        LINE - (x + RND()*2, y + RND()*2)
    next a

    ' ette point 1 and 2
    LINE (246+ RND()*2 , 240+ RND()*2) - (270+ RND()*2, 236+ RND()*2)

    ' ette 3-42
    For a = 1 to 40
        if loop = 1 then pause 80
        READ x : READ y
        LINE - (x + RND()*2, y + RND()*2)
    next a

    ' Cross tt
    LINE (306+ RND()*2 , 120+ RND()*2) - (456+ RND()*2, 104+ RND()*2)

    RESTORE
Next loop

'Jos points 3-50
DATA 84,240, 72,256, 60,260
DATA 48,256, 39,240, 36,216, 42,176, 48,144

DATA 66,88, 84,56, 102,32, 126,8, 141,4
DATA 150,8, 159,32, 162,72, 162,120, 156,184

DATA 144,264, 132,320, 114,360, 96,388, 84,396
DATA 72,400, 60,400, 48,384, 42,360, 48,336
```

```
DATA 54,312, 72,280, 120,224, 198,152, 198,188
DATA 192,208, 186,224, 177,236, 172,224, 180,192

DATA 198,152, 201,184, 210,188, 222,180, 258,136
DATA 258,208, 252,232, 246,240, 228,216, 219,184

'ette points 3-42
DATA 288,208, 297,184, 300,168
DATA 300,152, 297,144, 284,168, 276,184, 273,208

DATA 282,224, 288,236, 294,240, 306,232, 321,208
DATA 333,176, 354,96, 345,176, 342,200, 342,224

DATA 351,232, 366,208, 378,176, 396,96, 384,176
DATA 378,200, 381,224, 390,232, 408,224, 426,208

DATA 438,184, 447,160, 447,136, 444,124, 438,128
DATA 426,144, 417,176, 414,200, 417,224, 432,240

DATA 456,244, 474,240
```

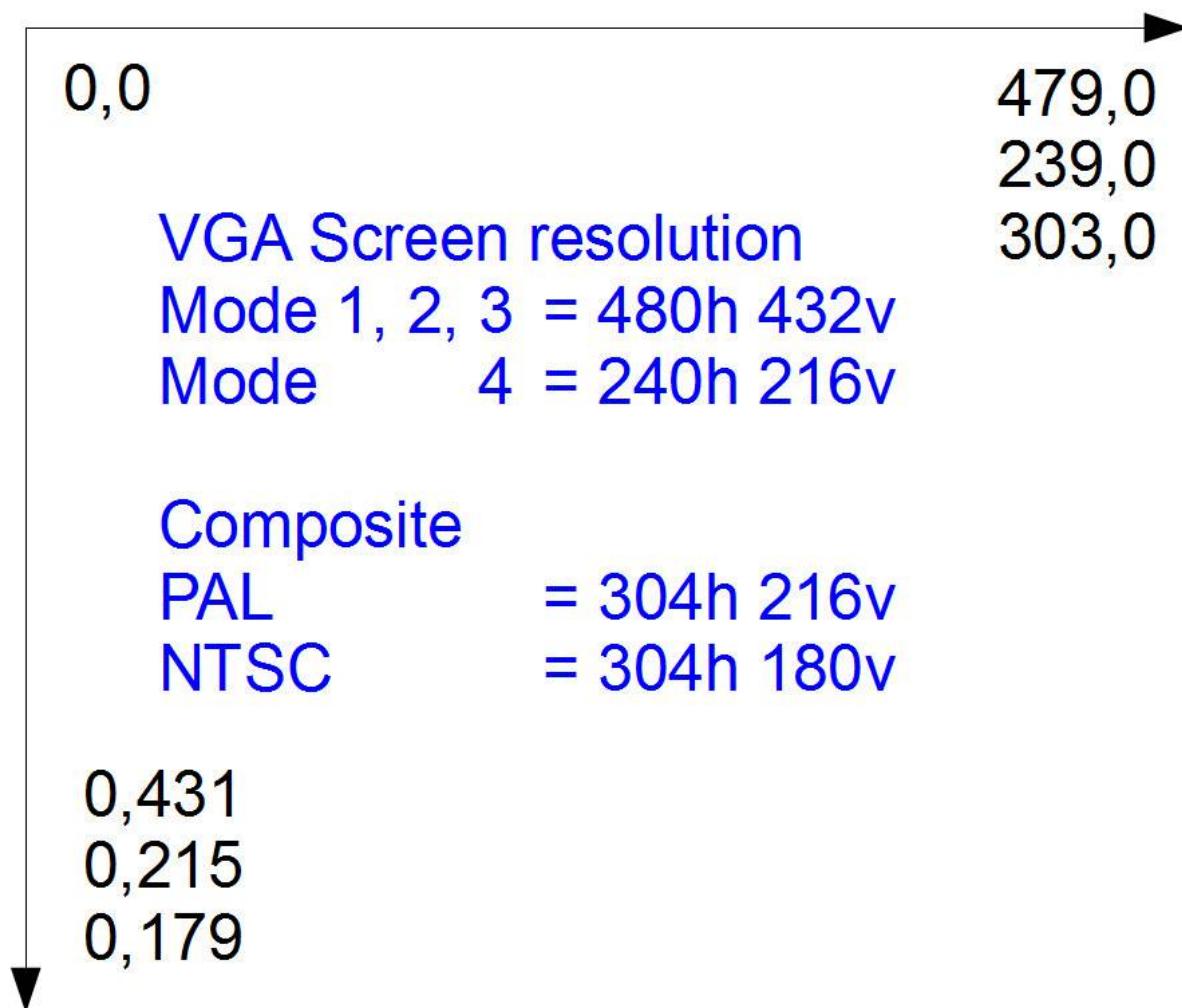


Illustration 162: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HRES	PRINT
CIRCLE	MM.VPOS	PRINT @
CLR\$	MM.VRES	PSET
CLS	MODE	SCANLINE
COLOR	PIXEL	SPRITE
LOCATE	POS	
MM.HPOS	PRESET	

LINE INPUT (from keyboard)

LINE INPUT [prompt\$,] string-variable\$

Reads entire line from the keyboard into 'string-variable\$'. If specified the 'prompt\$' will be printed first. Unlike INPUT, LINE INPUT will read a whole line, not stopping for comma delimited data items. A question mark is not printed unless it is part of 'prompt\$'.

See also:

INKEY\$

KEYDOWN

INPUT (from keyboard)

LOADBMP

LOADBMP file\$ [, x, y]

Load a bit-mapped image and display it on the video screen.

"file\$" is the name of the file and 'x' and 'y' are the screen coordinates for the top left hand corner of the image. If the coordinates are not specified the image will be drawn at the top left hand position on the screen.

If an extension is not specified ".BMP" will be added to the file name.

The file must use either a monochrome or 16 colors (4 bit) color depth and be in the uncompressed BMP format. Microsoft's Paint program is recommended for creating suitable images.

See also:

SAVEBMP

LOCATE

LOCATE x, y

Positions the cursor to a location in pixels and the next PRINT command will place its output at this location. Only affects the video output.

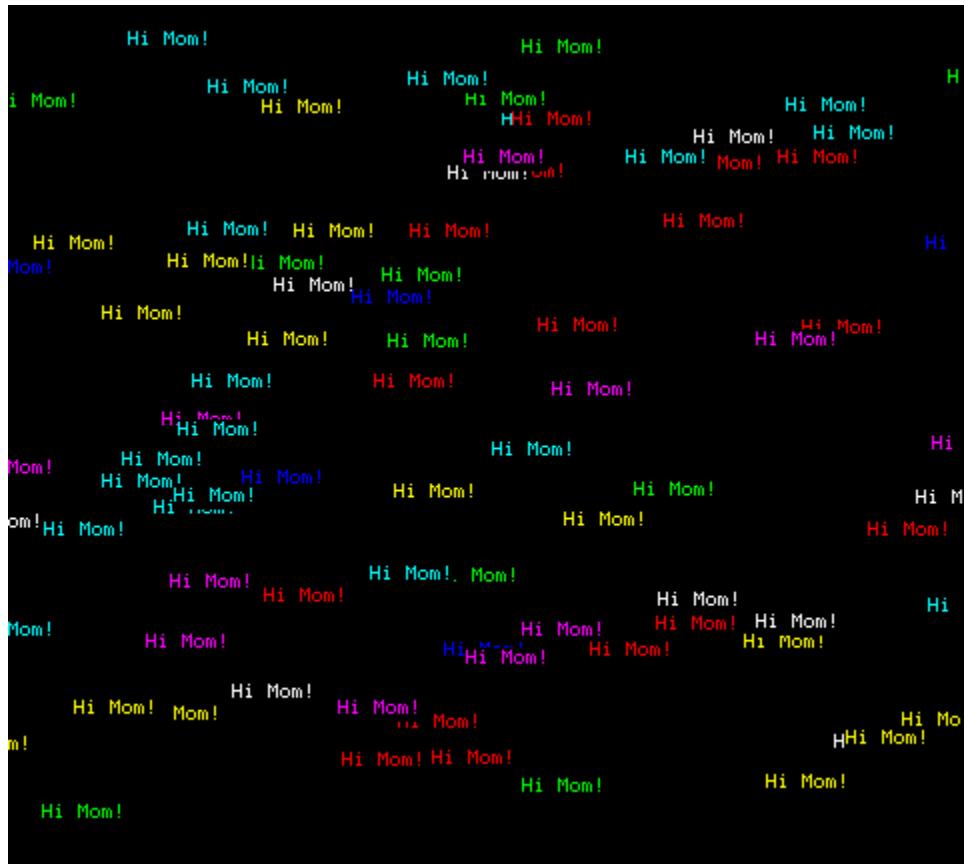


Illustration 163: LOCATE example output.

```
MODE 3
CLS

FOR a = 1 to 80
  LOCATE RND()*480,RND()*432
  COLOR RND()*6+1
  PRINT "Hi Mom!"
```

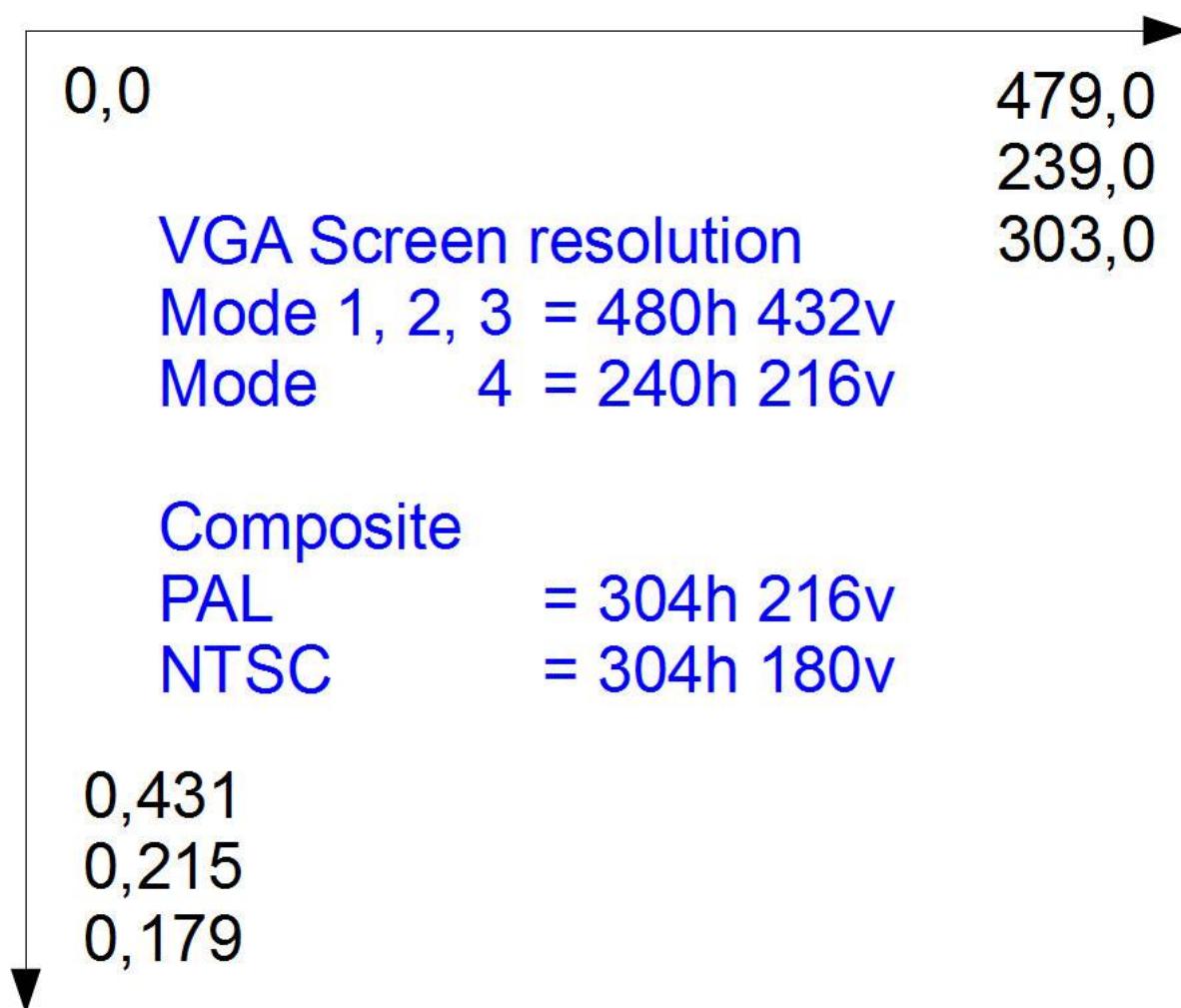


Illustration 164: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HPOS	POS
CIRCLE	MM.HRES	PRESET
CLR\$	MM.VPOS	PRINT
CLS	MM.VRES	PRINT @
COLOR	MODE	PSET
LINE	PIXEL	SCANLINE

SPRITE

MM.HPOS

MM.HPOS

The current horizontal position (in pixels) following the last graphic or print command.

```
CLS  
PRINT MM.HPOS  
  
LINE (0,0) - (100,100)  
  
PRINT MM.HPOS
```

```
0  
100
```

See also:

BLIT	MM.HRES	PRINT
CIRCLE	MM.VPOS	PRINT @
CLR\$	MM.VRES	PSET
CLS	MODE	SCANLINE
COLOR	PIXEL	SPRITE
LINE	POS	
LOCATE	PRESET	

MM.HRES

MM.HRES

The horizontal resolution of the current video display screen in pixels.

```
PRINT MM.HRES
```

```
480
```

0,0	479,0
	239,0
VGA Screen resolution	303,0
Mode 1, 2, 3 = 480h 432v	
Mode 4 = 240h 216v	
Composite	
PAL	= 304h 216v
NTSC	= 304h 180v
0,431	
0,215	
0,179	

Illustration 165: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HPOS	PRINT
CIRCLE	MM.VPOS	PRINT @
CLR\$	MM.VRES	PSET
CLS	MODE	SCANLINE
COLOR	PIXEL	SPRITE
LINE	POS	
LOCATE	PRESET	

MM.VPOS

MM.VPOS

The current vertical position (in pixels) following the last graphic or print command.

```
CLS  
PRINT MM.VPOS  
PRINT MM.VPOS  
PRINT MM.VPOS  
PRINT MM.VPOS  
PRINT MM.VPOS  
PRINT MM.VPOS  
PRINT MM.VPOS
```

When run, the output shows that each PRINT statement moves the vertical position by 12 pixels.

```
0  
12  
24  
36  
48  
60
```

See also:

BLIT	MM.HPOS	PRINT
CIRCLE	MM.HRES	PRINT @
CLR\$	MM.VRES	PSET
CLS	MODE	SCANLINE
COLOR	PIXEL	SPRITE
LINE	POS	
LOCATE	PRESET	

MM.VRES

MM.VRES

The vertical resolution of the current video display screen in pixels.

```
PRINT MM.VRES
```

```
432
```

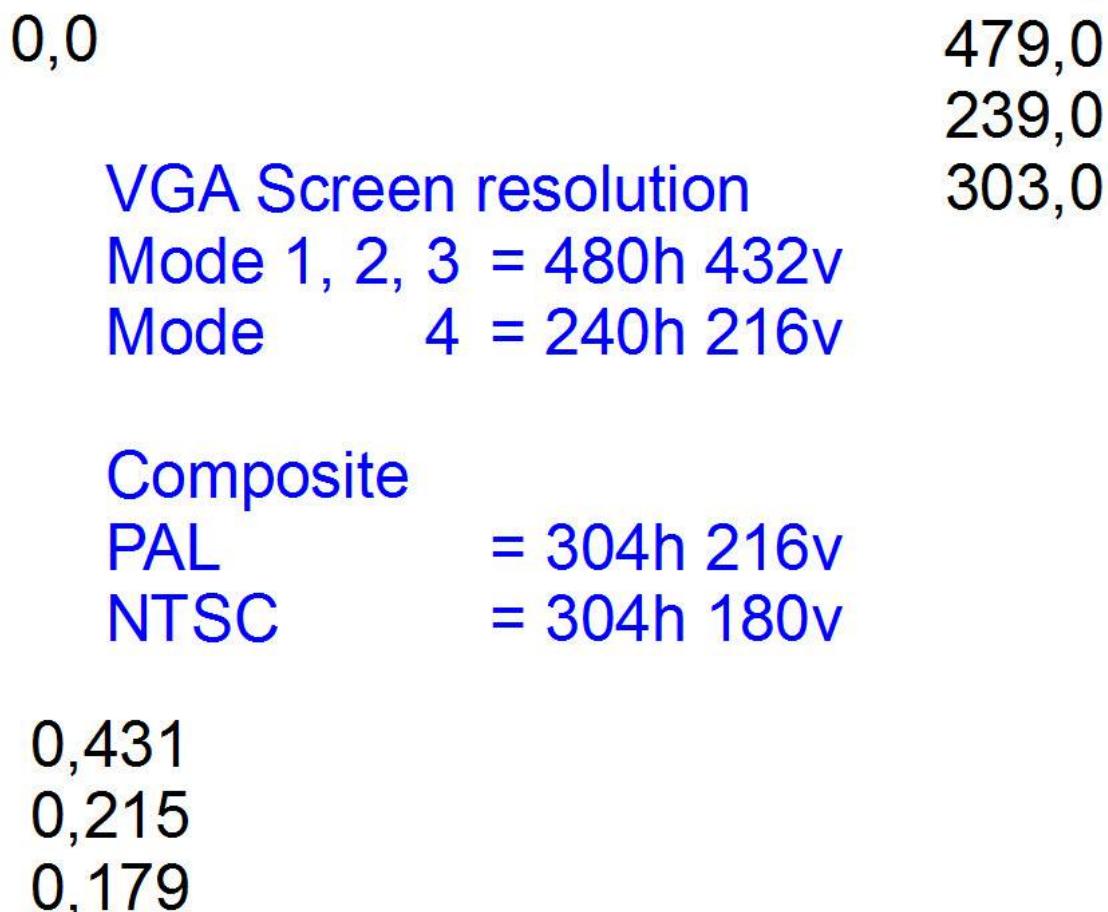


Illustration 166: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HPOS	PRINT
CIRCLE	MM.HRES	PRINT @
CLR\$	MM.VPOS	PSET
CLS	MODE	SCANLINE
COLOR	PIXEL	SPRITE
LINE	POS	
LOCATE	PRESET	

MODE

MODE mode [, palette]

Sets the number of colors that can be displayed on the screen.

'mode' can be:

- 1 Monochrome mode. 'palette' will select the color to use and can be 0 to 7 representing the colors black to white. This mode provides complete compatibility with programs written for the monochrome Maximite.
- 2 Four color mode. 'palette' can be a number from 1 to 6 and will select the range of colors available.

In mode 2 the colors available in each palette are:

palette = 1 Black, Red, Green, Yellow
palette = 2 Black, Red, Blue, Purple
palette = 3 Black, Red, Cyan, White
palette = 4 Black, Green, Blue, Cyan
palette = 5 Black, Green, Purple, White
palette = 6 Black, Blue, Yellow, White

- 3 Eight color mode. In this mode all eight colors (including black and white) can be used. 'palette' can be supplied but will be ignored.
- 4 240x216 pixel resolution with all eight colors (including black and white) available. 'palette' can be supplied but will be ignored.

Monochrome Mode

Any one colour can be selected for all output:

MODE 1,1	MODE 1,2	MODE 1,3	MODE 1,4	MODE 1,5	MODE 1,6	MODE 1,7
Blue	Green	Cyan	Red	Purple	Yellow	White

Four colour mode

Six colour palettes to chose from.

Each palette consists of three colours plus black:

Palette 1	MODE 2,1				(Red)	Yellow	Green)
Palette 2	MODE 2,2				(Red)	Blue	Purple)
Palette 3	MODE 2,3				(Red)	Cyan	White)
Palette 4	MODE 2,4				(Green)	Cyan	Blue)
Palette 5	MODE 2,5				(Green)	Purple	White)
Palette 6	MODE 2,6				(Yellow)	Blue	White)

Eight colour mode

MODE 3 All colours can be used simultaneously

240x216 eight colour mode

MODE 4

(press any key to exit):

All Eight Colours 240x216 Pixels
High Speed Maximum Free Memory

Illustration 167: Color graphics. Examples of the MODE command. (Image actually generated completely in mode 3.)

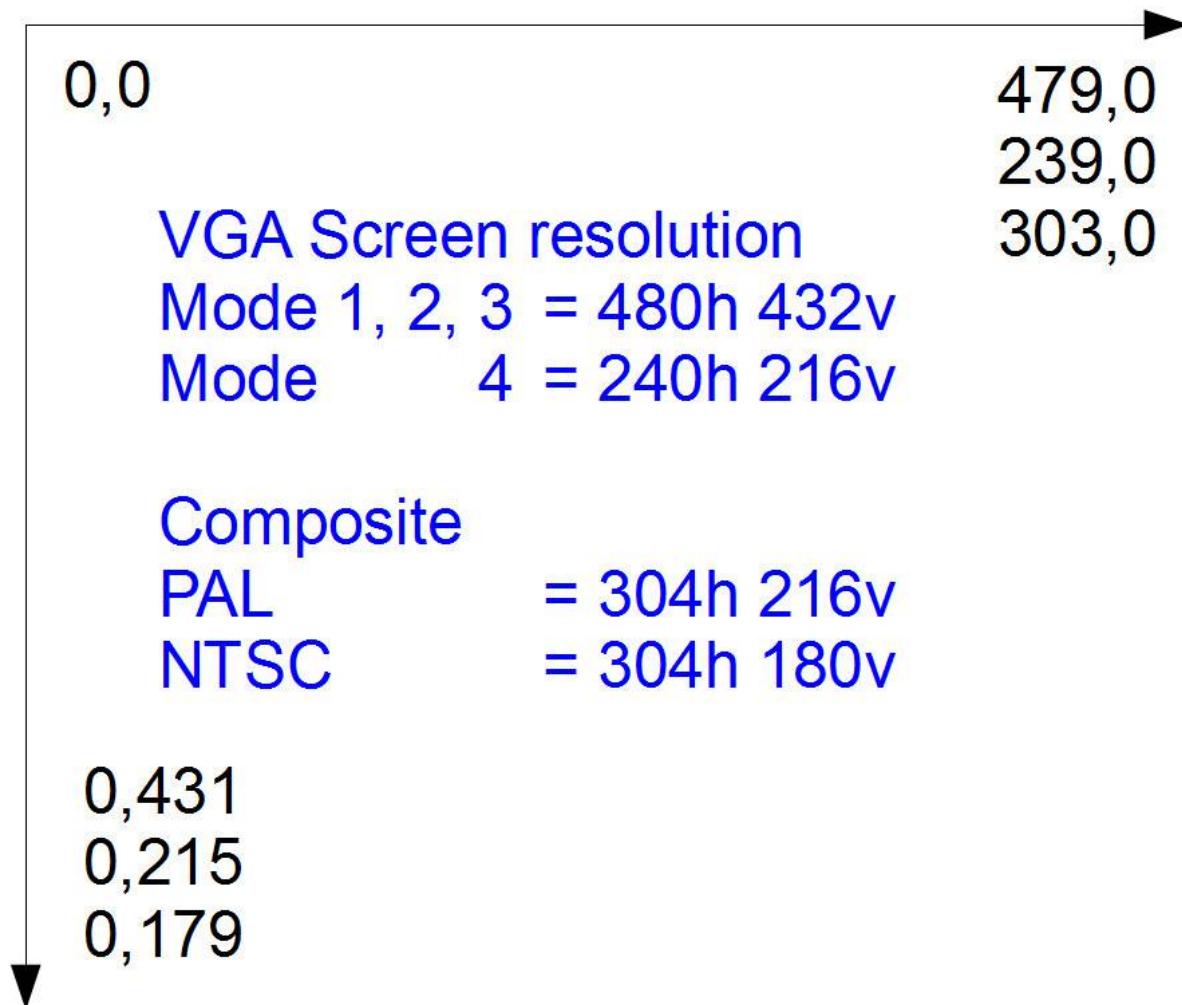


Illustration 168: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

The MODE command allows the programmer to trade the number of colors used and resolution against the amount of memory required by the video driver. Modes 1 and 4 use the least amount of memory while mode 3 uses the most.

See also:

BLIT	LINE	MM.VRES
CIRCLE	LOCATE	PIXEL
CLR\$	MM.HPOS	POS
CLS	MM.HRES	PRESET
COLOR	MM.VPOS	PRINT

PRINT @

SCANLINE

PSET

SPRITE

OPTION VIDEO

OPTION VIDEO ON | OFF

VIDEO OFF prevents the output from the PRINT command from being displayed on the video output (VGA or composite). The VIDEO ON option will revert to the normal action.

Normally this is used when a program wants to separately display data on the USB and video interfaces. This option is always reset to ON at the command prompt.

See also:

CONFIG COMPOSITE **CONFIG VIDEO**

PIXEL (command/function)

PIXEL(x,y) = color

Set a pixel on the VGA or composite screen to a color or inverted (if value is -1).

PIXEL(x, y)

PIXEL(x, y) returns the value of a pixel on the VGA or composite screen. Zero is off, 1 is on.

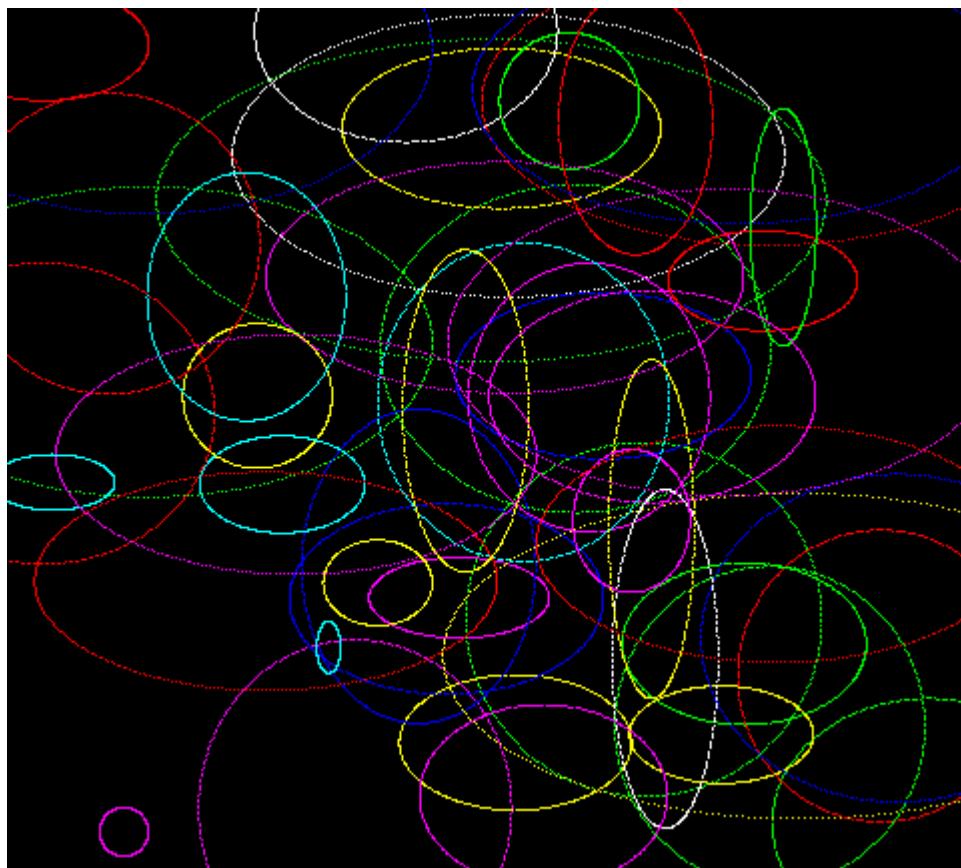


Illustration 169: PIXEL command example output.

PIXEL example code:

```
MODE 3
CLS

FOR a = 1 to 50
    OVAL( (RND*80)+10, (RND()*460)+10, (RND()*410)+10, (RND()*2)+.25, (RND()*6)+1 )
NEXT a

SUB OVAL(oval_radius, oval_x, oval_y, oval_aspect, oval_color)
    FOR degr = 1 TO 360
        angle = ( degr / 57.29 )
        x = ( oval_radius * COS( angle ) ) * oval_aspect
        y = oval_radius * SIN( angle )
        PIXEL( x + oval_x, y + oval_y ) = oval_color
    NEXT degr
END SUB
```

Beginning Maximite
Programming & Interfacing
with the CircuitGizmos
CGMMSTICK and CGCOLORMAX
www.circuitgizmos.com

Beginning Maximite Programming & Interfacing with the CircuitGizmos CGMMSTICK and CGCOLORMAX

www.circuitgizmos.com

Beginning Maximite
Programming & Interfacing
with the CircuitGizmos
CGMMSTICK and CGCOLORMAX
www.circuitgizmos.com

Beginning Maximite Programming & Interfacing with the CircuitGizmos CGMMSTICK and CGCOLORMAX

www.circuitgizmos.com

Illustration 170: Two outputs of program that reads the text on screen and makes a larger version using small circles for each pixel.

PIXEL reading example code:

```
MODE 3
CLS

PRINT "Beginning Maximite"
PRINT "Programming & Interfacing"
PRINT "with the CircuitGizmos"
PRINT "CGMMSTICK and CGCOLORMAX"
PRINT
PRINT "www.circuitgizmos.com"

FOR a = 0 to 170
    FOR b = 0 to 72
        state = PIXEL(a, b)
        IF state <> 0 THEN
            CIRCLE (10+3*a,100+(3*b)), 2, YELLOW, F
            ' Alternate image
            'CIRCLE (10+3*a,100+(3*b)), 1, RND()*7+1, F
        ENDIF
    NEXT b
NEXT a
```

0,0		479,0
		239,0
VGA Screen resolution		303,0
Mode 1, 2, 3	= 480h 432v	
Mode 4	= 240h 216v	
Composite		
PAL	= 304h 216v	
NTSC	= 304h 180v	
0,431		
0,215		
0,179		

Illustration 171: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HPOS	PRINT
CIRCLE	MM.HRES	PRINT @
CLR\$	MM.VPOS	PSET
CLS	MM.VRES	SCANLINE
COLOR	MODE	SPRITE
LINE	POS	
LOCATE	PRESET	

POS

POS

Returns the current cursor position in the line in characters.

```
PRINT POS  
PRINT "Hello "; POS
```

```
1  
Hello 7
```

See also:

BLIT	MM.HPOS	PRINT
CIRCLE	MM.HRES	PRINT @
CLR\$	MM.VPOS	PSET
CLS	MM.VRES	SCANLINE
COLOR	MODE	SPRITE
LINE	PIXEL	
LOCATE	PRESET	

PRESET

PRESET

Turn off (PRESET) or on (PSET) a pixel on the video screen. Alternate to PIXEL.

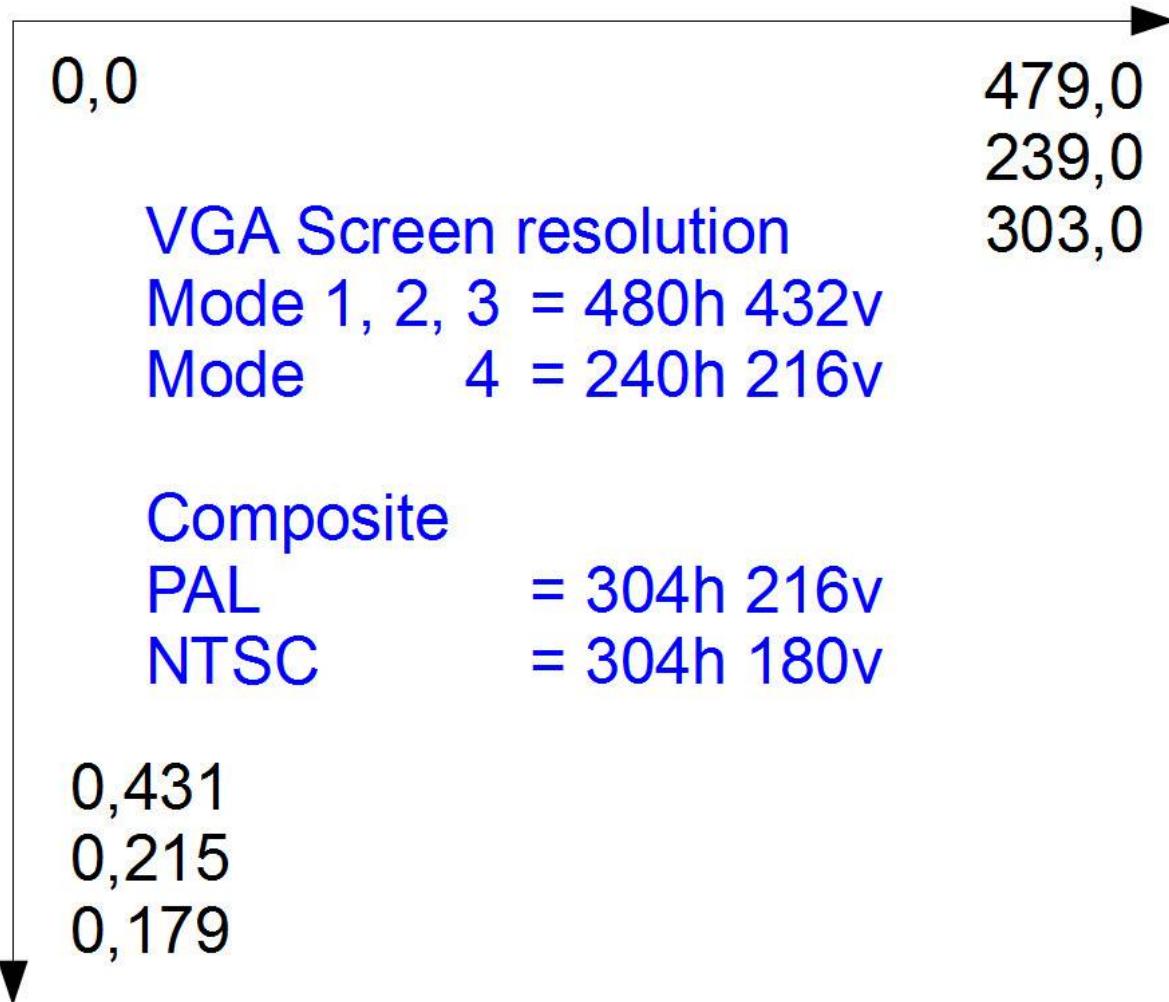


Illustration 172: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	CLS	LOCATE
CIRCLE	COLOR	MM.HPOS
CLR\$	LINE	MM.HRES

MM.VPOS	POS	SCANLINE
MM.VRES	PRINT	SPRITE
MODE	PRINT @	
PIXEL	PSET	

PRINT

PRINT expression [[,;]expression] ... etc

Outputs text to the screen. Multiple expressions can be used and must be separated by either a:

Comma (,) which will output the tab character

Semicolon (;) which will not output anything (it is just used to separate expressions).

Nothing or a space which will act the same as a semicolon.

A semicolon (;) at the end of the expression list will suppress the automatic output of a carriage return/ newline at the end of a print statement.

When printed, a number is preceded with a space if positive or a minus (-) if negative but is not followed by a space. Integers (whole numbers) are printed without a decimal point while fractions are printed with the decimal point and the significant decimal digits. Large numbers (greater than six digits) are printed in scientific format.

The function FORMAT\$() can be used to format numbers. The function TAB() can be used to space to a certain column and the string functions can be used to justify or otherwise format strings.

A single question mark (?) can be used as a shortcut for the PRINT keyword.

See also:

BLIT	MM.HPOS	PRESET
CIRCLE	MM.HRES	PRINT @
CLR\$	MM.VPOS	PSET
CLS	MM.VRES	SCANLINE
COLOR	MODE	SPRITE
LINE	PIXEL	
LOCATE	POS	

PRINT @

PRINT @(x, y [, m]) expression

Same as the PRINT command except that the cursor is positioned at the coordinates x, y.

Example:

```
PRINT @(150, 45) "Hello World"
```

The @ function can be used anywhere in a print command.

Example:

```
PRINT @(150, 45) "Hello"  @(150, 55)  "World"
```

The @(x,y) function can be used to position the cursor anywhere on or off the screen. For example PRINT @(-10, 0) "Hello" will only show "llo" as the first two characters could not be shown because they were off the screen.

The @(x,y) function will automatically suppress the automatic line wrap normally performed when the cursor goes beyond the right screen margin.

If 'm' is specified the mode of the video operation will be as follows:

m = 0 Normal text (white letters, black background)

m = 1 The background will not be drawn (ie, transparent)

m = 2 The video will be inverted (black letters, white background)

m = 5 Current pixels will be inverted (transparent background)

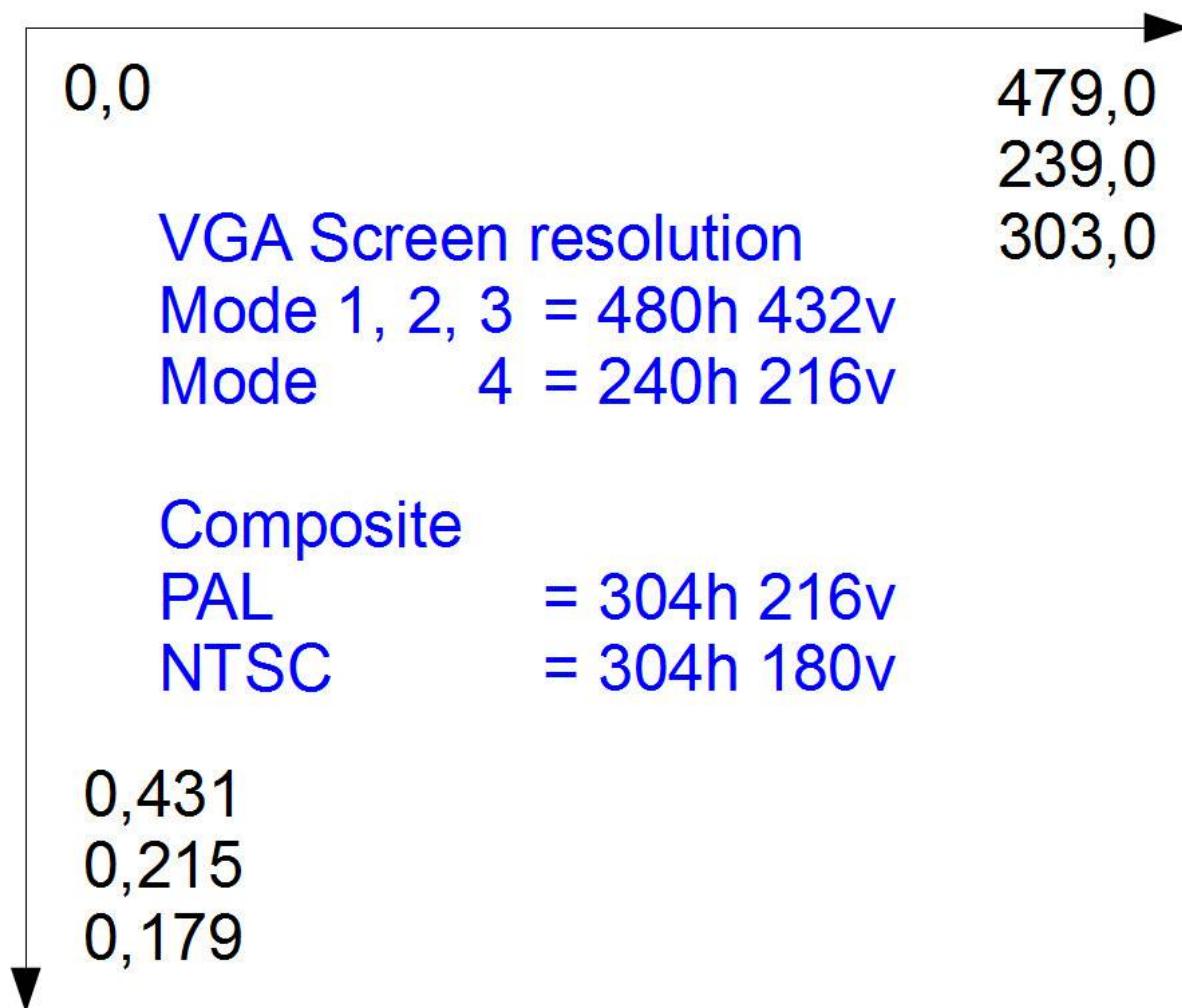


Illustration 173: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	MM.HPOS	PRESET
CIRCLE	MM.HRES	PRINT
CLR\$	MM.VPOS	PSET
CLS	MM.VRES	SCANLINE
COLOR	MODE	SPRITE
LINE	PIXEL	
LOCATE	POS	

PSET

PSET

Turn off (PRESET) or on (PSET) a pixel on the video screen. Alternate to PIXEL.

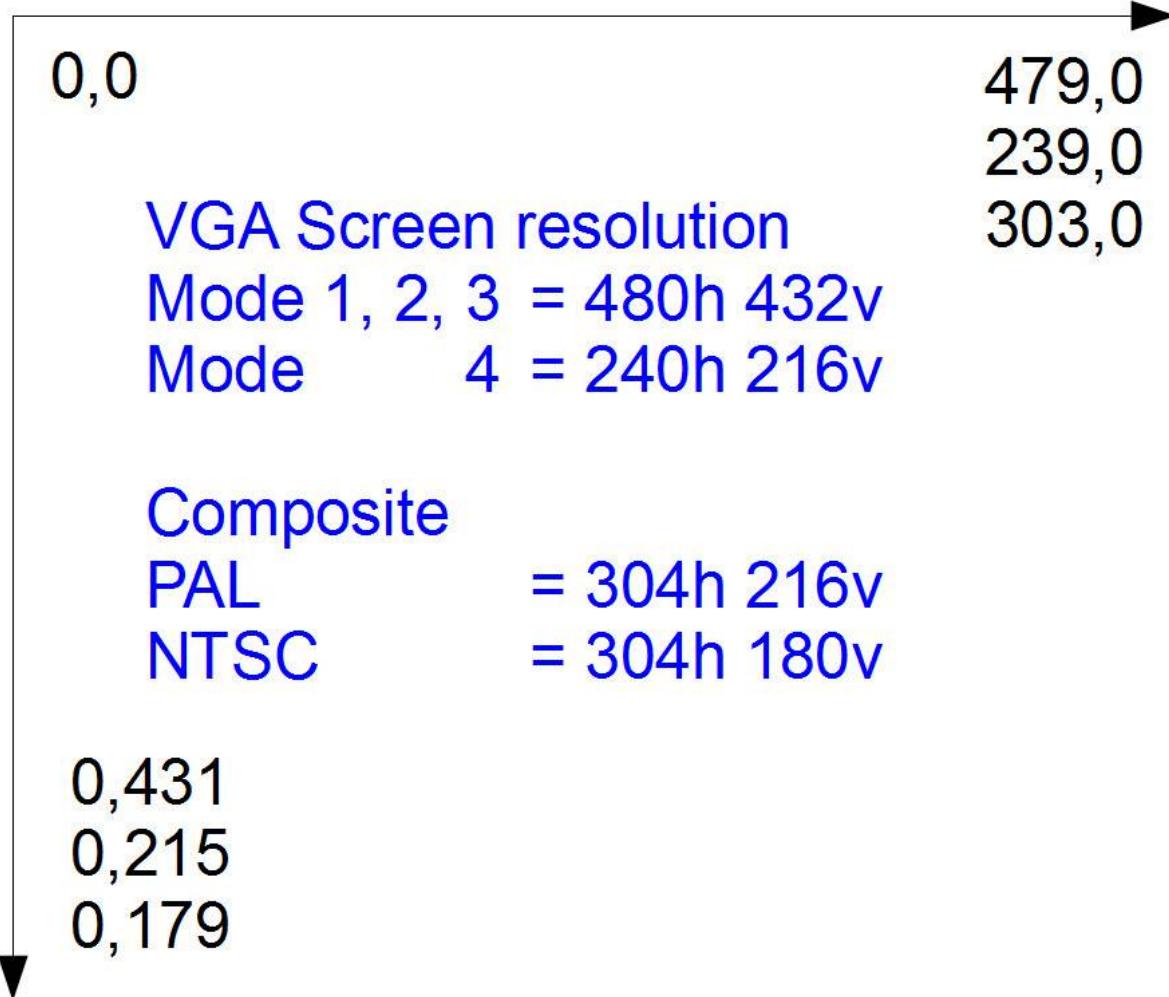


Illustration 174: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT

CLS

LOCATE

CIRCLE

COLOR

MM.HPOS

CLR\$

LINE

MM.HRES

MM.VPOS	POS	SCANLINE
MM.VRES	PRESET	SPRITE
MODE	PRINT	
PIXEL	PRINT @	

SAVEBMP

SAVEBMP file\$

Saves the current VGA or composite screen as a BMP file in the current working directory on the current drive. The CGCOLORMAX will save the file as a 16 color (4 bit) file.

Example:

```
SAVEBMP "IMAGE.BMP"
```

If an extension is not specified ".BMP" will be added to the file name.

Note that Windows 7 Paint has trouble displaying monochrome images. This appears to be a bug in Paint as all other software tested (including Windows XP Paint) can display the image without fault.

See also:

LOADBMP

SCANLINE

SCANLINE color, startline [, endline]

This command can be used to set the color of individual horizontal scan lines on the VGA monitor when in MODE 1,7 (monochrome with white foreground). This applies to all video output displayed before and after the SCANLINE command.

'color' can be any color specified by name or number from 0 to 7.

'startline' and 'endline' specify the range of scan lines to set. If 'endline' is not specified only one line will be set. Multiple calls to SCANLINE can be used to set the color of other scan lines or to change the color of lines already set (ie, the settings accumulate).

This command is valid only when the color mode is set to MODE 1,7 (monochrome with the color set to white). All settings made by SCANLINE are automatically canceled whenever the MODE command is used or when MMBasic returns to the command prompt.

```
CircuitGizmos
```

Illustration 175: SCANLINE example as it would appear on your VGA display.

Example SCANLINE code:

```
MODE 1,7

FOR a = 1 TO 35
    PRINT "CircuitGizmos"
NEXT a

SCANLINE RED, 0, 105
SCANLINE YELLOW, 106, 200
SCANLINE BLUE, 201, 300

PAUSE 4000
```

Note that a SAVEBMP action *does not* capture the colors.

See also:

BLIT	MM.HPOS	PRESET
CIRCLE	MM.HRES	PRINT
CLR\$	MM.VPOS	PRINT @
CLS	MM.VRES	PSET
COLOR	MODE	SPRITE
LINE	PIXEL	
LOCATE	POS	

sprite

sprite load file

sprite on n, x, y [, color]

sprite move n, x, y [, color]

sprite off n [, n [, ...]]

sprite off all

sprite unload

Load and manipulate sprites on the screen. Sprites are 16x16 pixel objects that can be moved about on the screen without erasing or disturbing text or other underlying graphics. This command is mostly used in animated games.

'n' is the sprite number, 'x' and 'y' are the sprite's coordinates on the screen.

SPRITE LOAD will load a sprite file into memory. This file defines the graphic images of one or more sprites.

SPRITE ON will display an individual sprite contained in the sprite file.

SPRITE MOVE will move the sprite to a new location and restore the background at the old location.

For both ON and MOVE 'color' can be optionally specified and this color will be used for the background. When using a solid background color this is faster and does not require special handling for overlapping sprites.

SPRITE OFF will remove the sprite from the screen and restore the background graphics that was obscured when the sprite was turned on.

A number of sprites may be quickly removed in sequence by specifying a comma separated list. ie, SPRITE OFF 4, 8, 2, 3. As a shortcut SPRITE OFF ALL will remove all active sprites.

SPRITE UNLOAD will disable the sprites, unload the file and reclaim the memory used.

The sprite file can contain many individual sprites which can be simultaneously displayed and independently manipulated at the same time. The number of sprites is limited only by the available memory.

0,0		479,0
		239,0
VGA Screen resolution		303,0
Mode 1, 2, 3	= 480h 432v	
Mode 4	= 240h 216v	
Composite		
PAL	= 304h 216v	
NTSC	= 304h 180v	
0,431		
0,215		
0,179		

Illustration 176: VGA screen resolution is 480 horizontal by 432 vertical in modes 1-3, and 240x416 in mode 4. PAL composite is 304 by 216 and NTSC composite is 304 by 180.

See also:

BLIT	LOCATE	POS
CIRCLE	MM.HPOS	PRESET
CLR\$	MM.HRES	PRINT
CLS	MM.VPOS	PRINT @
COLLISION	MM.VRES	PSET
COLOR	MODE	SCANLINE
LINE	PIXEL	

TAB

TAB(number)

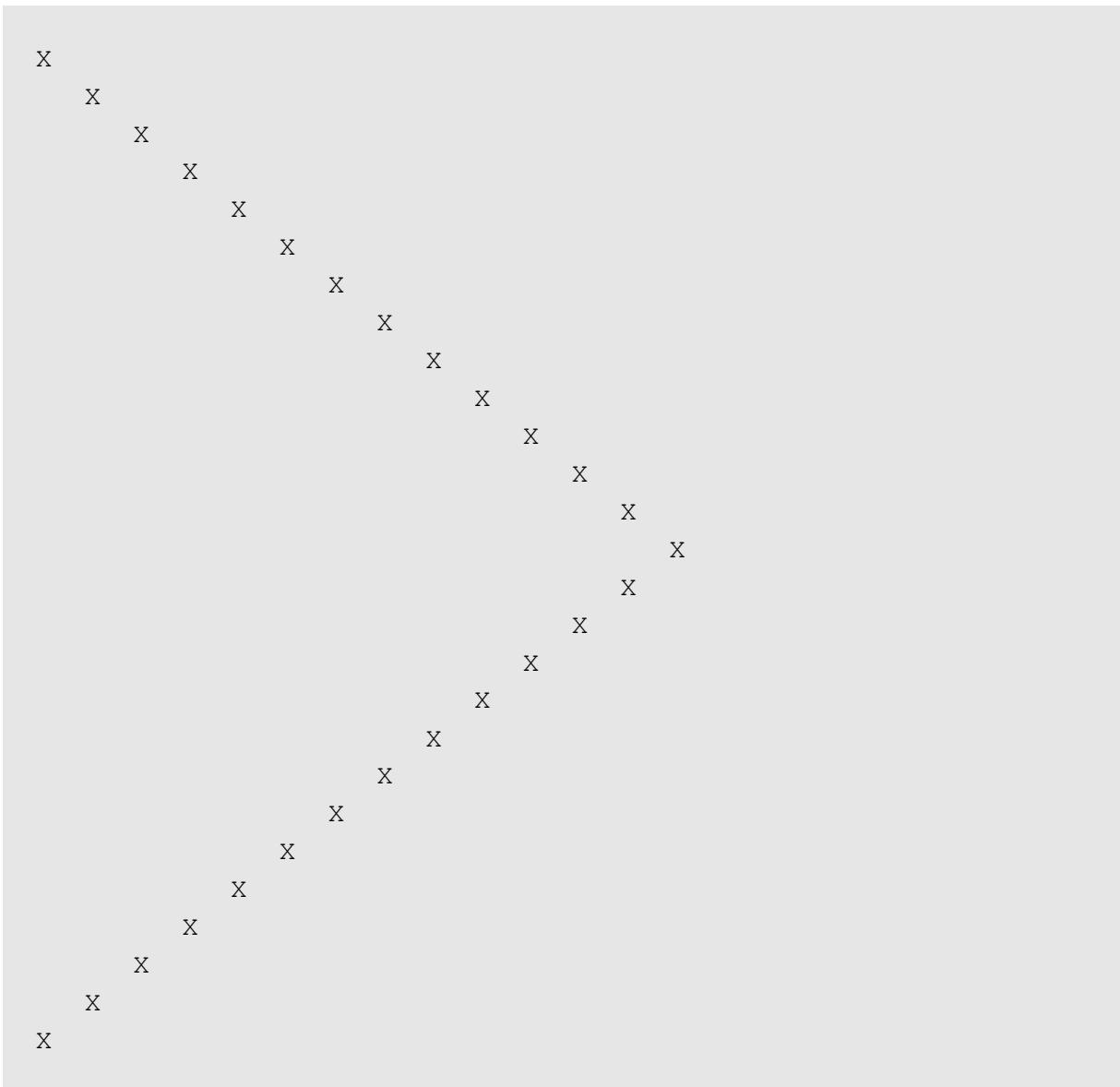
Outputs spaces until the column indicated by 'number' has been reached.

Example TAB program:

```
FOR a = 1 TO 14
    PRINT TAB( a * 3 ) "X"
NEXT A

FOR a = 13 TO 1 STEP -1
    PRINT TAB( a * 3 ) "X"
NEXT A
```

Output of TAB example:



See also:

PRINT

PRINT @

MMBasic math functions

Numeric constants may begin with a numeric digit (0-9) for a decimal constant, &H for a hexadecimal constant, &O for an octal constant or &B for a binary constant. For example &B1000 is the same as the decimal constant 8.

Decimal constants may be preceded with a minus (-) or plus (+) and may be terminated with 'E' followed by an exponent number to denote exponential notation. For example 1.6E+4 is the same as 16000.

OPERATORS AND PRECEDENCE

The following operators, in order of precedence, are recognized. Operators that are on the same level (for example + and -) are processed with a left to right precedence as they occur on the program line.

Arithmetic operators	
^	Exponentiation
* / \ MOD	Multiplication, division, integer division and modulus (remainder)
+ -	Addition and subtraction

ABS

ABS(number)

Returns the absolute value of the argument 'number' (ie, any negative sign is removed and the positive number is returned).

```
PRINT ABS(123)
```

```
123
```

```
PRINT ABS(123.45)
```

```
123.45
```

```
PRINT ABS(-123)
```

```
123
```

```
PRINT ABS(0)
```

```
0
```

See also:

CINT

FIX

SGN

EXP

INT

ATN

ATN(number)

Returns the arc tangent value of the argument 'number' in radians.

```
PRINT ATN(2)  
1.10715
```

The range of radians is from $-\pi/2$ (-1.5708) to $\pi/2$ (1.5708).

ATN is the inverse of TAN. If $z = \text{ATN}(x)$, then $x = \text{TAN}(z)$.

If you want the result in degrees:

```
PRINT DEG(ATN(2))  
63.4349
```

```
PRINT DEG(1.10715)  
63.435
```

See also:

COS

PI

TAN

DEG

SIN

LOG

SQR

CINT

CINT(number)

Round numbers with fractional portions up or down to the next whole number or integer.

```
PRINT CINT(45.47)
```

```
45
```

```
PRINT CINT(45.57)
```

```
46
```

```
PRINT CINT(-34.45)
```

```
-34
```

```
PRINT CINT(-34.55)
```

```
-35
```

See also:

ABS

FIX

SGN

EXP

INT

COS

COS(number)

Returns the cosine of the argument 'number' in radians.

```
PRINT COS(.6)  
0.825336
```

See also:

ATN

PI

TAN

DEG

SIN

LOG

SQR

DEG

DEG(radians)

Converts 'radians' to degrees.

```
PRINT DEG(.7853981)  
45
```

See also:

ATN

PI

SQR

COS

RAD

TAN

LOG

SIN

EXP

EXP(number)

Returns the exponential value of 'number'.

```
PRINT EXP(10)  
22026.5
```

See also:

ABS

FIX

SGN

CINT

INT

FIX

FIX(number)

Truncate a number to a whole number by eliminating the decimal point and all characters to the right of the decimal point.

For example 9.89 will return 9 and -2.11 will return -2.

The major difference between FIX and INT is that FIX provides a true integer function (ie, does not return the next lower number for negative numbers as INT() does). This behavior is for Microsoft compatibility.

```
PRINT FIX(12.6)
```

```
12
```

```
PRINT FIX(-12.6)
```

```
-12
```

```
PRINT INT(12.6)
```

```
12
```

```
PRINT INT(-12.6)
```

```
-13
```

See also:

ABS

EXP

SGN

CINT

INT

INT

INT(number)

Truncate an expression to the next whole number less than or equal to the argument. For example 9.89 will return 9 and -2.11 will return -3. This behavior is for Microsoft compatibility, the FIX() function provides a true integer function.

```
PRINT FIX(12.6)
```

```
12
```

```
PRINT FIX(-12.6)
```

```
-12
```

```
PRINT INT(12.6)
```

```
12
```

```
PRINT INT(-12.6)
```

```
-13
```

See also:

ABS

EXP

SGN

CINT

FIX

LOG

LOG(number)

Returns the natural logarithm of the argument 'number'.

```
PRINT LOG(2)  
0.693147
```

See also:

ATN

PI

TAN

COS

SIN

DEG

SQR

PI

PI

Returns the value of pi.

```
PRINT "Mmmmm. pi" PI  
Mmmmm. pi 3.14159
```

See also:

ATN

LOG

TAN

COS

SIN

DEG

SQR

RAD

RAD(degrees)

Converts 'degrees' to radians.

```
PRINT RAD(30)  
0.523599
```

See also:

ATN

LOG

SQR

COS

PI

TAN

DEG

SIN

RND

RND(number)

Returns a pseudo random number in the range of 0 to 0.99999. The 'number' value is ignored if supplied. The RANDOMIZE command reseeds the random number generator.

```
PRINT RND  
0.513871
```

```
PRINT RND  
0.175726
```

```
PRINT RND  
0.308634
```

See also:

RANDOMIZE

SGN

SGN(number)

Returns the sign of the argument 'number', +1 for positive numbers, 0 for 0, and -1 for negative numbers.

```
PRINT SGN(123)
```

```
1
```

```
PRINT SGN(-123)
```

```
-1
```

```
PRINT SGN(0)
```

```
0
```

See also:

ABS

EXP

INT

CINT

FIX

SIN

SIN(number)

Returns the sine of the argument 'number' in radians.

```
PRINT SIN(1.25)  
0.948985
```

See also:

ATN

LOG

TAN

COS

PI

DEG

SQR

SQR

SQR(number)

Returns the square root of the argument 'number'.

```
PRINT SQR(1764)  
42
```

See also:

ATN

LOG

TAN

COS

PI

DEG

SIN

TAN

TAN(number)

Returns the tangent of the argument 'number' in radians.

```
PRINT TAN(3)  
-0.142547
```

The range of radians is from $-\pi/2$ (-1.5708) to $\pi/2$ (1.5708).

ATN is the inverse of TAN. If $z = \text{ATN}(x)$, then $x = \text{TAN}(z)$.

If you want the result in degrees:

```
PRINT DEG(TAN(3))  
-8.16731  
  
PRINT DEG(-0.142547)  
-8.16731
```

See also:

ATN	LOG	SQR
COS	PI	
DEG	SIN	

MMBasic character and string functions

MMBasic supports manipulation of ASCII characters as well as text strings.

EXPRESSIONS

In most cases where a number or string is required you can also use an expression. For example:

```
FNAME$ = "TEST"  
RUN FNAME$ + ".BAS"
```

String constants are surrounded by double quote marks (""). Eg, "Hello World".

String operators	
+	Join two strings
<> < > <= =< >= =>	Inequality, less than, greater than, less than or equal to, less than or equal to (alternative version), greater than or equal to, greater than or equal to (alternative version)
=	Equality

ASC

ASC(string\$)

Returns the ASCII code for the first letter in the argument 'string\$'.

```
PRINT ASC ("MMBasic")
77
```

```
PRINT ASC ("M")
77
```

See also:

BIN\$

HEX\$

STR\$

CHR\$

LEN

TIME\$

DATE\$

OCT\$

VAL

BIN\$

BIN\$(number)

Returns a string giving the binary (base 2) value for the 'number'.

```
PRINT BIN$(77)  
1001101
```

```
PRINT BIN$(&h77)  
1110111
```

See also:

ASC	LEN	VAL
CHR\$	OCT\$	
DATE\$	STR\$	
HEX\$	TIME\$	

CHR\$

CHR\$(number)

Returns a one-character string consisting of the character corresponding to the ASCII code indicated by argument 'number'.

```
PRINT CHR$ (69)  
E
```

See also:

ASC	LEN	VAL
BIN\$	OCT\$	
DATE\$	STR\$	
HEX\$	TIME\$	

FORMAT\$

FORMAT\$(nbr [, fmt\$])

Will return a string representing 'nbr' formatted according to the specifications in the string 'fmt\$'.

The format specification starts with a % character and ends with a letter. Anything outside of this construct is copied to the output as is.

The structure of a format specification is:

% [flags] [width] [.precision] type

Where 'flags' can be:

- Left justify the value within a given field width
- 0 Use 0 for the pad character instead of space
- + Forces the + sign to be shown for positive numbers
- space Causes a positive value to display a space for the sign. Negative values still show the – sign

'width' is the minimum number of characters to output, less than this the number will be padded, more than this the width will be expanded. 'precision' specifies the number of fraction digits to generate with an e, or f type or the maximum number of significant digits to generate with a g type. If specified the precision must be preceded by a dot (.) .

'type' can be one of:

- g Automatically format the number for the best presentation.
- f Format the number with the decimal point and following digits
- e Format the number in exponential format

If uppercase G or F is used the exponential output will use an uppercase E. If the format specification is not specified "%g" is assumed.

```
PRINT FORMAT$ (45)
45

PRINT FORMAT$ (45, "%g")
45

PRINT FORMAT$ (24.1, "%g")
24.1

PRINT FORMAT$ (24.1, "%f")
24.10000
```

```
PRINT FORMAT$ (24.1, "%e")
2.410000e+01

PRINT FORMAT$ (24.1, "%09.3f")
00024.100

PRINT FORMAT$ (24.1, "%+.3f")
+24.100

PRINT FORMAT$ (24.1, "***%-9.3f***")
**24.100    **
```

See also:

INSTR	MID\$	STRING\$
LCASE	RIGHT\$	UCASE\$
LEFT\$	SPACE\$	
LEN	SPC	

HEX\$

HEX\$(number)

Returns a string giving the hexadecimal (base 16) value for the 'number'.

```
PRINT HEX$(90)  
5A
```

See also:

ASC

DATE\$

STR\$

BIN\$

LEN

TIME\$

CHR\$

OCT\$

VAL

INSTR

INSTR([start-position,] string-searched\$, string-pattern\$)

Returns the position at which string-pattern\$ occurs in string-searched\$, beginning at start-position.

```
big$ = "This is a test"  
  
PRINT INSTR(big$, "is")  
3  
  
PRINT INSTR(4, big$, "is")  
6
```

See also:

FORMAT\$	MID\$	STRING\$
LCASE	RIGHT\$	UCASE\$
LEFT\$	SPACE\$	
LEN	SPC	

LCASE\$

LCASE\$(string\$)

Returns 'string\$' converted to lowercase characters.

```
PRINT LCASE$("CAN YOU HEAR ME NOW?")
can you hear me now?
```

See also:

FORMAT\$

MID\$

STRING\$

INSTR

RIGHT\$

UCASE\$

LEFT\$

SPACE\$

LEN

SPC

LEFT\$

LEFT\$(string\$, number-of-chars)

Returns a substring of 'string\$' with 'number-of-chars' from the left (beginning) of the string.

```
PRINT LEFT$("Now is the time for all good men", 15)  
Now is the time
```

See also:

FORMAT\$	MID\$	STRING\$
INSTR	RIGHT\$	UCASE\$
LCASE	SPACE\$	
LEN	SPC	

LEN(string\$)

Returns the number of characters (length) in string\$.

```
PRINT LEN("What is the length of this string?")
34
```

See also:

ASC	DATE\$	STR\$
BIN\$	HEX\$	TIME\$
CHR\$	OCT\$	VAL

MID\$

MID\$(string\$, start-position-in-string[, number-of-chars])

Returns a substring of 'string\$' beginning at 'start-position-in-string' and continuing for 'number-of-chars' bytes. If 'number-of-chars' is omitted the returned string will extend to the end of 'string\$'

```
PRINT MID$("Now is the time for all good men", 8, 12)
the time for
```

See also:

FORMAT\$	LEN	STRING\$
INSTR	RIGHT\$	UCASE\$
LCASE	SPACE\$	
LEFT\$	SPC	

OCT\$

OCT\$(number)

Returns a string giving the octal (base 8) representation of 'number'.

```
PRINT OCT$(32)  
40
```

See also:

ASC	DATE\$	STR\$
BIN\$	HEX\$	TIME\$
CHR\$	LEN	VAL

RIGHT\$

RIGHT\$(string\$, number-of-chars)

Returns a substring of 'string\$' with 'number-of-chars' from the right (end) of the string.

```
PRINT RIGHT$("Now is the time for all good men", 12)
all good men
```

See also:

FORMAT\$	LEN	STRING\$
INSTR	MID\$	UCASE\$
LCASE	SPACE\$	
LEFT\$	SPC	

SPACE\$

SPACE\$(number)

Returns a string of blank spaces 'number' bytes long.

```
PRINT "*" + SPACE$(8) + "*"  
* * *
```

See also:

FORMAT\$	LEN	STRING\$
INSTR	MID\$	UCASE\$
LCASE	RIGHT\$	
LEFT\$	SPC	

SPC

SPC(number)

Returns a string of blank spaces 'number' bytes long. This function is similar to the SPACE\$() function and is only included for Microsoft compatibility.

```
PRINT "*" + SPC(10) + "*"
*           *
```

See also:

FORMAT\$	LEN	STRING\$
INSTR	MID\$	UCASE\$
LCASE	RIGHT\$	
LEFT\$	SPACE\$	

STR\$

STR\$(number)

Returns a string in the decimal (base 10) representation of 'number'.

```
this$ = STR$(123)  
  
PRINT this$  
123
```

See also:

ASC

DATE\$

OCT\$

BIN\$

HEX\$

TIME\$

CHR\$

LEN

VAL

STRING\$

STRING\$(number, ascii-value|string\$)

Returns a string 'number' bytes long consisting of either the first character of string\$ or the character representing the ASCII value ascii-value.

```
PRINT STRING$(5, "Hello")
HHHHH
```

```
PRINT STRING$(5, 100)
ddddd
```

See also:

FORMAT\$	LEN	SPC
INSTR	MID\$	UCASE\$
LCASE	RIGHT\$	
LEFT\$	SPACE\$	

UCASE\$

UCASE\$(string\$)

Returns 'string\$' converted to uppercase characters.

```
PRINT UCASE$ ("what?")
WHAT?
```

See also:

FORMAT\$	LEN	SPC
INSTR	MID\$	STRING\$
LCASE	RIGHT\$	
LEFT\$	SPACE\$	

VAL

VAL(string\$)

Returns the numerical value of the 'string\$'. If 'string\$' is an invalid number the function will return zero.

This function will recognize the &H prefix for a hexadecimal number, &O for octal and &B for binary.

```
PRINT VAL("123")
123

PRINT VAL("Hello")
0
```

See also:

ASC	DATE\$	OCT\$
BIN\$	HEX\$	STR\$
CHR\$	LEN	TIME\$

MMBasic File System

MMBasic supports two “drives” the internal flash memory (A:) and the very large drive that exists when you use an SD card (B:). File system commands allow you to load and save data from/to these drives, as well as manipulate directories/files on those drives.



Illustration 177: Using a micro SD card with the CGMMSTICK for program/data storage. This is the B: drive.

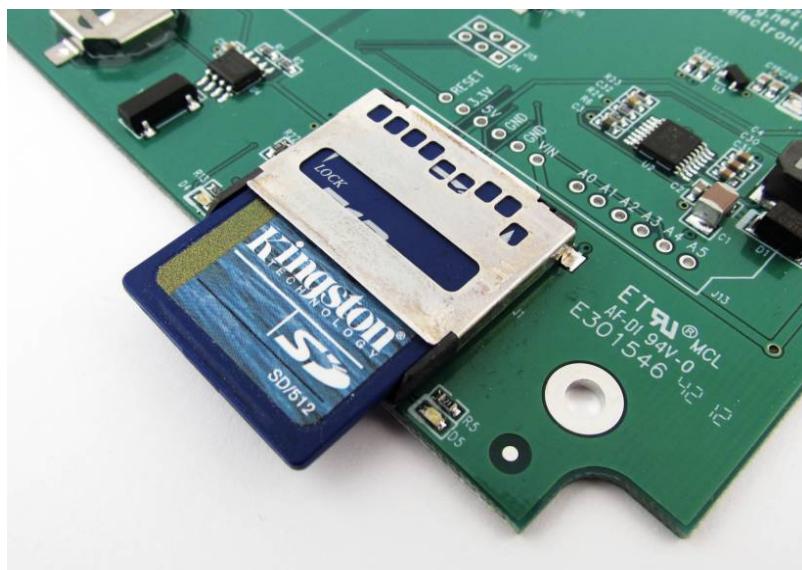


Illustration 178: Using an SD card with the CGCOLORMAX for program/data storage. This is the B: drive.

Storage Commands and Functions

Data files can be opened using OPEN and read from using INPUT, LINE INPUT, or INPUT\$() or written to using PRINT or WRITE. On the SD card data is stored using standard text and can be read and edited in Windows, Apple Mac, Linux, etc. An SD card can have up to 10 files open simultaneously while the internal flash drive has a maximum of one file open at a time.

You can list the programs stored on a drive with the FILES command, delete them using KILL and rename them using NAME. On an SD card the current working directory can be changed using CHDIR. A new directory can be created with MKDIR or an old one deleted with RMDIR.

Whenever specified a file name can be a string constant (ie, enclosed in double quotes) or a string variable. This means you must use double quotes if you are directly specifying a file name. Eg, RUN "TEST.BAS"

PROGRAM AND DATA STORAGE

On the CGMMSTICK and CGCOLORMAX two “drives” are available for storing and loading programs and data:

Drive “A:” is a virtual drive using the PIC32’s internal flash memory and has a size of about 190KB.

Drive “B:” is the SD card (if present). MMBasic supports MMC, SD or SDHC memory cards formatted as FAT16 or FAT32 with capacities up to the largest that you can purchase.

File names must be in 8.3 format prefixed with an optional drive prefix A: or B: (the same as DOS or Windows). Long file names and long directory names are not supported. The default drive on power up is B: and this can be changed with the DRIVE command.

On the Maximite MMBasic will look for a file on start up called “AUTORUN.BAS” first in the root directory of the internal flash drive (A:) then in the root directory of the SD card (B:). If the file is found it will be automatically loaded and run, otherwise MMBasic will print a prompt (">") and wait for input.

Note that the video output will go blank for a short time while writing data to the internal flash drive A:. This is normal and is caused by a requirement to shut off the video while reprogramming the memory.

When using drive A: you need to be careful not to wear out the flash (the same applies to SD cards). If drive A: is empty, you could write and delete a file on it every day for 175 years before you would reach the endurance limit - but if the interval was once a minute you would reach the limit in about 6 weeks.

CHDIR

CHDIR dir\$

Change the current working directory on the SD card to 'dir\$'

The special entry ".." represents the parent directory of the current directory and "." represents the current directory.

See also:

COPY	FILES	NAME
CWD\$	KILL	RMDIR
DIR\$	MKDIR	
DRIVE	MM.DRIVES\$	

CLOSE (file)

CLOSE [#]nbr [,[#]nbr] ...

Close the file(s) previously opened with the file number 'nbr'. The # is optional.

See also:

EOF

LINE INPUT

WRITE

INPUT#

OPEN

INPUT\$

PRINT

COPY

COPY src\$ TO dest\$

Copy the file named 'src\$' to another file named 'dest\$'.

'dest\$' can be just a drive designation (ie, A:) and this makes it convenient to copy files between drives.

Note that it is not possible to have both the source and destination on the internal flash drive A:.

See also:

CHDIR	FILES	NAME
CWD\$	KILL	RMDIR
DIR\$	MKDIR	
DRIVE	MM.DRIVES\$	

CWD\$

CWD\$

Returns the current working directory on the SD card as a string.

See also:

CHDIR	FILES	NAME
COPY	KILL	RMDIR
DIR\$	MKDIR	
DRIVE	MM.DRIVES\$	

DIR\$

DIR\$([fspec], [type])

Will search an SD card for files and return the names of entries found. 'fspec' is a file specification using wildcards the same as used by the FILES command. Eg, "*.*" will return all entries, "*.TXT" will return text files. 'type' is the type of entry to return and can be one of:

VOL Search for the volume label only

DIR Search for directories only

FILE Search for files only (the default if 'type' is not specified)

The function will return the first entry found. To retrieve subsequent entries use the function with no arguments. ie, DIR\$(). The return of an empty string indicates that there are no more entries to retrieve.

This function only operates on the SD card and you must change to the required directory before invoking it.

See also:

CHDIR	FILES	NAME
COPY	KILL	RMDIR
CWD\$	MKDIR	
DRIVE	MM.DRIVE\$	

DRIVE

DRIVE drivespec\$

Change the default drive used for file operations that do not specify a drive to that specified in drivespec\$. This can be the string "A:" or "B:".

See also the predefined read only variable MM.DRIVE\$.

See also:

CHDIR	FILES	NAME
COPY	KILL	RMDIR
CWD\$	MKDIR	
DIR\$	MM.DRIVE\$	

EOF

EOF([#]nbr)

Will return true if the file previously opened for INPUT with the file number 'nbr' is positioned at the end of the file.

If used on a file number opened as a serial port this function will return true if there are no characters waiting in the receive buffer. The # is optional.

See also:

CLOSE	LINE INPUT	WRITE
INPUT#	OPEN	
INPUT\$	PRINT	

FILES

FILES [search_pattern\$]

Lists files in the current directory on the SD card.

The SD card (drive B:) may use an optional 'fspec \$'. Question marks (?) will match any character and an asterisk (*) will match any number of characters. If omitted, all files will be listed. For example:

. Find all entries

*.TXT Find all entries with an extension of TXT

E*.* Find all entries starting with E

X?X.* Find all three letter file names starting and ending with X

See also:

CHDIR	DRIVE	NAME
COPY	KILL	RMDIR
CWD\$	MKDIR	
DIR\$	MM.DRIVE\$	

INPUT #

INPUT #nbr, list of variables

Allows input from from a file previously opened for INPUT as 'nbr' to a list of variables.

The input must contain commas to separate each data item if there is more than one variable.

For example, if the command is:

```
INPUT #1, a, b, c
```

And the following is in the file: 23, 87, 66

Then a = 23 and b = 87 and c = 66

See also:

CLOSE	LINE INPUT	WRITE
EOF	OPEN	
INPUT\$	PRINT	

INPUT\$ (from file)

INPUT\$(nbr, [#]fnbr)

Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number 'fnbr'. This function will read all characters including carriage return and new line without translation.

See also:

CLOSE

LINE INPUT

WRITE

EOF

OPEN

INPUT#

PRINT

KILL

KILL files\$

Deletes the file specified by file\$ from the SD card.

Quote marks are required around a string constant and the extension, if there is one, must be specified. Example: KILL "SAMPLE.DAT"

See also:

CHDIR	DRIVE	NAME
COPY	FILES	RMDIR
CWD\$	MKDIR	
DIR\$	MM.DRIVES\$	

LINE INPUT (file)

LINE INPUT #nbr, string-variable\$

Reads entire line from the file previously opened for INPUT as 'nbr'. Unlike INPUT, LINE INPUT will read a whole line, not stopping for comma delimited data items.

See also:

CLOSE

INPUT\$

WRITE

EOF

OPEN

INPUT#

PRINT

MKDIR

MKDIR dir\$

Make, or create, the directory 'dir\$' on the SD card.

See also:

CHDIR	DRIVE	NAME
COPY	FILES	RMDIR
CWD\$	KILL	
DIR\$	MM.DRIVES\$	

MM.DRIVE\$

MM.DRIVE\$

The current default drive returned as a string containing either "A:" or "B:".

See also:

CHDIR	DRIVE	NAME
COPY	FILES	RMDIR
CWD\$	KILL	
DIR\$	MKDIR	

MM.ERRNO

MM.ERRNO

Is set to the error number if a statement involving the SD card fails or zero if the operation succeeds. This is dependent on the setting of OPTION ERROR. The possible values for MM.ERRNO are:

- 0 = No error
- 1 = No SD card found
- 2 = SD card is write protected
- 3 = Not enough space
- 4 = All root directory entries are taken
- 5 = Invalid filename
- 6 = Cannot find file
- 7 = Cannot find directory
- 8 = File is read only
- 9 = Cannot open file
- 10 = Error reading from file
- 11 = Error writing to file
- 12 = Not a file
- 13 = Not a directory
- 14 = Directory not empty
- 15 = Hardware error accessing the storage media
- 16 = Flash memory write failure

See also:

CHDIR	FILES	OPEN
CLOSE	INPUT#	OPTION ERROR
COPY	INPUT\$	PRINT
CWD\$	KILL	RMDIR
DIR\$	LINE INPUT	WRITE
DRIVE	MKDIR	
EOF	NAME	

NAME

NAME old\$ AS new\$

Rename a file or a directory on the SD card from 'old\$' to 'new\$'

Unlike the other commands that work with file names the NAME command cannot accept a full path name (with directories).

See also:

CHDIR

DRIVE

MM.DRIVE\$

COPY

FILES

RMDIR

CWD\$

KILL

DIR\$

MKDIR

OPEN (file)

OPEN fname\$ FOR mode AS [#]fnbr

Opens a file for reading or writing.

'fname' is the file name (8 chars max) with an optional extension (3 chars max) separated by a dot (.). It can be prefixed with a directory path.

For example:

```
OPEN "B:\DIR1\DIR2\FILE.EXT" FOR INPUT AS #1
```

'mode' is INPUT or OUTPUT or APPEND.

INPUT will open the file for reading and throw an error if the file does not exist.

OUTPUT will open the file for writing and will automatically overwrite any existing file with the same name.

APPEND will also open the file for writing but it will not overwrite an existing file; instead any writes will be appended to the end of the file. If there is no existing file the APPEND mode will act the same as the OUTPUT mode (i.e. the file is created then opened for writing).

Note: APPEND is not supported on the flash file system (drive A:).

'fnbr' is the file number (1 to 10). The # is optional. Up to 10 files can be open simultaneously. The INPUT, LINE INPUT, PRINT, WRITE and CLOSE commands as well as the EOF() and INPUT\$() functions all use 'fnbr' to identify the file being operated on.

See also OPTION ERROR and MM.ERRNO for error handling.

The length of the path plus the file name (including punctuation) must be less than 127 characters.

The following code will try to open a file. If it is not there, an error occurs.

```
' Stop error from halting program
OPTION ERROR CONTINUE

FileStatus = 1

OPEN "test.txt" FOR INPUT AS #1

' Check error condition
```

```
IF MM.ERRNO = 6 THEN FileStatus = 0  
  
OPTION ERROR ABORT  
  
' If file opened without error...  
IF FileStatus = 1 THEN  
  
    ' Process opened file here  
  
ENDIF
```

See also:

CLOSE

INPUT\$

WRITE

EOF

LINE INPUT

INPUT#

PRINT

OPTION ERROR

OPTION ERROR ABORT | CONTINUE

Sets the treatment for errors in file input/output. The option CONTINUE will cause MMBasic to ignore file related errors. The program must check the variable MM.ERRNO to determine if and what error has occurred.

The option ABORT sets the normal behavior (ie, stop the program and print an error message). The default is ABORT.

Note that this option only relates to errors reading or writing from the SD card, it does not affect the handling of syntax and other program errors.

See also:

CHDIR	FILES	NAME
CLOSE	INPUT#	OPEN
COPY	INPUT\$	PRINT
CWD\$	KILL	RMDIR
DIR\$	LINE INPUT	WRITE
DRIVE	MKDIR	
EOF	MM.ERRNO	

PRINT

PRINT #nbr, expression [[,;]expression] ... etc

Same as PRINT except that the output is directed to a file previously opened for OUTPUT or APPEND as 'nbr'.

See also:

CLOSE

INPUT\$

WRITE

EOF

LINE INPUT

INPUT#

OPEN

RMDIR

RMDIR dir\$

Remove, or delete, the directory 'dir\$' on the SD card.

See also:

CHDIR	DRIVE	MM.DRIVE\$
COPY	FILES	NAME
CWD\$	KILL	
DIR\$	MKDIR	

WRITE

WRITE [#nbr,] expression [,expression] ...

Outputs the value of each 'expression' separated by commas (,). If the 'expression' is a number it is outputted without preceding or trailing spaces.

If it is a string it is surrounded by double quotes (""). The list is terminated with a new line.

If '#nbr' is specified the output will be directed to a file on the SD card previously opened for OUTPUT or APPEND as '#nbr'.

WRITE can be replaced by the PRINT command.

See also:

CLOSE	INPUT\$	PRINT
EOF	LINE INPUT	
INPUT#	OPEN	

XMODEM

XMODEM SEND file\$

XMODEM RECEIVE file\$

Transfers a file to or from a remote computer using the XModem protocol. The transfer is done over the USB connection or, if a serial port is opened as console, over that serial port. 'file\$' is the file (on the SD card or internal flash) to be sent or received. The XModem protocol requires a cooperating software program running on the remote computer and connected to its serial port.

The transfer can take up to 15 seconds to start and if the XMODEM command fails to establish communications it will return to the MMBasic prompt after 60 seconds.

See also:

CHDIR	DRIVE	MM.DRIVE\$
COPY	FILES	NAME
CWD\$	KILL	
DIR\$	MKDIR	

MMBasic input and output

The CGMMSTICK and CGCOLORMAX are all about interfacing to hardware. MMBasic supports simple control of the data lines as well as asynchronous serial communication, SPI, I2C, PWM, analog input, and other useful interface functions.

INPUT AND OUTPUT

You can configure an external I/O pin using the SETPIN command, set its output using the PIN()= command and read the current input value using the PIN() function. Digital I/O uses the number zero to represent a low voltage and any non-zero number for a high voltage. An analogue input will report the measured voltage as a floating point number.

The CGMMSTICK Maximite has 20 I/O pins numbered 1 to 20. (See J1 Pinout) Pins 1 to 10 can be used for analog input and digital input/output with a maximum input voltage of 3.3V. Pins 11 to 20 are digital only but support input voltages up to 5V and can be set to open collector.

Normally digital output is 0V (low) to 3.3V (high) but you can use open collector to drive 5V circuit. This means that the pin can be pulled down (when the output is low) but will go high impedance when the output is high. With a pull up resistor to 5V an output configured as open collector you can drive 5V logic signals. Typical value of the pull up resistor is 1K to 4.7K.

CGCOLORMAX "ARDUINO SHIELD" CONNECTOR

In addition to the 20 I/O pins described above the CGCOLORMAX Maximite has an extra 20 I/O pins on the Arduino compatible connector (40 I/O pins in total). These are labeled D0 to D13 and A0 to A5.

You can use the labels D0, D1, etc in the SETPIN and PIN statements or you can use their corresponding numbers (D0 = 21, D1 = 22, etc and A0 = 35, A1 = 36, etc). The digital pins (D0 to D13) have the same characteristics (5V, open collector, etc) as the digital pins 11 to 20 and the analog capable pins (A0 to A5) have the same capabilities as pins 1 to 10.

Communications

Two serial ports are supported with speeds up to 19200 baud with configurable buffer sizes and optional hardware flow control. The serial ports are opened using the OPEN command and any command or function that uses a file number can be used to send and receive data.

Communications to slave or master devices on an I2C bus is supported with eight commands MMBasic fully supports bus master and slave mode, 10 bit addressing, address masking and general call, as well as bus arbitration (ie, bus collisions in a multi-master environment).

The Serial Peripheral Interface (SPI) communications protocol is supported with the SPI command.

The 1-Wire protocol is also supported.

Interrupts

Any external I/O pin can be configured to generate an interrupt using the SETPIN command with up to 29 interrupts (including the tick interrupt) active at any one time. Interrupts can be set up to occur on a rising or falling digital input signal and will cause an immediate branch to a specified line number or label (similar to a GOSUB). The target can be the same or different for each interrupt. Return from an interrupt is via the IRETURN statement. All statements (including GOSUB/RETURN) can be used within an interrupt.

If two or more interrupts occur at the same time they will be processed in order of pin numbers (ie, an interrupt on pin 1 will have the highest priority). During processing of an interrupt all other interrupts are disabled until the interrupt routine returns with an IRETURN. During an interrupt (and at all times) the value of the interrupt pin can be accessed using the PIN() function.

Interrupts can occur at any time but they are disabled during INPUT statements. If you need to get input from the keyboard while still accepting interrupts you should use the INKEY\$ function. When using interrupts the main program is completely unaffected by the interrupt activity unless a variable used by the main program is changed during the interrupt.

For most programs MMBasic will respond to an interrupt in under 100µS. To prevent slowing the main program by too much an interrupt should be short and execute the IRETURN statement as soon as possible.

Also remember to disable an interrupt when you have finished needing it – background interrupts can cause strange and non-intuitive bugs.

CAN (Controller Area Network)

These commands provide access to the CAN hardware available on the CGCOLORMAX2.

A CAN module supports up to 32 channels, each channel can be configured to transmit or receive data. For receiving data the CAN module writes received messages into a different FIFO buffer for each channel. One entry in a FIFO buffer is 16 bytes long (4 address bytes, 4 length bytes, 8 data bytes).

These commands allow the user to specify the number of records in each FIFO buffer and the internals keep track of the memory requirements and automatically allocates memory from the available heap. A command is provided to free the allocated memory when the CAN interface is no longer required.

Note that CAN channels are numbered from 0 to 31 and that, per the Microchip documentation, CAN channels should be configured contiguously from 0 upwards. I.e. if 5 channels are required use channels 0, 1, 2, 3 & 4; and not 1, 2, 3, 4 & 5 or 3, 5, 10, 22, 31 or any other combination.

The CAN module is first configured by placing it in configuration mode and then issuing configuration commands. When the desired configuration is set up then the CAN module is enabled. Once enabled the RX and TX commands can be used to receive and send data on the bus. Once the CAN module is finished it should be disabled and the memory associated with the FIFO buffers should be freed (for example at the end of the application)

An RX channel can either have a filter for a specific id or be configured to receive all messages. In the latter case the user can request the id of the received command and then perform processing based on the id of the command. However, on a CAN bus with moderate to high utilization, more reliable operation will be achieved by setting up multiple channels with each channel filtering for one id of interest. This is because of the relatively small buffer sizes and the relatively slow operation of the BASIC interpreter – when trying to receive all messages into one buffer the buffer overruns and data is lost. Furthermore, when a channel is being filtered for a single ID it may be appropriate to set the FIFO buffer size to just 1 record, especially if the goal is to update a single value display. This will always give you the most up-to-date data.

The examples provided were designed for (and tested on) a second generation Toyota Prius (Model years 2004 – 2009).

Throughout this overview “CAN commands” have been referred to in the plural. In fact there is only one true CAN command – the other commands are “sub-commands” from this one master. By structuring the commands this way there is less impact on the limitation for the number of top-level/master commands that MMBasic can support.

EXAMPLE ONE – Minimal example of reading one channel

```

' (c) John Harding, 2012

' Example designed for Gen 2 Prius

' Configures connection speed to 500kbps and
' monitors for CAN id 52Ch when data is received
' we calculate the ECT from the appropriate data
' bytes.

' Note that the period of this message is
' approximately 1Hz

Cls
Dim ok
Dim data(8)
Dim ect

CAN CONFIG ok
CAN SETSPEED 500000, ok
CAN ADDRXCHNL 0,&h52C,0,1,ok
CAN ENABLE ok

Timer = 0
Do
    If (Inkey$ = "q") Then Exit
    CAN RX 0,data(0),ok
    If (ok=1) Then
        Print Timer ": ECT = " (data(1)/2)
    EndIf
Loop

CAN FREE

End

```

EXAMPLE TWO – Example of reading all channels and displaying just one (but don't do this!)

```

' (c) John Harding, 2012

```

```

' Example designed for Gen 2 Prius

' Configures connection speed to 500kbps and
' configures a channel to receive all messages
' into a FIFO buffer with 32 records.
'

' When a message with id 52Ch is received
' we calculate the ECT from the appropriate data
' bytes.

' Note that the period of this message is
' approximately 1Hz but that we receive many
' wrong ids before we get the message we want

' This example is provided to contrast with
' example 1. It is suggested to use example 1
' as the basis for your code.

Dim ok
Dim data(8)
Dim id
Dim typ
Dim length

CAN CONFIG ok
CAN SETSPEED 500000, ok
CAN ADDRXCNL 0, 0, 0, 32, ok
CAN ENABLE ok
Timer=0
Do
    q$ = Inkey$
    If (q$="q") Then Exit
    CAN RX 0, id, typ, length, data(0), ok
    If (ok=1) Then
        If (id=&H52C) Then
            Print ""
            Print Timer ": " Hex$(id) " : " length " : ECT= " data(1) / 2 " C"
        Else
            Print Timer ": " Hex$(id) " ";
        Endif
    EndIf
Loop

```

CAN FREE

End

EXAMPLE THREE – Reading manufacturer specific PID's on a Toyota Prius 2005.

```
' Example to query PIDs for the battery ECU
' and convert the received data into engineering values
' JDH 10/8/12

Dim ok
Dim txID : Dim txData(8) : Dim txLen
Dim rxID : Dim rxData(8)

txData(0) = 0 : txData(1) = 0 : txData(2) = 0 : txData(3) = 0
txData(4) = 0 : txData(5) = 0 : txData(6) = 0 : txData(7) = 0
txLen = 8 ' always transmit 8 bytes even though payload may be less

txID = &H7E3      ' Battery ECU module
rxID = txID + 8 ' id of reply is 8 higher than module number

' Check the "ok" result, if it fails print a message and exit
Sub checkOK(okay, failed$, succeeded$, xit)
    If (okay=0) Then
        Print failed$
        If (xit=1) Then Exit
    Else
        Print succeeded$
    EndIf
End Sub

' print formatted hex numbers
Function toHex$(val)
    toHex$=""
    If (val<16) Then toHex$="0"
    toHex$ = toHex$ + Hex$(val)
End Function

' print timer : id : len : data
Sub PrintRawData
    Print Timer ":" Hex$(rxId) " :" rxLen " :" toHex$(rxData(0)) " ";
    Print toHex$(rxData(1)) " " toHex$(rxData(2)) " " toHex$(rxData(3)) " ";

```

```

        Print toHex$(rxData(4)) " " toHex$(rxData(5)) " " toHex$(rxData(6)) " ";
        Print toHex$(rxData(7))
End Sub

' make the PID request
Sub SendModeAndPID(mode, pid)
    Local txOK
    txData(0) = 3
    txData(1) = mode
    txData(2) = pid
    CAN TX 0,txID,0,txLen,txData(0),txOk
    checkOK(txOK, "FAILED TO SEND PID", "SENT PID " + Hex$(pid) + " TO " +
    Hex$(txId), 0)
End Sub

' Send the acknowledgement after receiving frame
Sub SendReadyForMore()
    Local txOK
    txData(0) = &H30
    txData(1) = 0
    txData(2) = 0
    CAN TX 0,txID,0,txLen,txData(0),txOk
    checkOK(txOK, "FAILED TO SEND PID", "SENT 0x30 TO " + Hex$(txId), 0)
End Sub

' Chains together the initial PID request and the acknowledgement
Sub RequestPID(mode, pid)
    ' Note, we're "cheating" here and simply waiting 10msec and sending the
    ' acknowledgement. Strictly speaking we should wait for the response
    ' header first (frame 0x10). This works because in the CAN setup (below)
    ' we've setup a big enough buffer to receive all the frames from the
    ' longest PID.
    SendModeAndPID(mode, pid)
    Pause 10
    SendReadyForMore
End Sub

Sub DisplayD0
    ' retrieve the data (see DisplayCE for commentary)
    RequestPID &H21,&HD0
    Do
        CAN RX 1, rxData(0), ok
    Loop Until (ok=1)
    PrintRawData
    Local b(rxData(1)+15)
    Local j
    Local i

```

```

j=0
For i=4 To 7
    b(j)=rxData(i)
    j=j+1
Next i
Do
    CAN RX 1, rxData(0), ok
    If (ok=1) Then
        PrintRawData
        For i=1 To 7
            b(j)=rxData(i)
            j=j+1
        Next i
    EndIf
Loop Until (ok=0)

' Convert to engineering units (see DisplayCE for commentary)
Print "Block Count      = " b(0)
Print "Time in LOW      = " (256*b(1)+b(2))
Print "Time in DC Inhibit = " (256*(b3)+b(4))
Print "Time in HIGH      = " (256*(b5)+b(6))
Print "Time in HOT       = " (256*(b7)+b(8))
Print "Lowest Block      = " b(9)
Print "Lowest Voltage    = " (2.56*b(10)+0.1*b(11)-327.68)
Print "Highest Block     = " b(12)
Print "Highest Voltage   = " (2.56*b(13)+0.1*b(14)-327.68)
For i=15 To 28
    Print "Block " toHex$(i-14) " Resistance = " (0.001*b(i)) " Ohms"
Next i
End Sub

Sub DisplayCF
    ' retrieve the data (see DisplayCE for commentary)
    RequestPID &H21,&HCF
    Do
        CAN RX 1, rxData(0), ok
    Loop Until (ok=1)
    PrintRawData
    Local b(rxData(1)+15)
    Local j
    Local i
    j=0
    For i=4 To 7
        b(j)=rxData(i)
        j=j+1
    Next i

```

```

Do
    CAN RX 1, rxData(0), ok
    If (ok=1) Then
        PrintRawData
        For i=1 To 7
            b(j)=rxData(i)
            j=j+1
        Next i
    EndIf
Loop Until (ok=0)

' Convert to engineering units (see DisplayCE for commentary)
Print "Air Intake Temp = " (4.608*b(0)+0.018*b(1)-557.824) " F"
Print "Fan Motor Voltage = " (0.2*b(2) - 25.6) " V"
Print "Aux Batt Voltage = " (0.2*b(3) - 25.6) " V"
Print "Battery Charge = " (b(4) - 64)
Print "Battery Discharge = " (b(5) - 64)
Print "Delta SOC = " (0.01 * b(6)) " %"
Print "Fan Speed = " b(8)
Print "Batt Temp 1 = " (4.608*b(10)+0.018*b(11)-557.824) " F"
Print "Batt Temp 2 = " (4.608*b(12)+0.018*b(13)-557.824) " F"
Print "Batt Temp 3 = " (4.608*b(14)+0.018*b(15)-557.824) " F"
End Sub

Sub DisplayCE
    ' send the request
    RequestPID &H21,&HCE

    ' retrieve the first frame
    Do
        CAN RX 1, rxData(0), ok
    Loop Until (ok=1)
    PrintRawData
    ' create an array big enough to hold all the data
    ' the +15 is because there always seems to be 1 more frame than expected
    ' and if the amount of data is 1 larger than a multiple of 8 we need 7
    ' bytes for the remainder of that frame + 8 bytes for the "extra" frame
    Local buffer(rxData(1)+15)

    ' Copy the data from the first frame into the buffer
    Local i
    Local j
    j=0
    For i=4 To 7
        buffer(j)=rxData(i)
        j=j+1

```

```

Next i

' now process the remaining frames
Do
    CAN RX 1, rxData(0), ok
    If (ok=1) Then
        PrintRawData
        For i=1 To 7
            buffer(j)=rxData(i)
            j=j+1
        Next i      EndIf
    Loop Until (ok=0)

' We have the raw data, now convert it to engineering values
' These calculations are from USBSeaWolf's spreadsheet available on the
' PriusChat.com forum.
Print "SOC      = " (0.5 * buffer(0)) "%"
Print "Current  = " (2.56*buffer(1)+0.1*buffer(2)-327.68) " A"
j=3
For i=1 To 14
    Print "Block " toHex$(i) " = " (2.56*buffer(j)+0.1*buffer(j+1)-327.68) " V"
    j=j+2
Next i
End Sub

' ****
' START HERE...

Cls

CAN CONFIG ok           : checkOK(ok, "CAN CONFIG FAILED", "", 1)
CAN SETSPEED 500000, ok : checkOK(ok, "CAN SETSPEED FAILED", "", 1)
CAN ADDTXCHNL 0,1,ok     : checkOK(ok, "CAN ADDTXCHNL FAILED", "", 1)
CAN ADDRXCHNL 1,rxId,0,10,ok : checkOK(ok, "CAN ADDRXCHNL FAILED", "", 1)
CAN ENABLE ok           : checkOK(ok, "CAN ENABLE FAILED", "", 1)
CAN PRINTCONFIG

' Note the above configuration for channel 1
' (a) we're filtering on the response ID (which is the moduleID + 8)
' (b) we create a 10 record buffer this allows us to capture all the
response
'     frames in one go

Print "q:quit, t:0xCE, u:0xCF, v:0xD0 send pid to " Hex$(txID)

Timer = 0

```

```
Do
k$ = Inkey$
If (k$ = "q") Then Exit
If (k$ = "t") Then DisplayCE
If (k$ = "u") Then DisplayCF
If (k$ = "v") Then DisplayD0
CAN RX 1, rxData(0), ok
If (ok=1) Then PrintRawData
Loop

CAN FREE
```

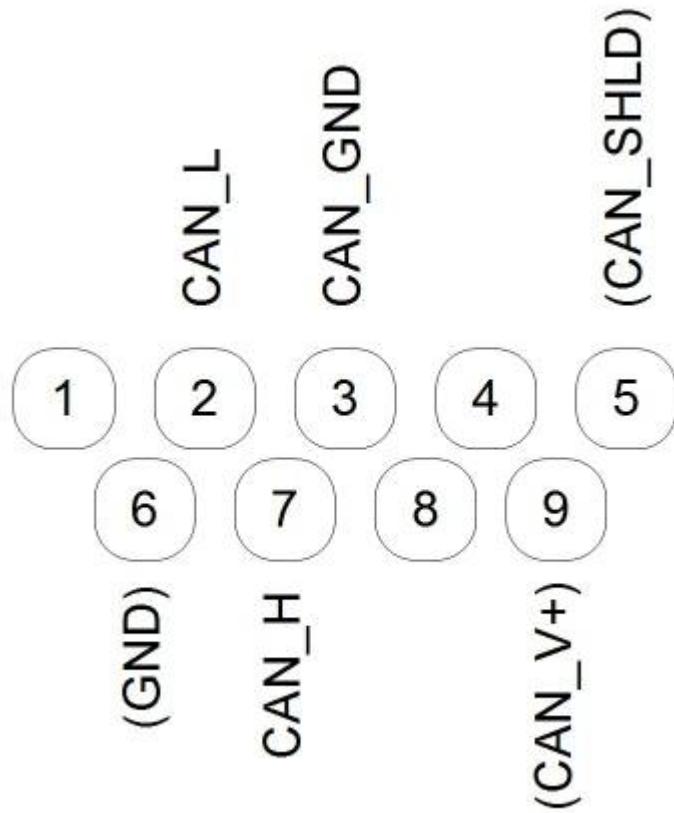


Illustration 179: CAN-in-Automation connector pin out.

There are several types of connectors used with CAN. Illustrated here is a standard using a 9-pin D-sub connector. This follows the CiA DS-102 pin out.

Pin	Signal	Description
1		
2	CAN_L	CAN_L bus line
3	CAN_GND	CAN ground
4		
5	(CAN_SHLD)	CAN shield (optional)
6	(GND)	Ground (optional)
7	CAN_H	CAN_H bus line
8		
9	(CAN_V+)	CAN positive supply (optional)

Audio and PWM Output

MMBasic there are a number of ways that you can use the sound output. You can play synthesized music, generate tones or generate program controlled voltages (PWM).

PLAYMOD

This command will play synthesized music in the background while the program is running. The music must be in the MOD format and the file containing the music must be located on the internal flash drive (drive A:).

The audio is high quality and MMBasic will generate full stereo on the CGCOLORMAX Maximite.

The MOD format is a music file format originating from the MOD file format on Amiga systems in the late 1980s. It is not a recording of the music (like a MP3 file) - instead it contains instructions for synthesizing the music. On the original Amiga this task was performed by dedicated hardware.

MMBasic will read this file and continuously play the music in the background while the program that launched the music will continue running in the foreground. Be aware that synthesizing music is a CPU intensive activity and uses a lot of memory and this could affect the performance of the program.

A description of the MOD format can be found at:

[http://en.wikipedia.org/wiki/MOD_\(file_format\)](http://en.wikipedia.org/wiki/MOD_(file_format))

A large selection of files that can be played on the Maximite can be found at:

<http://modarchive.org> (look for files with the .MOD extension). Because the file must be located on drive A: to play it would be wise to select reasonably small files.

You can also create your own music using a tracker. For an example see:

<http://www.modplug.com>

TONE

This command will create two tones for the CGCOLORMAX Maximite that will be outputted separately on the left and right sound channels. On the monochrome Maximite only one tone is generated. The tone is a synthesized sine wave and can be in the range of 1Hz to 20KHz with a resolution of 1Hz and is very accurate as it is locked to the PIC32's crystal oscillator. When the frequency is changed there is no interruption in the output so the output can be made to glide smoothly across a range of frequencies.

The playing time can be specified in milliseconds and the tone will play in the background (ie, the program continues running).

SOUND

The sound command is included only for compatibility with older programs. It generates a single frequency square wave and should be replaced with the tone or PWM command in new programs.

PWM

The PWM (Pulse Width Modulation) command allows the Maximite to generate two square waves with programmed controlled duty cycle. By varying the duty cycle you can generate a program controlled voltage outputs for use in controlling external devices that require an analog input (power supplies, motor controllers, etc). The CGCOLORMAX Maximite has two channels while the monochrome CGMMSTICK Maximite has a single channel.

The frequency for both channels is the same and can be from 20Hz to 1MHz. The duty cycle for each channel can be independently set from between 0% and 100% with a 0.1% resolution.

This command uses the sound output for generating the PWM signal so the components on this output may need to be changed to allow this output to work as a PWM output.

BYTE2NUM

BYTE2NUM(array(x))

BYTE2NUM(arg1, arg2, arg3, arg4)

Return the number created by storing the four arguments as consecutive bytes (MMBasic numbers are stored as the C float type and are four bytes in length).

The bytes can be supplied as four separate numbers (arg1 - arg4) or as four elements of 'array' starting at index 'x'.

See the command NUM2BYTE for the reverse of this function.

See also:

I2CDIS

I2CSDIS

MM.I2C

I2CEN

I2CSEN

NUM2BYTE

I2CRCV

I2CSRCV

I2CSEND

I2CSSEND

CAN

CAN

Displays a list of the available commands.

CAN PRINTCONFIG

Displays details of the current configuration. The details displayed are module status (online or offline), speed and configured channels.

CAN CONFIG ok

ok (output) – 1 if successful, 0 otherwise

Clears any pre-existing configuration and puts the CAN module into configuration mode.

CAN SETSPEED speed, ok

speed (input) – baud rate in kbps from 10,000 to 1,000,000

ok (output) – 1 if successful, 0 otherwise

Sets the baud rate of the CAN connection. Value provided is bits per second with a minimum of 10kbps and a maximum of 1Mbps.

CAN ADDRXCHNL channel_num, can_id, msg_type, buffer_size, ok

channel_num (input) – a CAN channel from 0 to 31

can_id (input) – a CAN id to filter for (set to 0 to receive all CAN messages)

msg_type(input) – 0 for standard 11-bit IDs, 1 for extended 29-bit IDs

buffer_size(input) – size of the FIFO buffer for this channel expressed as number of records

ok (output) – 1 if successful, 0 otherwise

Configures the specified channel as a receive channel. To filter for an individual id provide the CAN id of interest, to receive all messages pass in a zero id. If you're monitoring a single id (with this channel) and want to act on the latest data set the buffer size to 1. Larger buffer sizes can be set to capture more data – note that no indication is given when buffer overrun occurs (the oldest data is simply discarded).

CAN ADDTXCHNL channel_num, buffer_size, ok

channel_num (input) – a CAN channel from 0 to 31

buffer_size(input) – size of the FIFO buffer for this channel expressed as number of records

ok (output) – 1 if successful, 0 otherwise

Configures the specified channel as a transmit channel. Normally a buffer size of 1 is sufficient. However, larger sizes allow you to separate the construction and buffering of transmissions from the actual transmission.

CAN ENABLE ok

ok (output) – 1 if successful, 0 otherwise

Description: Once the configuration is complete call this command to put the CAN module into normal operating mode and ready to receive or transmit data.

CAN RX channel_num, can_id, msg_type, length, data(8), ok

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

data(8) (output) – an array to receive the data from the FIFO record

ok (output) – 1 if successful, 0 if no data available or other failure occurs

This command is intended to read the data only from a channel that has been previously configured to monitor for a given ID (hence the id is already known and doesn't need to be retrieved from the buffer).

Note: The description of the RX and TX commands specify the required size of the data array. However, when calling these functions you will pass in a reference to the first item of the array. Assuming you are using zero-based arrays and an array called data then when you call the function this will be data(0). See the examples for clarification

CAN RX channel_num, data(8), ok

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

can_id (output) – the CAN id of the message read from the FIFO buffer

msg_type (output) – the message type of the message read from the FIFO buffer

length (output) – the amount of data read (between 0 and 8 bytes)

data(8) (output) – an array to receive the data from the FIFO record

ok (output) – 1 if successful, 0 if no data available or other failure occurs

Description: This command is intended to read information from the FIFO buffer of a channel that has been previously configured to receive all messages – hence the need to provide variables to retrieve the full information about the message.

Note: The description of the RX and TX commands specify the required size of the data array. However, when calling these functions you will pass in a reference to the first item of the array. Assuming you are using zero-based arrays and an array called data then when you call the function this will be data(0). See the examples for clarification

CAN TX channel_num, can_id, msg_type, length, data(8), ok

channel_num (input) – a CAN channel from 0 to 31 for a channel previously configured for RX

can_id (input) – the CAN id to send

msg_type (input) – the message type being sent (0=SID, 1=EID)

length (input) – the amount of data being sent (between 0 and 8 bytes)

data(8) (input) – an array of data bytes to send (values between 0 and 255)

ok (output) – 1 if successful, 0 otherwise

Places data into the FIFO buffer for this channel. Data will be sent on the bus when the CAN module detects the bus is available (i.e. not busy).

Note: The description of the RX and TX commands specify the required size of the data array. However, when calling these functions you will pass in a reference to the first item of the array. Assuming you are using zero-based arrays and an array called data then when you call the function this will be data(0). See the examples for clarification

CAN DISABLE ok

ok (output) – 1 if successful, 0 otherwise

Description: Puts the module into offline mode, but does not destroy the configuration. This can be used to stop all processing of CAN messages while another processor intensive task takes place. CAN ENABLE can then be called to re-enable the pre-existing configuration.

CAN FREE

Takes the module off line and frees all memory associated with the existing configuration.

CLOSE (serial)

CLOSE [#]nbr [,[#]nbr] ...

Close the serial port(s) previously opened with the file number 'nbr'. The # is optional. Also see the OPEN command.

See also:

CLOSE CONSOLE

LOC

OPEN

INPUT\$

LOF

OPEN AS COLSOLE

CLOSE CONSOLE (serial)

CLOSE CONSOLE

Will close a serial port that had been previously opened as the console.

See also:

CLOSE

LOF

SETPIN

INPUT\$

OPEN

LOC

OPEN AS COLSOLE

I2CDIS

I2CDIS

Disables the slave I2C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN. It will also send a stop if the bus is still held.

See also:

BYTE2NUM	I2CSDIS	MM.I2C
I2CEN	I2CSEN	NUM2BYTE
I2CRCV	I2CSRCV	SETPIN
I2CSEND	I2CSSEND	

I2CEN

I2CEN speed, timeout [, int]

Enables the I2C module in master mode.

'speed' is a value between 10 and 400 (for bus speeds 10kHz to 400kHz).

'timeout' is a value in milliseconds after which the master send and receive commands will be interrupted if they have not completed. The minimum value is 100. A value of zero will disable the timeout (though this is not recommended).

'int' is optional. It specifies the line number of an interrupt routine to be run when the send or receive command completes. If this is not supplied, the send and receive command will only return when they have completed or timed out. If it is supplied then the send and receive will complete immediately and the command will execute in the background.

See also:

BYTE2NUM	I2CSDIS	MM.I2C
I2CDIS	I2CSEN	NUM2BYTE
I2CRCV	I2CSRCV	SETPIN
I2CSEND	I2CSSEND	

I2CRCV

I2CRCV **addr, bus_hold, rcvlen, rcvbuf [,sendlen, senddata [,senddata]]**

Receive data from the I2C slave device with the optional ability to send some data first.

'addr' is the slave i2c address (note that 10 bit addressing is not supported).

'option' is a number between 0 and 3

1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command)

2 = treat the address as a 10 bit address

3 = combine 1 and 2 (hold the bus and use 10 bit addresses).

'rcvlen' is the number of bytes to receive.

'rcvbuf' is the variable to receive the data - this is a one dimensional array or if rcvlen is 1 then this may be a normal variable. The array subscript does not have to be zero and will be honoured, also bounds checking is performed.

Optionally you can specify data to be sent first using 'sendlen' and 'senddata'. These parameters are used the same as in the I2CSEND command (ie, senddata can be a constant, an array or a string variable).

Examples:

```
I2CRCV &h6f, 1, 1, avar
```

```
I2CRCV &h6f, 1, 5, anarray(0)
```

```
I2CRCV &h6f, 1, 4, anarray(2), 3, &h12, &h34, &h89
```

```
I2CRCV &h6f, 1, 3, anarray(0), 4, anotherarray(0)
```

The automatic variable MM.I2C will hold the result of the transaction.

See also:

BYTE2NUM

I2CSDIS

MM.I2C

I2CDIS

I2CSEN

NUM2BYTE

I2CEN

I2CSRCV

SETPIN

I2CSEND

I2CSSEND

I2CSEND

I2CSEND addr, option, sendlen, senddata [,senddata]

Send data to the I2C slave device.

'addr' is the slave i2c address.

'option' is a number between 0 and 3

1 = keep control of the bus after the command (a stop condition will not be sent at the completion of the command)

2 = treat the address as a 10 bit address

3 = combine 1 and 2 (hold the bus and use 10 bit addresses).

'sendlen' is the number of bytes to send.

'senddata' is the data to be sent - this can be specified in various ways (all values sent will be between 0 and 255):

The data can be supplied in the command as individual bytes. Example: I2CSEND &H6F, 1, 3, &H23, &H43, &H25

The data can be in a one dimensional array (the subscript does not have to be zero and will be honored also bounds checking is performed). Example: I2CSEND &H6F, 1, 3, ARRAY(0)

The data can be a string variable (not a constant). Example: I2CSEND &H6F, 1, 3, STRING\$

The automatic variable MM.I2C will hold the result of the transaction.

See also:

BYTE2NUM

I2CSDIS

MM.I2C

I2CDIS

I2CSEN

NUM2BYTE

I2CEN

I2CSRCV

SETPIN

I2CRCV

I2CSSEND

I2CSDIS

I2CSDIS

Disables the slave I2C module and returns the external I/O pins 12 and 13 to a "not configured" state. Then can then be configured as per normal using SETPIN.

See also:

BYTE2NUM	I2CSEND	MM.I2C
I2CDIS	I2CSEN	NUM2BYTE
I2CEN	I2CSRCV	SETPIN
I2CRCV	I2CSSEND	

I2CSEN

I2CSEN addr, mask, option, send_int, rcv_int

Enables the I2C module in slave mode.

'addr' is the slave i2c address

'mask' is the address mask (bits set as 1 will always match)

'option' is a number between 0 and 3

1 = allows MMBasic to respond to the general call address. When this occurs the value of MM.I2C will be set to 4.

2 = treat the address as a 10 bit address

3 = combine 1 and 2 (respond to the general call address and use 10 bit addresses).

'send_int_line' is the line number of a send interrupt routine to be invoked when the module has detected that the master is expecting data

'rcv_int_line' is the line number of a receive interrupt routine to be invoked when the module has received data from the master.

See also:

BYTE2NUM

I2CSEND

MM.I2C

I2CDIS

I2CSDIS

NUM2BYTE

I2CEN

I2CSRCV

SETPIN

I2CRCV

I2CSSEND

I2CSRCV

I2CSRCV rcvlen, rcvbuf, rcvd

Receive data from the I2C master device. This command should be used in the receive interrupt (ie in the 'rcv_int_line' when the master has sent some data). Alternatively a flag can be set in the receive interrupt routine and the command invoked from the main program loop when the flag is set.

'rcvlen' is the maximum number of bytes to receive.

'rcvbuf' is the variable to receive the data - this is a one dimensional array or if rcvlen is 1 then this may be a normal variable. The array subscript does not have to be zero and will be honored, also bounds checking is performed.

'rcvd' will contain actual number of bytes received by the command.

See also:

BYTE2NUM	I2CSEND	MM.I2C
I2CDIS	I2CSDIS	NUM2BYTE
I2CEN	I2CSEN	SETPIN
I2CRCV	I2CSSEND	

I2CSSEND

I2CSSEND sendlen, senddata [,senddata]

Send the data to the I2C master. This command should be used in the send interrupt (ie in the 'send_int_line' when the master has requested data).

Alternatively a flag can be set in the send interrupt routine and the command invoked from the main program loop when the flag is set.

'sendlen' is the number of bytes to send.

'senddata' is the data to be sent. This can be specified in various ways, see the I2CSEND commands for details.

See also:

BYTE2NUM	I2CSEND	MM.I2C
I2CDIS	I2CSDIS	NUM2BYTE
I2CEN	I2CSEN	SETPIN
I2CRCV	I2CSRCV	

INPUT\$ (from serial)

INPUT(nbr, [#]fnbr)

Will return a string composed of 'nbr' characters read from a file previously opened for INPUT with the file number 'fnbr'. This function will read all characters including carriage return and new line without translation.

When reading from a serial communications port this will return as many characters as are waiting in the receive buffer up to 'nbr'. If there are no characters waiting it will immediately return with an empty string. The # is optional. Also see the OPEN command.

See also:

CLOSE

LOC

OPEN

CLOSE CONSOLE

LOF

OPEN AS COLSOLE

LOC

LOC([#]nbr)

Will return the number of bytes waiting in the receive buffer of a serial port (ie, COM1: or COM2:) that has been opened as #nbr. The # is optional.

See also:

CLOSE	LOF
CLOSE CONSOLE	OPEN
INPUT\$	OPEN AS COLSOLE

LOF

LOF([#]nbr)

Will return the space (in bytes) remaining in the transmit buffer of a serial port (ie, COM1: or COM2:) that has been opened as #nbr. The # is optional.

See also:

CLOSE

INPUT\$

OPEN

CLOSE CONSOLE

LOC

OPEN AS COLSOLE

MM.I2C

MM.I2C

Is set to indicate the result of an I2C operation.

- 0 = The command completed without error.
- 1 = Received a NACK response
- 2 = Command timed out
- 4 = Received a general call address (when in slave mode)

See also:

BYTE2NUM

I2CSEND

I2CSSEND

I2CDIS

I2CSDIS

NUM2BYTE

I2CEN

I2CSEN

I2CRCV

I2CSRCV

MM.OW

MM.OW

MM.OW variable is set to 1 = OK (one-wire device presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

See also:

OWCRC8

OWREAD

OWSEARCH

OWCRC16

OWRESET

OWWRITE

NUM2BYTE

NUM2BYTE number, array(x)

NUM2BYTE number, variable1, variable2, variable3, variable4

Convert 'number' to four numbers containing the four separate bytes of 'number' (MMBasic numbers are stored as the C float type and are four bytes in length).

The bytes can be returned as four separate variables, or as four elements of 'array' starting at index 'x'.

See the function BYTE2NUM() for the reverse of this command.

See also:

BYTE2NUM

I2CSEND

I2CSSEND

I2CDIS

I2CSDIS

MM.I2C

I2CEN

I2CSEN

I2CRCV

I2CSRCV

OPEN (serial)

OPEN comspec\$ AS [#]fnbr

Will open a serial port for reading and writing. Two ports are available (COM1: and COM2:) and both can be open simultaneously.

'comspec\$' is the serial port specification and has the form:

"COMn: baud, buf, int, intlevel" with an optional ",FC" and/or ",OC" appended.

COM1: uses pin 15 for receive data and pin 16 for transmit data and if flow control is specified pin 17 for RTS and pin 18 for CTS.

COM2: uses pin 19 for receive data and pin 20 for transmit data on the CGMMSTICK and D0 (receive) and D1 (transmit) on the CGCOLORMAX.

If the port is opened using 'fnbr' the port can be written to and read from using any commands or functions that use a file number.

See also:

CLOSE

LOC

SETPIN

CLOSE CONSOLE

LOF

INPUT\$

OPEN AS COLSOLE

OPEN AS CONSOLE (serial)

OPEN comspec\$ AS console

A serial port can be opened with "AS CONSOLE". In this case any data received will be treated the same as keystrokes received from the keyboard and any characters sent to the video output will also be transmitted via the serial port. This enables the remote control of MMBasic via a serial interface.

See also:

CLOSE

LOC

SETPIN

CLOSE CONSOLE

LOF

INPUT\$

OPEN

OWCRC8

OWCRC8(len, cdata [, cdata...])

Processes the cdata and returns the 8 bit CRC

len - length of data to process

cdata - data to process

The cdata can be a string, array or a list of variables

See also:

MM.OW

OWREAD

OWSEARCH

OWCRC16

OWRESET

OWWRITE

OWCRC16

OWCRC16(len, cdata [, cdata...])

Processes the cdata and returns the 16 bit CRC

len - length of data to process

cdata - data to process

The cdata can be a string, array or a list of variables

See also:

MM.OW

OWREAD

OWSEARCH

OWCRC8

OWRESET

OWWRITE

OWREAD

OWREAD pin, flag, length, data [, data...]

pin - the MMBasic I/O pin to use

flag - a combination of the following options:

1 - send reset before command

2 - send reset after command

4 - only send/recv a bit instead of a byte of data

8 - invoke a strong pullup after the command (the pin will be set high and open drain disabled)

length - length of data to send or receive

data - data to send or receive

After the command is executed, the pin will be set to the not configured state unless flag option 8 is used. The data and ser arguments can be a string, array or a list of variables.

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable. to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

```
' Read data/temperature command
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &hBE - read data command
OWWRITE 20, 1, 2, &hCC, &hBE

' Read data/temperature from DS1822
' Pin 20, send reset after, read two bytes
OWREAD 20, 2, 2, LowTemp, HighTemp

' Combine and adjust according to data sheet
Cel = ((HighTemp And &b111) * 256 + LowTemp) / 16

' Print Celsius and convert/print Fahrenheit
Print "Celsius" Cel
Print "Fahrenheit" ( 9 / 5 ) * Cel + 32;
```

See also:

MM.OW

OWRESET

SETPIN

OWCRC8

OWSEARCH

OWCRC16

OWWRITE

OWRESET

OWRESET pin [,presence]

pin - the MMBasic I/O pin to use

presence - an optional variable to receive the presence pulse (1 = device response, 0 = no device)

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable. to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

```
' Reset the 1w bus on I/O Pin 20
OWRESET 20
```

See also:

MM.OW

OWREAD

SETPIN

OWCRC8

OWSEARCH

OWCRC16

OWWRITE

OWSEARCH

OWSEARCH pin, srchflag, ser [,ser...]

pin - the MMBasic I/O pin to use

srchflag - a combination of the following options:

1 - start a new search

2 - only return devices in alarm state

4 - search for devices in the requested family (first byte of ser)

8 - skip the current device family and return the next device

16 - verify that the device with the serial number in ser is available

If srchflag = 0 (or 2) then the search will return the next device found

ser - serial number (8 bytes) will be returned (srchflag 4 and 16 will also use the values in ser)

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable. to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

See also:

MM.OW

OWREAD

SETPIN

OWCRC8

OWRESET

OWCRC16

OWWRITE

OWWRITE

OWWRITE pin, flag, length, data [, data...]

pin - the MMBasic I/O pin to use

flag - a combination of the following options:

1 - send reset before command

2 - send reset after command

4 - only send/recv a bit instead of a byte of data

8 - invoke a strong pullup after the command (the pin will be set high and open drain disabled)

length - length of data to send or receive

data - data to send or receive

After the command is executed, the pin will be set to the not configured state unless flag option 8 is used. The data and ser arguments can be a string, array or a list of variables.

The OWRESET and OWSEARCH commands (and the OWREAD and OWWRITE commands if a reset is requested) set the MM.OW variable. to 1 = OK (presence detected, search successful) or 0 = Fail (presence not detected, search unsuccessful).

```
' Start temperature conversion on DS1822 device
' Pin 20, send reset first, send two bytes
' &hCC - Skip ROM
' &h44 - Start conversion
OWWRITE 20, 1, 2, &hCC, &h44
```

See also:

MM.OW

OWREAD

SETPIN

OWCRC8

OWRESET

OWCRC16

OWSEARCH

PEEK

PEEK(hiword, loword)

PEEK(keyword, ±offset)

Will return a byte within the PIC32 virtual memory space.

The address is specified by 'hiword' which is the top 16 bits of the address while 'loword' is the bottom 16 bits.

Alternatively 'keyword' can be used and 'offset' is the \pm offset from the address of the keyword. The keyword can be VIDEO (CGMMSTICK video buffer) or RVIDEO, GVIDEO, BVIDEO (red, blue and green video buffers on the CGCOLORMAX), PROGMEM (program memory) or VARTBL (the variable table). The input keystroke buffer is KBUF, the position of the head of the keystroke queue is KHEAD and the tail is KTAIL (the buffer is 256 bytes long).

See the POKE command for notes and warnings related to memory access.

See also:

POKE

PIN (command/function)

PIN(pin) = value

For a 'pin' configured as digital output this will set the output to low ('value' is zero) or high ('value' non zero). You can set an output high or low before it is configured as an output and that setting will be the default output when the SETPIN command takes effect.

'pin' zero is a special case and will always control the LED on the front panel. A 'value' of non zero will turn the LED on, or zero for off.

PIN(pin)

PIN returns the value on the external I/O 'pin'. Zero means digital low, 1 means digital high and for analog inputs it will return the measured voltage as a floating point number.

Frequency inputs will return the frequency in Hz (maximum 200KHz). A period input will return the period in milliseconds while a count input will return the count since reset (counting is done on the positive rising edge). The count input can be reset to zero by resetting the pin to counting input (even if it is already so configured).

'pin' zero is a special case which will always return the state of the bootload push button on the PC board (non zero means that the button is down).

Also see the SETPIN command.

See also:

PORT

PWM

SETPIN

PULSE

PWM STOP

PLAYMOD

PLAYMOD file [, dur]

Play synthesised music or sound effects. 'file' specifies a file which must be located on the internal drive A: and must be in the .MOD format.

'dur' specifies the duration in milliseconds that the audio will play for - if not specified it will play until explicitly stopped or the program terminates.

The audio is synthesized in the background and is not disturbed by the running program.

The command PLAYMOD STOP will immediately halt any music or sound effect that is currently playing.

NOTE: The file needs to be located on the internal drive A: for performance reasons, it will not play from the SD card.

See also:

PLAYMOD STOP

TONE

SOUND

TONE STOP

PLAYMOD STOP

PLAYMOD STOP

The command PLAYMOD STOP will immediately halt any music or sound effect that is currently playing.

See also:

PLAYMOD

TONE

SOUND

TONE STOP

POKE

POKE hiword, loword, val

POKE keyword, offset, val

Will set a byte within the PIC32 virtual memory space.

The address is specified by 'hiword' which is the top 16 bits of the address while 'loword' is the bottom 16 bits.

Alternatively 'keyword' can be used and 'offset' is the \pm offset from the address of the keyword. The keyword can be VIDEO (CGMMSTICK video buffer) or RVIDEO, GVIDEO, BVIDEO (red, blue and green video buffers on the CGCOLORMAX), PROGMEM (program memory) or VARTBL (the variable table). The input keystroke buffer is KBUF, the position of the head of the keystroke queue is KHEAD and the tail is KTAIL (the buffer is 256 bytes long).

This command is for expert users only. The PIC32 maps all control registers, flash (program) memory and volatile (RAM) memory into a single address space so there is no need for INP or OUT commands.

The PIC32MX5XX/6XX/7XX Family Data Sheet lists the details of this address space.

Note that MMBasic stores most data (including video) as 32 bit integers and the PIC32 uses little endian format.

WARNING: No validation of the parameters is made and if you use this facility to access an invalid memory address you will get an "internal error" which causes the processor to reset and clear all memory.

See also:

PEEK

PORT (command/function)

PORT(start, nbr) = value

Set a number of I/O consecutive pins simultaneously (ie, with one command).

'start' is an I/O pin number and the lowest bit in 'value' (bit 0) will be used to set that pin. Bit 1 will be used to set the pin 'start' plus 1, bit 2 will set pin 'start'+2 and so on for 'nbr' number of bits. The I/O pins used must be numbered consecutively and configured as outputs before this command is used.

For example;

```
PORT(4, 8) = &B10000011
```

will set eight consecutive I/O pins starting with pin 4. Pins 4, 5 and 12 will be set high while 6 to 11 will be set to a low.

PORT(start, nbr)

Returns the value of a number of consecutive I/O pins in one operation.

'start' is an I/O pin number and its value will be returned as bit 0. 'start'+1 will be returned as bit 1, 'start'+2 will be returned as bit 2, and so on for 'nbr' number of bits. The I/O pins used must be numbered consecutively and configured as inputs before this command is used.

This command can be used to conveniently communicate with parallel devices like LCD displays. Any number of I/O pins (and therefore bits) can be used from 1 pin up to 23 pins.

See also:

PIN **PWM STOP**

PWM **SETPIN**

PULSE

PULSE pin, width

Will generate a pulse on 'pin' with duration of 'width' mS.

'width' can be a fraction. For example, 0.01 is equal to 10 µS This enables the generation of very narrow pulses.

The generated pulse is of the opposite polarity to the state of the I/O pin when the command is executed. For example, if the output is set high the PULSE command will generate a negative going pulse.

Notes: For a pulse of less than 3 mS the accuracy is $\pm 1 \mu\text{S}$. For a pulse of 3 mS or more the accuracy is $\pm 0.5 \text{ mS}$. A pulse of 3 mS or more will run in the background. 'pin' must be configured as an output.

See also:

PIN

PWM STOP

PWM

SETPIN

PWM

PWM freq, ch1, ch2

Generate a pulse width modulated (PWM) output for driving analogue circuits.

'freq' is the output frequency (between 20 Hz and 1 MHz) . The frequency can be changed at any time by issuing a new PWM command.

The output will run continuously in the background while the program is running and can be stopped using the PWM STOP command.

'ch1' and 'ch2' are the output duty cycles for channel 1 and 2 as a percentage. If the percentage is close to zero the output will be a narrow positive pulse, if 50 a square wave will be generated and if close to 100 it will be a very wide positive pulse. Both are optional and if not specified will default to the previously used duty cycle for that channel.

The PWM output is generated on the PWM/sound connector and that assumes that the connector has been wired for PWM output. The frequency of the output is locked to the PIC32 crystal and is very accurate and for frequencies below 100 KHz the duty cycle will be accurate to 0.1%.

The original monochrome Maximite has only one PWM/sound output so only 'ch1' can be set on that model.

See also:

PIN **PWM STOP**

PULSE **SETPIN**

PWM STOP

PWM STOP

Stop PWM output.

See also:

PIN

PWM

PULSE

SETPIN

SETPIN

SETPIN pin, cfg [,target]

Will configure the external I/O 'pin' according to 'cfg':

The original Maximite has 20 I/O pins numbered 1 to 20, the Color Maximite adds another 20 I/O pins on the Arduino connector. These are labeled D0 to D13 and A0 to A5.

- 0 Not configured or inactive
- 1 Analog input (pins 1 to 10, A0 to A5)
- 2 Digital input (all pins and 5V tolerant on pins 11 to 20)
- 3 Frequency input (pins 11 to 14)
- 4 Period input (pins 11 to 14)
- 5 Counting input (pins 11 to 14)
- 6 Interrupt on low to high input change (pins 1 to 20, D0 to D8) for target
- 7 Interrupt on high to low input change (pins 1 to 20, D0 to D8) for target
- 8 Digital output (all pins)
- 9 Open collector digital output to 5V (pins 11 to 20, D0 to D13) In this mode the function PIN() will also return the output value.

The starting line number of the interrupt routine is specified in the third parameter 'target'. This mode also configures the pin as a digital input so the value of the pin can always be retrieved using the function PIN().

See also IRETURN to return from the interrupt.

See also:

IRETURN

PORT

PWM

PIN

PULSE

PWM STOP

SOUND

SOUND freq [, dur]

Generate a single tone of 'freq' (between 20Hz and 1MHz) for 'dur' milliseconds. The sound is played in the background and does not stop program execution.

If 'dur' is not specified the sound will play forever until turned off. If 'dur' is zero, any active SOUND statement is turned off.

The command has been replaced with the TONE command that generates a pure sine wave (not a square wave).

See also:

PLAYMOD

TONE

PLAYMOD STOP

TONE STOP

SPI

SPI(rx, tx, clk[, data[, speed[, mode[, bits]]]])

Sends and receives a byte using the SPI protocol with MMBasic as the master (ie, it generates the clock).

'rx' is the pin number for the data input (MISO)

'tx' is the pin number for the data output (MOSI)

'clk' is the pin number for the clock generated by MMBasic (CLK)

'data' is optional and is an integer representing the data byte to send over the data output pin. If it is not specified the 'tx' pin will be held low.

'speed' is optional and is the speed of the clock. It is a single letter either H, M or L where H is 500KHz, M is 50KHz and L is 5KHz. Default is H.

'mode' is optional and is a single numeric digit representing the transmission mode. The default mode is 3.

'bits' is optional and represents the number of bits to send/receive. Range is 1 to 23 (this limit is defined by how many bits can be stored in a floating point number). The default is 8.

Mode CPOL CPHA Description

Mode: 0 (Clock polarity: 0 Clock phase: 0) Clock is active high, data is captured on the rising edge and output on the falling edge

Mode: 1 (Clock polarity: 0 Clock phase: 1) Clock is active high, data is captured on the falling edge and output on the rising edge

Mode: 2 (Clock polarity: 1 Clock phase: 0) Clock is active low, data is captured on the falling edge and output on the rising edge

Mode: 3 (Clock polarity: 1 Clock phase: 1) Clock is active low, data is captured on the rising edge and output on the falling edge

See also:

SETPIN

TONE

TONE left [, right [, dur]]

Generates a continuous sine wave on the sound output. 'left' and 'right' are the frequencies to use for the left and right channels. The tone plays in the background (the program will continue running after this command) and 'dur' specifies the number of milliseconds that the tone will sound for. If the duration is not specified the tone will continue until explicitly stopped or the program terminates.

The command TONE STOP will immediately halt the tone output.

The frequency can be from 1Hz to 20KHz and is very accurate (it is based on the PIC32 crystal oscillator). The frequency can be changed at any time by issuing a new TONE command.

In the monochrome Maximite and compatibles only the left frequency will play but a dummy or empty value is still required for the right channel.

See also:

PLAYMOD

SOUND

PLAYMOD STOP

TONE STOP

TONE STOP

TONE STOP

The command TONE STOP will immediately halt the tone output.

See also:

PLAYMOD

SOUND

PLAYMOD STOP

TONE

FAQ – Frequently Asked Questions

Q: When I add a CAN or RS485 driver chip to the CGCOLORMAX2 do I have to connect the driver to the microcontroller I/O lines?

A: For the CGCOLORMAX1 and RS485 the logic level lines needed to be connected, as the RS485 circuit was floating separately from the rest of the design.

For the CGCOLORMAX2 the CAN and RS485 logic level lines are connected to the appropriate lines of the microcontroller. Without the driver chips in place, the circuits are effectively not there. Once the chips (through-hole DIP packages) are installed, the circuits become a part of the CGCOLORMAX2.

Q: When I add the CAN driver chip to the CGCOLORMAX2, do I lose some I/O lines? And what CAN driver is recommended?

A: Yes. The data lines for the CAN driver chip connect to pins D4 and D5 on the shield connector. To avoid interference any shield used should not have a connection to D4 and D5. One recommended CAN driver chip is the Microchip MCP2551.

Q: When I add the RS485 driver chip to the CGCOLORMAX2, do I lose some I/O lines? And what RS485 driver is recommended?

A: RS485 capability makes use of MMBasic serial port 1 TX, RX, and RTS lines. Adding the RS485 driver chip uses these lines and renders them unusable as general I/O lines. These lines are MMBasic I/O pins 15-17. There are quite a few RS485 driver chips that have the same pin out. The TI SN75176 is one such appropriate chip.

Q: How does the RS232 driver chip support one or two serial ports?

A: MMBasic serial port 1 can be TX/RX, or it can be TX/RX/RTS/CTS. When the RS232 driver chip is used for all four signal lines of MMBasic serial port 1, then the driver chip can support level conversion for TX/RX/RTS/CTS.

As an alternate, two RS232 serial channels can be supported by the driver chip. MMBasic serial port 1 would use just TX/RX for communication (not the RTS/CTS lines of MMBasic serial port 1) and MMBasic serial port 2 TX/RX lines would connect to the RS232 chips other driver/receiver pair that are labeled RTS (for MMBasic serial port 2 TX) and CTS (for MMBasic serial port 2 RX).

The MAX232 driver chip is just two RS232 transmit/drivers and two receivers. You can exchange them as needed.

Q: What driver chip should be used as an RS232 driver?

A: The Maxim MAX232.

Q: When I insert the RS232 chip, am I automatically connected to the microcontroller like the RS485 chip does?

A: No. For the CGCOLORMAX1 and CGCOLORMAX2 you have to hook up the logic level (MMBasic port lines) to the chips. This is set up so that you have the greatest flexibility in using the RS232 driver chip.

Q: Does the shield reset line reset the MMBasic microcontroller?

A: Yes. That reset line is active low, by the way.

ASCII Tables

Dec	Hex	Bin	Char	
0	&h00	&b00000000	NUL	Null
1	&h01	&b00000001	SOH	Start of heading
2	&h02	&b00000010	STX	Start of text
3	&h03	&b00000011	ETX	End of text
4	&h04	&b00000100	EOT	End of transmission
5	&h05	&b00000101	ENQ	Enquiry
6	&h06	&b00000110	ACK	Acknowledge
7	&h07	&b00000111	BEL	Bell
8	&h08	&b00001000	BS	Backspace
9	&h09	&b00001001	TAB	Horizontal tab
10	&h0A	&b00001010	LF	Line feed
11	&h0B	&b00001011	VT	Vertical tab
12	&h0C	&b00001100	FF	Form feed
13	&h0D	&b00001101	CR	Carriage return
14	&h0E	&b00001110	SO	Shift out
15	&h0F	&b00001111	SI	Shift in
16	&h10	&b00010000	DLE	Data link escape
17	&h11	&b00010001	DC1	Device control 1
18	&h12	&b00010	DC2	Device control 2
19	&h13	&b00010	DC3	Device control 3
20	&h14	&b00010	DC4	Device control 4
21	&h15	&b00010	NAK	Negative acknowledge
22	&h16	&b00010	SYN	Synchronous idle
23	&h17	&b00010	ETB	End of transmission block
24	&h18	&b00011	CAN	Cancel
25	&h19	&b00011	EM	End of medium
26	&h1A	&b00011	SUB	Substitute
27	&h1B	&b00011	ESC	Escape
28	&h1C	&b00011	FS	File separator
29	&h1D	&b00011	GS	Group separator
30	&h1E	&b00011	RS	Record separator
31	&h1F	&b00011	US	Unit separator

Dec	Hex	Bin	Char	
32	&h20	&b00100000		Space
33	&h21	&b00100001	!	Exclamation
34	&h22	&b00100010	"	Double quote
35	&h23	&b00100011	#	Hash
36	&h24	&b00100100	\$	Dollar
37	&h25	&b00100101	%	Percent
38	&h26	&b00100110	&	And, ampersand
39	&h27	&b00100111	'	Single quote
40	&h28	&b00101000	(Open parenthesis
41	&h29	&b00101001)	Closed parenthesis
42	&h2A	&b00101010	*	Asterisk
43	&h2B	&b00101011	+	Plus
44	&h2C	&b00101100	,	Comma
45	&h2D	&b00101101	-	Minus, dash
46	&h2E	&b00101110	.	Period
47	&h2F	&b00101111	/	Forward slash
48	&h30	&b00110000	0	Zero
49	&h31	&b00110001	1	One
50	&h32	&b00110010	2	Two
51	&h33	&b00110011	3	Three
52	&h34	&b00110100	4	Four
53	&h35	&b00110101	5	Five
54	&h36	&b00110110	6	Six
55	&h37	&b00110111	7	Seven
56	&h38	&b00111000	8	Eight
57	&h39	&b00111001	9	Nine
58	&h3A	&b00111010	:	Colon
59	&h3B	&b00111011	;	Semi colon
60	&h3C	&b00111100	<	Lesser
61	&h3D	&b00111101	=	Equal
62	&h3E	&b00111110	>	Greater
63	&h3F	&b00111111	?	Question mark

Dec	Hex	Bin	Char	
64	&h40	&b01000000	@	At symbol (apetail)
65	&h41	&b01000001	A	Alpha
66	&h42	&b01000010	B	Bravo
67	&h43	&b01000011	C	Charlie
68	&h44	&b01000100	D	Delta
69	&h45	&b01000101	E	Echo
70	&h46	&b01000110	F	Foxtrot
71	&h47	&b01000111	G	Golf
72	&h48	&b01001000	H	Hotel
73	&h49	&b01001001	I	India
74	&h4A	&b01001010	J	Juliett
75	&h4B	&b01001011	K	Kilo
76	&h4C	&b01001100	L	Lima
77	&h4D	&b01001101	M	Mike
78	&h4E	&b01001110	N	November
79	&h4F	&b01001111	O	Oscar
80	&h50	&b01010000	P	Papa
81	&h51	&b01010001	Q	Quebec
82	&h52	&b01010010	R	Romeo
83	&h53	&b01010011	S	Sierra
84	&h54	&b01010100	T	Tango
85	&h55	&b01010101	U	Uniform
86	&h56	&b01010110	V	Victor
87	&h57	&b01010111	W	Whiskey
88	&h58	&b01011000	X	X-ray
89	&h59	&b01011001	Y	Yankee
90	&h5A	&b01011010	Z	Zulu
91	&h5B	&b01011011	[Opening square bracket
92	&h5C	&b01011100	\	Backslash
93	&h5D	&b01011101]	Closing square bracket
94	&h5E	&b01011110	^	Caret
95	&h5F	&b01011111	_	Underscore

Dec	Hex	Bin	Char	
96	&h60	&b01100000	`	Opening single quote
97	&h61	&b01100001	a	Able
98	&h62	&b01100010	b	Baker
99	&h63	&b01100011	c	Charlie
100	&h64	&b01100100	d	Dog
101	&h65	&b01100101	e	Easy
102	&h66	&b01100110	f	Fox
103	&h67	&b01100111	g	George
104	&h68	&b01101000	h	How
105	&h69	&b01101001	i	Item
106	&h6A	&b01101010	j	Jig
107	&h6B	&b01101011	k	King
108	&h6C	&b01101100	l	Love
109	&h6D	&b01101101	m	Mike
110	&h6E	&b01101110	n	Nan
111	&h6F	&b01101111	o	Oboe
112	&h70	&b01110000	p	Peter
113	&h71	&b01110001	q	Queen
114	&h72	&b01110010	r	Roger
115	&h73	&b01110011	s	Sugar
116	&h74	&b01110100	t	Tare
117	&h75	&b01110101	u	Uncle
118	&h76	&b01110110	v	Victor
119	&h77	&b01110111	w	William
120	&h78	&b01111000	x	X-ray
121	&h79	&b01111001	y	Yoke
122	&h7A	&b01111010	z	Zebra
123	&h7B	&b01111011	{	Opening curly bracket
124	&h7C	&b01111100		Vertical line
125	&h7D	&b01111101	}	Closing curly bracket
126	&h7E	&b01111110	~	Tilde
127	&h7F	&b01111111	DEL	Delete

Index

1-Wire.....	482	CGCOLORMAX1 Technical Information.....	162
1-Wire Interface Example.....	117	CGKEYCHIP1.....	137 , 195
8.3 format.....	457	CGMMSTICK.....	11, 13, 17, 26, 307
A:.....	456, 463, 471	CGMMSTICK / CGCOLORMAX Setup.....	35
ABORT.....	476	CGMMSTICK and CGCOLORMAX I/O Characteristics	204
ABS.....	420	CGMMSTICK1 Technical Information.....	144
absolute value.....	420	CGVGAKB1 Schematic.....	152
Amiga.....	494	CHAIN.....	292
analog.....	481	character.....	439
Analog Input Example.....	81	CHDIR.....	457p.
APPEND.....	474, 477	CHR\$.....	439
arc tangent.....	421	CiA.....	493
ARDUINO SHIELD.....	481	CINT.....	422
Arduino Shield footprint.....	168, 188	CIRCLE.....	320, 328, 347
Arduino Shields.....	15	CircuitGizmos.....	11, 13, 37, 156, 174, 198
arguments.....	221, 224	CLEAR.....	235, 251
array.....	221, 273	CLOSE.....	474
ASC.....	437	CLOSE (file).....	459
ASCII.....	362, 371, 436p., 439, 453	CLOSE (serial).....	500
ASCII Tables.....	542	CLOSE CONSOLE (serial).....	501
ATN.....	421	CLR\$.....	328, 352
audio.....	494	CLS.....	320, 354
AUTO.....	291	COLLISION.....	335, 355
AUTORUN.BAS.....	457	Collision Detection.....	335
B:.....	456, 463, 471	COLOR.....	328, 352, 354, 356
background.....	494	column.....	417
base 10.....	452	command line.....	221
base 16.....	442	command prompt.....	246, 275, 287
base 2.....	438	composite.....	287, 319p., 395p., 411
base 8.....	448	composite video.....	152, 205
battery backed clock.....	288	CONFIG CASE.....	293
BE.....	294	CONFIG COMPOSITE.....	152, 205, 359
BIN\$.....	438	CONFIG KEYBOARD.....	294
binary.....	419, 438, 455	CONFIG TAB.....	295
BLACK.....	329	CONFIG VIDEO.....	361
blank spaces.....	450p.	CONTINUE.....	236, 476
BLIT.....	333, 343	Control-C.....	291
BLUE.....	329	COPY.....	460
BMP.....	320, 380, 411	COPYRIGHT.....	296
box.....	372	COS.....	423
branching.....	222	CTRL C.....	287
break key.....	311	Ctrl-C.....	311
BYTE2NUM.....	496, 516	CTRL-C.....	236
CAN.....	484, 493, 497	current directory.....	458
card insertion.....	146	current working directory.....	458, 461
card removal (CGMMSTICK1).....	149	Custom Keypad Example.....	137
CASE.....	293	CWD\$.....	461
CGCOLORMAX.....	11, 13 , 17, 26, 307	CYAN.....	329
CGCOLORMAX Composite Output.....	205		

D1.....	145	FIX.....	426p.
D2.....	145	font.....	221
D4.....	172, 194	FONT.....	362
D5.....	172, 194	FONT LOAD.....	337, 362
DATA.....	237, 279, 281	FONT LOAD/UNLOAD.....	365
DATE.....	288, 297	FOR.....	270
DATE\$ (command/function).....	297	FOR TO STEP.....	256
decimal.....	452	format specification.....	440
decimal point.....	426	FORMAT\$.....	440
default drive.....	463, 471	FR.....	294
DEG.....	424	full screen editor.....	289
degrees.....	421, 424, 430, 435	function.....	223, 268
delay.....	277	FUNCTION.....	247, 254, 257
DELETE.....	298	function key.....	274
DIM.....	225, 238	functions.....	221
DIR.....	462	games.....	333
DIR\$.....	462	Geoff Graham.....	12, 26
Direct I/O.....	22, 53	GOSUB.....	259, 282
directory.....	478	GOTO.....	260
DO.....	269, 286	GR.....	294
DO LOOP.....	239	GREEN.....	329
DO LOOP UNTIL.....	240	HEX\$.....	442
DO WHILE LOOP.....	241	hexadecimal.....	419, 442, 455
DRIVE.....	457, 463	horizontal position.....	384
EDIT.....	287, 289, 299	I2C.....	481p.
editor.....	290, 299	I2C Interface Example.....	111
ELSE.....	242, 261, 263	I2CDIS.....	502
ELSEIF THEN.....	244	I2CEN.....	503
END.....	246	I2CRCV.....	504
END FUNCTION.....	247, 257	I2CSDIS.....	507
END SUB.....	248, 284	I2CSEN.....	508
ENDIF.....	249	I2CSEND.....	506
endline.....	332, 412	I2CSRVC.....	509
EOF.....	464, 474	I2CSSEND.....	510
ERASE.....	235, 251	IF THEN ELSE.....	261
ERROR.....	301	IF THEN ELSE IF ELSEIF ENDIF.....	265
error number.....	472	IF THEN GOTO.....	263
EXIT.....	239, 252	Implied RUN.....	313
EXIT FOR.....	253	INKEY.....	483
EXIT FUNCTION.....	247, 254, 257	INKEY\$.....	369
EXIT SUB.....	248, 255	INPUT.....	379, 457, 464, 469, 474, 481, 483
EXP.....	425	INPUT (from keyboard).....	370
exponent.....	419	INPUT #.....	466
exponential.....	425	Input Example.....	81
exponential notation.....	419	INPUT\$ (from file).....	467
EXPRESSIONS.....	436	INPUT\$ (from serial).....	511
FAT16.....	221, 457	INSTR.....	443
FAT32.....	221, 457	INT.....	426p.
file.....	474	integers.....	221
FILE.....	462	internal flash.....	221
FILES.....	457, 462, 465	internal flash memory.....	457
fill.....	347, 372	interrupt.....	266

invert.....	328	loops.....	222
IRETURN.....	234, 266, 283, 483	LOWER.....	293
IT.....	294	lowercase.....	444
J1.....	36, 58, 144, 151p., 481	Maximite.....	11p., 17, 26
J10.....	171	Maximite BASIC.....	59
J11.....	171, 192	Maximite I/O Connector.....	167
J12.....	171, 192	MEMORY.....	304
J13.....	168, 188	MERGE.....	288, 305
J14.....	168, 189	micro SD card.....	147
J15.....	168, 189	MID\$.....	447
J16.....	172, 195	milliseconds.....	233, 285
J17.....	195	MKDIR.....	457, 470
J18.....	195	MM.CMDLINE.....	313
J19.....	195	MM.CMDLINE\$.....	306
J20.....	196	MM.DEVICE\$.....	307
J21.....	196	MM.DRIVE.....	463
J3.....	145	MM.DRIVE\$.....	471
J4.....	145, 152	MM.ERRNO.....	472, 474, 476
J5.....	145, 152, 163, 181	MM.FNAME\$.....	308
J6.....	163, 181p.	MM.HPOS.....	384
J7.....	172, 196	MM.HRES.....	320, 385
J8.....	145, 172, 195	MM.I2C.....	514
J9.....	145, 167, 186	MM.OW.....	515
J9A.....	167, 186	MM.VER.....	309
Joystick Connections.....	206	MM.VPOS.....	387
Keyboard.....	319	MM.VRES.....	320, 389
KEYBOARD.....	294	MMBasic.....	24, 48, 59, 210
KILL.....	457, 468	MMC.....	457
label.....	221, 260	MMIDE.....	21, 35, 38, 43, 47
LCASE\$.....	444	MOD.....	494
LCD Interface Example.....	85	MODE.....	330, 332, 391
LCD Shield.....	124	MODE 1,7.....	412
LED.....	527	modulus.....	419
LED Control Example.....	61	NAME.....	457, 473
LEFT\$.....	445	NEW.....	310
LEN.....	446	NEXT.....	256, 270
length.....	446	NTSC.....	152, 205, 320, 326, 359
LET.....	267	NUM2BYTE.....	496, 516
LINE.....	320, 328, 372	OCT\$.....	448
LINE INPUT.....	457, 474	octal.....	419, 448, 455
LINE INPUT (file).....	469	ON nbr GOTO GOSUB.....	272
LINE INPUT (from keyboard).....	379	OPEN.....	457, 482
LIST.....	287, 302	OPEN (file).....	474
LOAD.....	288, 303, 305	OPEN (serial).....	517
LOADBMP.....	320, 380	OPEN AS CONSOLE (serial).....	518
LOC.....	512	OPERATORS AND PRECEDENCE.....	419
LOCAL.....	225, 268	OPTION BASE.....	273
LOCATE.....	381	OPTION BREAK.....	236, 287, 311
LOF.....	513	OPTION ERROR.....	472, 474, 476
LOG.....	428	OPTION Fnn.....	274
logarithm.....	428	OPTION PROMPT.....	275
LOOP UNTIL.....	269	OPTION USB.....	276

OPTION VIDEO.....	395	revision.....	309
OUTPUT.....	474, 477, 481	RIGHT\$.....	449
OWCRC16.....	520	RMDIR.....	457, 478
OWCRC8.....	519	RND.....	431
OWREAD.....	521	Round.....	422
OWRESET.....	523	RS232.....	15
OWSEARCH.....	524	RS485.....	15
OWWRITE.....	525	RTC.....	288
PAL.....	152, 205, 320, 326, 359	RUN.....	288, 292, 313, 457
palette.....	330	SAVE.....	288, 308, 315
parameter lists.....	224	SAVEBMP.....	320, 347, 411 , 413
parent directory.....	458	SCANLINE.....	332, 412
PAUSE.....	233, 277	Schematic (CGMMSTICK1).....	155
PEEK.....	526	screen resolution.....	326
pi.....	429	SD card 456pp., 461p., 465, 468, 470, 472p., 476, 478p.	
PI.....	429	SDHC.....	457
PIN.....	481	sea-of-holes.....	15
PIN (command/function).....	527	serial console.....	319
PIXEL.....	320, 403, 409	Serial I/O Example.....	92
PIXEL (command/function).....	396	SETPIN.....	481, 483, 535
PLAYMOD.....	494, 528	SETTICK.....	233p., 283
PLAYMOD STOP.....	529	SGN.....	432
POKE.....	530	sign.....	432
PORT (command/function).....	531	SIN.....	433
POS.....	402	sine.....	433
precedence.....	419	solderless breadboard.....	13, 30
PRESET.....	403	SOUND.....	494, 536
PRINT.....	405p., 457, 474, 477, 479	Sound Example.....	74
PRINT @.....	406	SPACE.....	451
PROGRAM AND DATA STORAGE.....	457	SPACE\$.....	450
PSET.....	409	SPC.....	451
PULSE.....	532	Special Keyboard Keys.....	143, 311, 341
Pulse Width Modulation.....	495	SPI.....	481p., 537
PURPLE.....	329	SPI Interface Example.....	104
PWM.....	481, 494p., 533	SPRITE.....	333, 414
PWM STOP.....	534	SPRITE LOAD.....	333, 414
RAD.....	430	SPRITE MOVE.....	333, 414
radians.....	421, 423p., 430, 433, 435	SPRITE OFF.....	333, 414
random.....	278	SPRITE OFF ALL.....	414
random number.....	431	SPRITE ON.....	333, 414
RANDOMIZE.....	278, 431	SPRITE UNLOAD.....	414
READ.....	237, 279, 281	SQR.....	434
REAL-TIME.....	288	square root.....	434
RED.....	329	startline.....	332, 412
REM.....	280	stereo.....	494
remainder.....	419	STORAGE.....	288
remarks.....	280	STR\$.....	452
Rename.....	473	string.....	221, 440, 452p.
RENUMBER.....	312	String.....	436
Resolution.....	320	String operators.....	436
RESTORE.....	281	STRING\$.....	453
RETURN.....	259, 282		

STRUCTURED STATEMENTS.....	222	USB interface.....	319
SUB.....	248, 255, 284	Utility.....	51
subroutine.....	223, 268, 284	Vacuum Fluorescent Display Example.....	87
subroutines.....	221	VAL.....	455
substring.....	445, 447, 449	value.....	455
TAB.....	295, 417	variable.....	221
TAN.....	435	version.....	309
tangent.....	435	vertical position.....	387
text strings.....	436	VGA.....	287, 319, 326, 359, 395p., 411
THEN.....	261, 263	VIDEO OFF.....	361, 395
TIME.....	288	VIDEO ON.....	361, 395
TIME\$ (command/function).....	316	VOL.....	462
TIMER.....	233, 278	VT100.....	300
TIMER (command/function).....	285	WEND.....	286
TITLE.....	293	WHILE.....	286
TONE.....	494, 538	WHILE WEND.....	286
TONE STOP.....	539	WHITE.....	329
TROFF.....	317	WRITE.....	457, 474, 479
TRON.....	318	XMODEM.....	288, 480
Truncate.....	426p.	YELLOW.....	329
UCASE\$.....	454	155, 196 , 261, 263
UK.....	294	?.....	405
United States.....	152, 205	'.....	280
UPPER.....	293	@.....	406
uppercase.....	454	&B.....	419, 455
US.....	294	&H.....	419, 455
USB.....	276, 287, 395	&O.....	419, 455