Armmite H7
User
Manual

MMBasic Ver 5.05.07

The Armmite H7 is a new addition to the Micromite family using the STM32H7. The Armmite H7 firmware implements most of the features of the standard Micromite and the Micromite Plus as described in the *Micromite User Manual* and the *Micromite Plus Manual*. It has a number of differences and additional features and they are described in this document.
The focus of this manual is to describe just the features that are **unique** to the Armmite H7. For general Micromite programming you should refer to the *Micromite User Manual* and the *Micromite Plus Manual* in addition to this manual.

Contents

Introduction 3
Micromite Family Summary 4
Hardware Considerations 6
144-pin Armmite H7 Pinout 8
Unique Armmite H7 Features 12
Commands (Armmite H7 Only) 15
Functions (Armmite H7 Only) 29
Appendix A - Sensor Fusion 34
Appendix B - Sprites 35
Appendix C - Nucleo Pin Definitions 37

Introduction

This manual provides an introduction for users who are familiar with the Micromite and the Micromite Plus and need a summary of the extra features in the Armmite H7.
The Armmite H7 is an extension of the standard Micromite and the Micromite Plus; most of the features of these two versions are also in the Armmite H7. This includes features of the BASIC language, input/output, communications, etc. Some commands have changed slightly but for the main part Micromite programs will run unchanged on the Armmite H7.
The following summarises features of the Armmite H7 as compared to the standard Micromite and the Micromite Plus:
The Armmite H7 is based on the STM32H743ZI. It has up to fifteen times the program space of the MX series used in the standard Micromite and is many times faster. The Armmite H7 is roughly 2x faster than even a 252MHz PIC32MZ.
The Armmite H7 uses the built in hardware floating point capability of the STM32H743ZI processor which is much faster than floating point on the standard Micromite and uses double precision floating point.
The Armmite H7 has 96 free I/O pins. The Armmite H7 supports 25 16-bit analogue inputs.
The Armmite H7 has two I^2C ports, four SPI ports, eight high speed PWM channels , two 12-bit DACs, a high speed counter input, and four serial COM ports.

The STM32H743ZI is conveniently available as a package on a development PCB the NUCLEO-H743ZI2 . This is widely available through the normal suppliers (RS, Farnell, Mouser, etc.) at low cost.

Micromite Family Summary

The Micromite Family consists of five major types, the standard Micromite, the Micromite Plus, the Micromite eXtreme, the Pi-cromite and the Armmite H7. All use the same BASIC interpreter and have the same basic capabilities however they differ in the number of I/O pins, the amount of memory, the displays that they support and their intended use.

Standard Micromite	Comes in a 28-pin or 44-pin package and is designed for small embedded controller applications and supports small LCD display panels. The 28-pin version is particularly easy to use as it is easy to solder and can be plugged into a standard 28-pin IC socket.
Micromite Plus	This uses a 64-pin and 100-pin TQFP surface mount package and supports a wide range of touch sensitive LCD display panels from 1.44" to 8" in addition to the standard features of the Micromite. It is intended as a sophisticated controller with easy to create on-screen controls such as buttons, switches, etc.
Micromite eXtreme	This comes in 64, 100-pin and 144-pin TQFP surface mount packages. The eXtreme version has all the features of the other two Micromites but is faster and has a larger memory capacity plus the ability to drive a VGA monitor for a large screen display. It works as a powerful, self contained computer with its own BASIC interpreter and instant start-up.
Pi-cromite	Runs on all versions of the Raspberry Pi with a 40-pin I/O connector. No analogue input capability but 5x faster than a Micromite eXtreme when running on a Pi 3.
Armmite H7	Runs on the NUCLEO-H743ZI processor. This is the highest speed single-chip Micromite currently available.

	Micromite		**Micromite Plus**		**Micromite eXtreme**			**Armmite H7**
	28-pin DIP	44-pin SMD	64-pin SMD	100-pin SMD	100-pin SMD	144-pin SMD	64-pin SMD	NUCLEO-H743ZI2
Maximum CPU Speed	48 MHz	48 MHz	120 MHz	120 MHz	252MHz	252 MHz	252 MHz	480MHz
Maximum BASIC Program Size	59 KB	59 KB	100 KB	100 KB	540 KB	540 KB	540 KB	512KB
RAM Memory Size	52 KB	52 KB	108 KB	108 KB	460 KB	460 KB	460 KB	512KB
Clock Speed (MHz)	5 to 48	5 to 48	5 to 120	5 to 120	200 to 252	200 to 252	200 to 252	400
Total Number of I/O pins	19	33	45	77	75	115	46	102
Number of Analog Inputs	10	13	28	28	40	48	24	26
Number of Serial I/O ports	2	2	3 or 4	3 or 4	3 or 4	3 or 4	3 or 4	4
Number of SPI Channels	1	1	2	2	3	3	2	4
Number of I^2C Channels	1	1	1 + RTC	1 + RTC	2 + RTC	2 + RTC	1 + RTC	2
Number of 1-Wire I/O pins	19	33	45	77	75	115	46	96
PWM or Servo Channels	5	5	5	5	6	6	6	8
Serial Console	✓	✓	✓	✓	✓	✓	✓	✓
USB Console			✓	✓	✓	✓	✓	x
PS2 Keyboard and LCD Console			✓	✓	✓	✓	✓	
USB Keyboard and					✓	✓	✓	✓

LCD Console								
SD Card Interface			✓	✓	✓	✓	✓	✓
Supports ILI9341 LCD Displays	✓	✓	✓	✓	✓	✓	✓	✓
Supports Ten LCD Panels from 1.44" to 8" (diameter)			✓	✓	✓+ ILI9481	✓+ ILI9481	✓+ ILI9481	✓+ ILI9481
Supports VGA Displays					✓	✓		
Sound Output (WAV/tones)			✓	✓	✓	✓	✓	On-chip DACs
Supports PS2 Mouse Input					✓	✓	✓	
Floating Point Precision	Single	Single	Double S/W	Double S/W	Double H/W	Double H/W	Double H/W	Double H/W
Power Requirements	3.3V 30 mA	3.3V 30 mA	3.3V 80 mA	3.3V 80 mA	3.3V 160 mA	3.3V 160 mA	3.3V 160 mA	3.3V 200mA

Hardware considerations

The microcontroller used in the Armmite H7 is the STM32H743ZI manufactured by ST.

The default clock speed of the Armmite H7 is 400 MHz. The clock input to the chip must be an 8MHz external oscillator on pin 23.

The supported chips are:

STM32H743ZI	144-pin LQFP package (0.5 mm pin pitch) – maximum speed 400 MHz
STM32H753ZI	144-pin LQFP package (0.5 mm pin pitch) – maximum speed 400 MHz

144-pin Test and Development Board

The Nucleo-H743ZI2 is a complete module and breaks out all the pins on the chip. It includes a ST-LINK programmer for uploading the Armmite firmware and also provides a console connection via a USB CDC connection. It can be used standalone, or can be mounted directly onto a back pack PCB which provides connections to various LCD panels. Gerbers for the backpack are available on

http://www.thebackshed.com/forum/forum_posts.asp?TID=10700&PN=1

The power selector (JP3) on the Nucleo should be selected to E5V to use the daughter board

It is recommended that the following links on the Nucleo are removed:

JP4, SB13, SB160, JP6, SB127, SB125, SB164, SB178, SB181, JP7, SB183, SB182

In addition you must remove SB156 if using an external battery connected to VBAT

Please download and install ST-LINK/V2 utility software and use it to program the Nucleo rather than STM32CubeProgrammer.It takes a little longer but it correctly programs the chip. My uploads are all done through the development environment which uses ST-LINK. I've just tested the most recent posted binary with both STM32CubeProgrammer and ST-LINK and it only works properly when programmed with ST-LINK.

SDcard, SPI LCD and Touch Connections

The connections required to use an LCD panel are as follows

T_DO, SDO(MISO), SD_DO	Pin-20 (PF8)
T_DIN, SDI(MOSI), SD_DIN	Pin-21 (PF9)
T_CLK, SCK, SD_CLK	Pin-19 (PF7)
SD_CS	configurable
SD_CD	configurable
T_IRQ	configurable
T_CS	configurable

SSD1963, 16-bit ILI9341 Connections

Touch and SDcard connections as above

DB0	Pin-141 (PE0)	DB8	Pin-59 (PE8)	WR	Pin-125 (PG10)
DB1	Pin-142 (PE1)	DB9	Pin-60 (PE9)	RS	Pin-57 (PG1)
DB2	Pin-1 (PE2)	DB10	Pin-63 (PE10)	RESET	Pin-127 (PG12)
DB3	Pin-2 (PE3)	DB11	Pin-64 (PE11)	RD	Pin-128 (PG13) *

DB4	Pin-3 (PE4)	DB12	Pin-65 (PE12)	CS	GND
DB5	Pin-4 (PE5)	DB13	Pin-66 (PE13)		
DB6	Pin-5 (PE6)	DB14	Pin-67 (PE14)		
DB7	Pin-58 (PE7)	DB15	Pin-68 (PE15)		

SSD1963 displays should be configured for backlight control using 1963_PWM. Controlling the backlight using an Armmite H7 pin like the MM+ is not supported.

- Mandatory connection

OV7670 Camera connections

SIOC	Pin-139 (PB8)	SIOD	Pin-140 (PB9)
VSYNC	Pin-89 (PG4)	HREF	Pin-90 (PG5)
PCLK	Pin-88 (PG3)	XCLK	Pin-100 (PA8)
D7	Pin-97 (PC7)	D6	Pin-96 (PC6)
D5	Pin-45 (PC5)	D4	Pin-44 (PC4)
D3	Pin-29 (PC3)	D2	Pin-28 (PC2)
D1	Pin-27 (PC1)	D0	Pin-26 (PC0)
RESET	3.3V	PWDN	GND

144-pin Armmite H7 Pinouts

Pin	Features				
1	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB2		
2	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB3		
3	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB4		
4	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB5		
5	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB6		
6	VBAT				
7	DIGITAL_IN	DIGITAL_OUT	NUCLEO-B1		
8	32KHz Xtal				
9	32KHz Xtal				
10	DIGITAL_IN	DIGITAL_OUT	COUNT1		
11	DIGITAL_IN	DIGITAL_OUT	COUNT2		
12	DIGITAL_IN	DIGITAL_OUT	COUNT3		
13	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	COUNT4	
14	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	IR	
15	ANALOG_C	DIGITAL_IN	DIGITAL_OUT		
16	GND				
17	VDD				
18	ANALOG_C	DIGITAL_IN	DIGITAL_OUT		
19	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	SPI5-CLK	
20	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	SPI5-IN	
21	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	SPI5-OUT	
22	ANALOG_C	DIGITAL_IN	DIGITAL_OUT		
23	8MHz OSC				
24	N/C				
25	RESET				
26	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	OV7670-D0	
27	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	SPI2-OUT	OV7670-D1
28	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	SPI2-IN	OV7670-D2
29	ANALOG_C	DIGITAL_IN	DIGITAL_OUT	OV7670-D3	
30	VDD				
31	ANALOG-GND				
32	VREF				
33	ANALOG-VDD				
34	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	PWM-2A	
35	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	PWM-2B	
36	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	PWM-2C	
37	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	PWM-2D	

38	GND			
39	VDD			
40	DAC1	AUDIO-L		
41	DAC2	AUDIO-R		
42	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	SPI-IN
43	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	SPI-OUT
44	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	OV7670-D4
45	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	OV7670-D5
46	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	NUCLEO-GREEN
47	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	
48	DIGITAL_A	DIGITAL_OUT	SPI3-OUT	
49	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	
50	ANALOG_A	DIGITAL_IN	DIGITAL_OUT	
51	GND			
52	VDD			
53	ANALOG_B	DIGITAL_IN	DIGITAL_OUT	
54	ANALOG_B	DIGITAL_IN	DIGITAL_OUT	
55	DIGITAL_IN	DIGITAL_OUT		
56	DIGITAL_IN	DIGITAL_OUT		
57	DIGITAL_IN	DIGITAL_OUT	SSD1963-RS	
58	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB7	
59	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB8	
60	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB9	
61	GND			
62	VSS			
63	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB10	
64	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB11	
65	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB12	
66	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB13	
67	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB14	
68	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB15	
69	I2C2-SCL			
70	I2C2-SDA			
71	VCAP			
72	VDD			
73	DIGITAL_IN	DIGITAL_OUT	COM3-RX	
74	DIGITAL_IN	DIGITAL_OUT	COM3-TX	
75	DIGITAL_IN	DIGITAL_OUT	NUCLEO-RED	
76	DIGITAL_IN	DIGITAL_OUT	COM1-RX	
77	CONSOLE-TX			
78	CONSOLE-RX			

79	DIGITAL_IN	DIGITAL_OUT	
80	DIGITAL_IN	DIGITAL_OUT	
81	DIGITAL_IN	DIGITAL_OUT	PWM-1A
82	DIGITAL_IN	DIGITAL_OUT	PWM-1B
83	GND		
84	VDD		
85	DIGITAL_IN	DIGITAL_OUT	PWM-1C
86	DIGITAL_IN	DIGITAL_OUT	PWM-1D
87	DIGITAL_IN	DIGITAL_OUT	
88	DIGITAL_IN	DIGITAL_OUT	OV7670_PCLK
89	DIGITAL_IN	DIGITAL_OUT	OV7670_VSYNC
90	DIGITAL_IN	DIGITAL_OUT	OV7670_HREF
91	DIGITAL_IN	DIGITAL_OUT	
92	DIGITAL_IN	DIGITAL_OUT	
93	DIGITAL_IN	DIGITAL_OUT	COUNT5-HS
94	GND		
95	VDDUSB		
96	DIGITAL_IN	DIGITAL_OUT	OV7670-D6
97	DIGITAL_IN	DIGITAL_OUT	OV7670-D7
98	DIGITAL_IN	DIGITAL_OUT	
99	DIGITAL_IN	DIGITAL_OUT	
100	DIGITAL_IN	DIGITAL_OUT	OV7670_XCLK
101	DIGITAL_IN	DIGITAL_OUT	
102	DIGITAL_IN	DIGITAL_OUT	
103	USB-D+		
104	USB-D-		
105	SWDIO		
106	VCAP		
107	GND		
108	VDD		
109	SWCLK		
110	DIGITAL_IN	DIGITAL_OUT	
111	DIGITAL_IN	DIGITAL_OUT	
112	DIGITAL_IN	DIGITAL_OUT	
113	DIGITAL_IN	DIGITAL_OUT	
114	DIGITAL_IN	DIGITAL_OUT	
115	DIGITAL_IN	DIGITAL_OUT	
116	DIGITAL_IN	DIGITAL_OUT	
117	DIGITAL_IN	DIGITAL_OUT	SPI2-CLK
118	DIGITAL_IN	DIGITAL_OUT	COM2-DE
119	DIGITAL_IN	DIGITAL_OUT	COM2-TX

120	GND		
121	VDD		
122	DIGITAL_IN	DIGITAL_OUT	COM2-RX
123	DIGITAL_IN	DIGITAL_OUT	
124	DIGITAL_IN	DIGITAL_OUT	COM4-RX
125	DIGITAL_IN	DIGITAL_OUT	SSD1963_WR
126	DIGITAL_IN	DIGITAL_OUT	SPI-CLK
127	DIGITAL_IN	DIGITAL_OUT	SSD1963_RESET
128	DIGITAL_IN	DIGITAL_OUT	SSD1963_RD
129	DIGITAL_IN	DIGITAL_OUT	COM4-TX
130	GND		
131	VSS		
132	DIGITAL_IN	DIGITAL_OUT	
133	DIGITAL_IN	DIGITAL_OUT	SPI3-CLK
134	DIGITAL_IN	DIGITAL_OUT	SPI3-IN
135	DIGITAL_IN	DIGITAL_OUT	
136	DIGITAL_IN	DIGITAL_OUT	COM1-TX
137	DIGITAL_IN	DIGITAL_OUT	NUCLEO-BLUE
138	BOOT0		
139	I2C-SCL		
140	I2C-SDA		
141	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB0
142	DIGITAL_IN	DIGITAL_OUT	SSD1963-DB1
143	PDR-ON		
144	VDD		

Armmite H7 Features

480MHz clock

The Armmite H7 is the fastest single chip implementation of MMBasic

512Kbyte program and 498Kbyte variable space

The Armmite H7 supports MMBasic programs up to 512Kbytes in size. By default the maximum program size is set to 128Kb to expedite loading and saving programs but this can be extended as required with the ***OPTION FLASHPAGES*** command. Variable space is always 498Kbyte.

Dual 12-bit DACs

The Armmite H7 has 2 12-bit DACs built into the chip. The analogue levels can be set using the DAC command. In addition they are used for the PLAY TONE, PLAY WAV, PLAY FLAC, PLAY MP3 and TTS commands. The pins cannot be used for general purpose I/O. The DACs support an arbitrary function generator capability using the DAC START command

Double Precision Floating Point

The Armmite H7 uses the hardware floating point capability of the STM32H743ZI chip and can therefore process floating point calculations faster than the Micromite and Micromite Plus. All floating point uses double precision calculations.

Eight PWM Channels

Minimum frequency is 1Hz, maximum is 24MHz. Duty cycle and frequency accuracy will depend on frequency. The frequency can be any value of 240,000,000/n.

Three SPI Channels

The Armmite H7 supports three SPI channels. The second and third channels operate the same as the first, the only difference is that the commands use the notation SPI2 and SPI3 (for example SPI3 WRITE, etc).

Note that if the Armmite H7 is configured for a SPI based LCD panel, touch or an SD card then an additional SPI channel is used and does not impact the other three.

MM.DEVICE$

On the Armmite H7 the read only variable MM.DEVICE$ will return " Armmite H7".

Longstring handling

The Armmite H7 supports a comprehensive set of commands and functions for handling long strings stored in integer arrays

I2C

You can use I2C exactly the same as the Micromite with the following limitations:
The implementation does not support 10-bit addressing (i.e. options 0 and 1 only).
The implementation does not support I2C slave mode

A second I2C channel can be used using the command ***I2C2***.

I2C ports are dedicated and cannot be used for general purpose I/O

On-chip RTC supporting DATE$ and TIME$

Date$ and Time$ are derived from the STM32H743ZI on-chip real time clock. They are returned in standard MMBasic format when read. Setting them will update the real time clock. If a 3V battery is connected the STM32H743ZI will maintain the time even when powered off. An additional MMBasic function ***DAY$*** returns the day of the week as a string. RTC SETTIME and RTC GETTIME are not needed and not supported.

VAR command

The Armmite H7 allows up to 128Kbs of variables to be saved

Resetting the chip and options

The chip can be reset to “just-programmed” state by connecting pin 7 to VDD whilst resetting the chip by grounding the RESET pin.

16-bit ADCs

All analogue to digital conversion can be carried out in 16-bit resolution, 14-bit resolution, 12-bit resolution, 10-bit resolution, and 8-bit resolution. In addition the ADC can read the voltage being output on the DACs, the battery backup voltage, the chip die temperature and the internal reference voltage. Using the ADC command conversion of three channels can be set to run in the background at up to 500,000 samples per second per channel and one of the channels can be set to provide edge-triggering of the conversion.

USB Keyboard support

The Armmite H7 supports a USB keyboard in US or UK format. This is enabled with the ***OPTION USBKEYBOARD*** command. A keyboard can be plugged directly into the micro-USB port of a Nucleo-H743ZI using a suitable adapter. Optionally, a pin can be selected to drive the USB power supply to the keyboard.

Buffered Drivers for all displays

All displays up to and including 480x320 pixel resolution automatically use a buffered driver where a memory image of the display is maintained in the Armmite’s memory. This does not impact user RAM space but helps reduce artefacts whilst writing to the screen. Commands ***OPTION AUTOREFRESH*** and ***REFRESH*** can be used to control when the updates to the screen take place allowing the programmer maximum flexibility in using the screen effectively. An 8-bit buffered driver (64 colours) is included for 800x480 SSD1963 displays. This is specifically targeted at games programming as user RAM is still 473K with the driver loaded. A RGB565 buffered driver is also included for 800x480 displays but this reduces user RAM significantly (down to 98Kbytes free).

16-bit Interface to SSD1963 Based LCD Displays

The Armmite H7 can drive a SSD1963 display using a 16-bit parallel bus for extra speed. The extra I/O pins for this are listed as SSD1963-DB8 to SSD1963-DB15 on the pinout tables in this manual and they must be connected to the pins labelled DB8 to DB15 on the I/O connector on the SSD1963 display.

Note that in this mode the SSD1963 controller runs with a reduce colour range (65 thousand colours) compared to 16 million colours with the normal 8-bit interface.

WS2812 support

The Armmite H7 supports the WS2812 Led driver. This chip needs very specific timing to work properly and by incorporating support in the Armmite H7 firmware the user can program these chips with minimum effort. The command ***WS2812*** is used to set the colours of the LEDs. There is no limit to the size of the WS2812 string supported.

GPS support

The Armmite H7 support connection of a GPS to any of the 4 serial interfaces. The command ***OPEN “COMn:baudrate” as GPS*** is used to enable reception of NMEA GPS messages. THE ***GPS()*** function can then be used to interrogate the GPS data which is automatically parsed in the Armmite firmware. In addition ***PRINT #GPS,string$*** can be used to automatically append a correct checksum to a GPS message.

High speed frequency counter support

The Armmite H7 supports a frequency counter input to pin 93. This supports counting high speed signals (tested to 20MHz, but may go considerably higher).

OV7670 camera support with movement detection

The Armmite H7 supports a non-buffered OV7670 camera in full colour 640x480 pixel mode using the ***CAMERA*** command. To use this a 800x480 SSD1963 display must be connected and configured using a special driver (***OPTION LCDPANEL SSD1963_5_640, orientation*** or ***OPTION LCDPANEL SSD1963_7_640, orientation***) in landscape or reverse landscape mode. The driver maintains a complete image in Armmite memory which allows the software to compare a newly captured image with a stored one to check for movement using the ***MOVEMENT()*** function.

Sprites

The Armmite H7 supports a complete implementation of sprites including screen scrolling and collision detection

Extended WAV File Playback

The Armmite H7 can play WAV files (like the Micromite Plus) however, it is also capable of playing WAV files recorded with sampling rates of 24 KHz, 44.1KHz, and 48 KHz.

Random Number Generation

The Armmite H7 uses the hardware random number generator in the STM32 series of chips to deliver true random numbers. This means that the RANDOMIZE command is no longer needed and is not supported.

OPTION VCC command

The Armmite H7 supports the OPTION VCC command. This allows the user to precisely set the supply voltage to the chip and is used in the calculation of voltages when using analogue inputs e.g. OPTION VCC 3.15. The parameter is not saved and should be initialised either on the command line or in a program.

OPTION CPU SPEED command

The Armmite H7 supports the OPTION CPU SPEED command. This allows the user to set the speed of the core processor clock, values between 10MHz and 600MHz are allowed, anything over 480 MHz is overclocking the chip beyond the manufacturers specification.

Unsupported commands

The Armmite H7 does not currently support dynamically changing the CPU speed or the sleep function. Accordingly the commands CPU speed and CPU SLEEP are not available.
The Armmite H7 does not support the LIBRARY command but given the large program space available this should not create any issues. CFunctions are not supported.

Commands (Armmite H7 Only)

ADC	The ADC functionality that can capture up to 3 channels of data in the background at up to 500KHz per channel with user selectable triggering
ADC OPEN frequency, channel1-pin [,channel2-pin] [,channel3-pin] [, interrupt]	Open the ADC channels. "frequency" is the sampling frequency in Hz. The maximum frequency is 500KHz. From 320KHz to 500KHz the conversion is 8-bits per channel From 160KHz to 320KHz the conversion is 10-bits per channel From 80KHz to 160KHz the conversion is 12-bits per channel From 40KHz to 80KHz the conversion is 14-bits per channel Below 40KHz conversion is 16-bits per channel This is automatically compensated for in the firmware. "channel1-pin" can be one of 34, 35, 36, 37, 42, 43, 44, 45, 46, 47, 49, 50 "channel2-pin" can be one of 13, 14, 15, 18, 19, 21, 22, 26, 27, 28, 29 "channel3-pin" can be one of 53, 54 The "interrupt" parameter is a normal MMBasic subroutine that can be called when the conversion completes.
ADC FREQUENCY frequency	Allows the ADC frequency to be adjusted after the ADC START command. This command is only allowed if the number of bits calculated in the table above does not change otherwise it will give an error.
ADC TRIGGER channel, level	Sets up triggering of the ADC. This should be specified before the ADC START command. The "channel" can be a number between one and three depending on the number of pins specified in the ADC OPEN command. The level can be between -VCC and VCC. A positive number indicates that you are looking for a positive going transition through the specified voltage. A negative number indicates that you are looking for a negative going transition through the specified voltage.
ADC START channel1array!() [,channel2array!()] [,channel3array!()]	Starts ADC conversion. The floating point arrays must be the same size and the size will determine the number of samples. Once the start command is issued the ADC(s) will start converting the input signals into the arrays at the frequency specified. If the OPEN command includes an interrupt then the command is non-blocking. If an interrupt is not specified the command is blocking. The samples are returned as floating point values between 0 and VCC.
ADC CLOSE	Closes the ADC and returns the pins to normal use

BOX x1, y1, w, h [, lw] [,c] [,fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, w, and h must all be arrays or all be single variables /constants otherwise an error will be generated. lw, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
ARC x, y, r1, [r2], rad1, rad2, colour	Draws an arc of a circle or a given colour and width between two radials (defined in degrees). Parameters for the ARC command are: x: X coordinate of centre of arc y: Y coordinate of centre of arc r1: inner radius of arc r2: outer radius of arc - can be omitted if 1 pixel wide rad1: start radial of arc in degrees rad2: end radial of arc in degrees colour: Colour of arc
BEZIER xs, ys, xc1, yc1, xc2, yc2, xe, ye, colour	Draws a cubic Bezier curve by specifying the start and end points and two control points. Parameters for the BEZIER command are: xs: X coordinate of start point ys: Y coordinate of start point xc1: X coordinate of first control point yc1: Y coordinate of first control point xc2: X coordinate of second control point yc2: Y coordinate of second control point xe: X coordinate of end point ye: Y coordinate of end point colour: Colour of curve
BLIT ……	The BLIT and SPRITE commands can be used interchangeably. All Micromite Plus functionality is preserved with minor functional changes. See the SPRITE command for functionality over and above the Micromite Plus and command differences.

CAMERA OPEN	Initialises an OV7670 camera ready for use
CAMERA CAPTURE	Captures an image from the camera to a connected 800x480 SSD1963 display. NB: the display must be set to 640 pixel mode using the OPTION LCDPANEL command
CAMERA SAVE “filename”	saves the on-screen image to the SDcard. If the file extension is not specified then ".BMP" is appended.
CAMERA REGISTER register, value	can be used to change the camera settings see the datasheet for details
CAMERA CLOSE	disables the camera and frees the allocated pins
CIRCLE x, y, r [,lw] [, a] [, c] [, fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x, y and r must all be arrays or all be single variables /constants otherwise an error will be generated. lw, a, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
CLOSE [#]nbr [,[#]nbr]	The text “GPS” can be substituted for [#]nbr to close a communications port used for a GPS receiver
DAC n, voltage	Sets the DAC channel (1 or 2) to the voltage requested. This command cannot be used if the DACs are in use for audio output.
DAC START frequency, DAC1array%() [,DAC2array%()]	Sets up the DAC to create an arbitrary waveform. DAC1array%() and optional DAC2array%() should contain numbers in the range 0-4095 to suit the 12-bit DACs. Once started the output continues in the background and control returns to MMBasic. The software automatically and separately uses the number of items in each of the arrays to drive the DACs. The frequency is the rate at which the DACs change value. The maximum frequency is 700KHz. As an example if there are 180 items in the array c%() which are displayed at a frequency of 100,000 Hz this will give a waveform frequency of 100,000/180 = 555Hz. If there are 90 items in the array d%() at the same frequency of 100,000 Hz this will at the same time produce a waveform frequency of 100,000/90 = 1111Hz.
DAC CLOSE	Stops DAC output and returns the DACs to normal use.

FFT signalarray!(), FFTarray!()	Performs a fast fourier transform of the data in “signalarray!”. "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "FFTarray" must be floating point and have dimension 2*N where N is the same as the signal array (e.g. f(1,1023) assuming OPTION BASE is zero) The command will return the FFT as complex numbers with the real part in f(0,n) and the imaginary part in f(1,n)
FFT INVERSE FFTarray!(), signalarray!()	Performs an inverse fast fourier transform of the data in “FFTarray!”. "FFTarray" must be floating point and have dimension 2*N where N must be a power of 2 (e.g. f(1,1023) assuming OPTION BASE is zero) with the real part in f(0,n) and the imaginary part in f(1,n). "signalarray" must be floating point and the single dimension must be the same as the FFT array. The command will return the real part of the inverse transform in "signalarray".
FFT MAGNITUDE signalarray!(),magnitudearray!()	Generates magnitudes for frequencies for the data in “signalarray!” "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "magnitudearray" must be floating point and the size must be the same as the signal array The command will return the magnitude of the signal at various frequencies according to the formula: *frequency at array position N = N * sample_frequency / number_of_samples*
FFT PHASE signalarray!(), phasearray!()	Generates phases for frequencies for the data in “signalarray!”. "signalarray" must be floating point and the size must be a power of 2 (e.g. s(1023) assuming OPTION BASE is zero) "phasearray" must be floating point and the size must be the same as the signal array The command will return the phase angle of the signal at various frequencies according to the formula above.
GUI STARTLINE n	Sets the row in the graphics memory which will appear at the top of the screen (landscape or reverse landscape) or left of the screen (portrait or reverse portrait) for a 4.3” SSD1963 display initialised with OPTION LCDPANEL SSD1963_4P [_16]

I2C2 OPEN speed, timeout [, PU I2C2 WRITE addr, option, sendlen, senddata [,sendata ....] I2C2 READ addr, option, rcvlen, rcvbuf I2C2 CLOSE I2C2 SLAVE OPEN addr, mask, option, send_int, rcv_int I2C2 SLAVE WRITE sendlen, senddata [,sendata ....] I2C2 SLAVE READ rcvlen, rcvbuf, rcvd I2C2 SLAVE CLOSE	See Appendix B of the Micromite User Manual.
LINE x1, y1, x2, y2 [, LW [, C]]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, x2, and y2 must all be arrays or all be single variables /constants otherwise an error will be generated. lw and c can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
LIST COMMANDS	Lists all valid commands
LIST FUNCTIONS	Lists all valid functions and operators
LOAD SPRITE [#]n, fname$	Alternate form. See SPRITE LOAD
LONGSTRING APPEND array%(), string$ LONGSTRING CLEAR array%() LONGSTRING COPY dest%(), src%() LONGSTRING CONCAT dest%(), src%()	Append a normal MMBasic string to a long string variable. array%() is a long string variable while string$ is a normal MMBasic string expression. Will clear the long string variable array%(). ie, it will be set to an empty string. Copy one long string to another. dest%() is the destination variable and src%() is the source variable. Whatever was in dest%() will be overwritten. Concatenate one long string to another. dest%() is the destination variable and src%() is the source variable. src%() will the added to the end of dest%() (the destination will not be overwritten).

LONGSTRING LCASE array%()	Will convert any uppercase characters in array%() to lowercase. array%() must be long string variable.
LONGSTRING LEFT dest%(), src%(), nbr	Will copy the left hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%(). ie, copy from the beginning of src%(). src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING LOAD array%(), nbr, string$	Will copy 'nbr' characters from string$ to the long string variable array%() overwriting whatever was in array%().
LONGSTRING MID dest%(), src%(), start, nbr	Will copy 'nbr' characters from src%() to dest%() starting at character position 'start' overwriting whatever was in dest%(). ie, copy from the middle of src%(). 'nbr' is optional and if omitted the characters from 'start' to the end of the string will be copied src%() and dest%() must be long string variables. 'start' and 'nbr' must be an integer constants or expressions.
LONGSTRING PRINT [#n,] src%()	Prints the longstring stored in src%()
LONGSTRING REPLACE array%() , string$, start	Will substitute characters in the normal MMBasic string string$ into an existing long string array%() starting at position ‘start’ in the long string.
LONGSTRING RIGHT dest%(), src%(), nbr	Will copy the right hand 'nbr' characters from src%() to dest%() overwriting whatever was in dest%(). ie, copy from the end of src%(). src%() and dest%() must be long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING TRIM array%(), nbr	Will trim ‘nbr’ characters from the left of a long string. array%() must be a long string variables. 'nbr' must be an integer constant or expression.
LONGSTRING UCASE array%()	Will convert any lowercase characters in array%() to uppercase. array%() must be long string variable.
OPEN comspec$ AS GPS [,timezone_offset] [,monitor]	Will open a serial communications port for reading from a GPS receiver. See the GPS function for details. The timezone_offset parameter is used to convert UTC as received from the GPS to the local timezone. If omitted the timezone will default to UTC. The timezone_offset can be a any number between -12 and 14 allowing the time to be set correctly even for the Chatham Islands in New Zealand (UTC +12:45). If the monitor parameter is set to 1 then all GPS input is directed to the console. This can be stopped by closing the GPS channel.
OPTION CPU SPEED n	Sets the core CPU speed of the processor. Values between 10MHz and 600MHz are allowed, anything over 480 MHz is overclocking the chip beyond the manufacturers specification.

OPTION AUTOREFRESH mode	When using buffered driver TFT drivers this command controls when screen updates take place. When autorefresh is “ON” updates take place immediately. When autorefresh is “OFF” updates take place in the framebuffer and are only written to the screen when autorefresh is next turned “ON” or the REFRESH command is issued.
OPTION FLASHPAGES n	Sets the number of 128Kbyte pages to be used for storing programs. Valid values are 1 (default) to 4. Increasing the number of pages increases time to store programs and execute “NEW” commands.
OPTION LCDPANEL ILI9341_8, orientation	Selects 8-bit bus operation of the ILI9341 display. Pin usage is exactly as per the SSD1963. Pins SSD1963_DB8 to SSD1963_D15 are allocated but not used. Uses a memory resident framebuffer to improve performance.
OPTION LCDPANEL ILI9341_16, orientation	Selects 16-bit bus operation of the ILI9341 display. Pin usage is exactly as per the SSD1963. Uses a memory resident framebuffer to improve performance.
OPTION LCDPANEL ILI9841, orientation, DCpin, RESETpin, CSpin	Initialises a TFT display using the ILI9841 controller. This supports 480 * 320 resolution. See the Micromite User manual, ILI9341 section, for full details of parameter usage.
OPTION LCDPANEL SSD1963_n_16, orientation	Selects 16-bit bus operation of the various SSD1963 displays. See the Micromite plus manual for full details of syntax.
OPTION LCDPANEL SSD1963_n_8BIT, orientation	Selects 16-bit bus operation of the 5”, 7” and 8” SSD1963 displays using a memory framebuffer with 64 colours for improved performance. This uses 25K of memory leaving 473K for user programs
OPTION LCDPANEL SSD1963_n_BUFF, orientation	Selects 16-bit bus operation of the 5”, 7” and 8” SSD1963 displays using a memory framebuffer for improved performance. This uses 400K of memory leaving 98K for user programs
OPTION LCDPANEL SSD1963_n_640, orientation	Selects 16-bit bus operation of the 5” ,7” and 8” SSD1963 displays with a usable display width of 640 pixels using a memory framebuffer for improved performance. This uses 250K of memory leaving 248K for user programs. This driver must be selected in order to use the CAMERA commands. It can also be used effectively for game programming using the SPRITE commands and functions
OPTION MILLESECONDS OFF OPTION MILLISECONDS ON	Default. The time$ function returns the time as “HH:MM:SS” The time$ function returns the time as “HH:MM:SS.MMM”

OPTION RTC CALIBRATE n	Used to calibrate the RTC. n specifies the number of extra clock pulses to be "created" or "removed" every 2^20 real clock pulses. n can be between -511 and + 512. A change of +/-1 should equate to about 0.0824 seconds per day.
OPTION SERIAL PULLUP DISABLE OPTION SERIAL PULLUP ENABLE	permanently stored option that disables pullups on all serial ports default: permanently stored option that enables pullups on all serial ports
OPTION USBKEYBOARD layout [, powerpinno]	Sets the key layout for a connected USB keyboard. Valid layouts are UK and US. The optional “powerpinno” parameter specifies a pin which can be used to enable power to the USB port. Normally the pin will be set high when enabled. However, by specifying the pin number as a negative value the pin will be set low.
OPTION VCC voltage	This allows the user to precisely set the supply voltage to the chip and is used in the calculation of voltages when using analogue inputs. The parameter is not saved and should be initialised either on the command line or in a program.
PIXEL x, y [,c]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x and y must both be arrays or both be single variables /constants otherwise an error will be generated. c can be either an arrays or single variable/constant. See the Micromite User manual for full details of parameter usage.
PLAY FLAC file$ [, interrupt]	Will play a FLAC file on the DAC outputs. Supported frequencies are: 44100Hz 16-bit(CD quality) and 24-bit 48000Hz 16-bit and 24-bit 88200Hz 16-bit and 24-bit 96000Hz 24-bit 192000Hz 24-bit 'file$' is the FLAC file to play (the extension of .flac will be appended if missing). The FLAC file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing.
PLAY MP3 file$ [, interrupt]	Will play a MP3 file on the DAC outputs. 'file$' is the MP3 file to play (the extension of .mp3 will be appended if missing). The MP3 file is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the file has finished playing.

POLYGON xarray%(), yarray%() [, bordercolour] [, fillcolour]	Draws a outline or filled polygon defined by the x,y coordinate pairs in xarray%() and yarray%(). If fill colour is omitted then just the polygon outline is drawn. If bordercolour is omitted then it will default to the current gui foreground colour. The polygon should be closed with the first and last elements the same. The size of the array should exactly match the number of x,y coordinate pairs.
PRINT #GPS, string$	Outputs a NMEA string to an opened GPS device. The string must start with a $ character and end with a * character. The Checksum is calculated automatically by the firmware and is appended to the string together with cr,lf
PWM 1, freq, 1A [,1B] [,1C], [1D] PWM 2, freq, 2A [,2B] [,2C], [2D]	See description of the PWM command in the Micromite User Manual. This command allows the specification of a frequency for extra PWM channels
REFRESH	Flushes the entire memory buffer of the display to the screen. This command can be used with OPTION AUTOREFRESH OFF to control when the screen is updated
RBOX x1, y1, w, h [, r] [,c] [,fill]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, w, and h must all be arrays or all be single variables /constants otherwise an error will be generated. r, c, and fill can be either arrays or single variables/constants. See the Micromite User manual for full details of parameter usage.
SETPIN n, AIN [,bits]	An optional parameter on SETPIN can be used to select how many bits of resolution should be used for ADC conversions. Valid values are 8, 10, 12, 14, and 16. The more bits the slower the conversion. A single conversion takes between 0.127mSec (8-bit) to 0.6mSec (16-bit).
SETPIN 93, CIN SETPIN 93 ,FIN	This enables a high speed frequency counter (tested to 20MHz) Period measurement (SETPIN 93, PIN) is not supported on this pin
SPRITE	The BLIT and SPRITE commands can be used interchangeably
SPRITE x1, y1, x2, y2, w, h	Copies the memory area specified by top right coordinate x1, y1 and of width w and height h to a new location where the top right coordinate is x2, y2.
SPRITE CLOSE [#]n	Closes sprite n and releases all memory resources. Updates the screen (see SPRITE HIDE). Sprites which have been “copied” cannot be closed until all “copies” have been closed
SPRITE CLOSE ALL	Closes all sprites and releases all memory resources. It does not change the screen.
SPRITE COPY [#]n, [#]m, nbr	Makes a copy of sprite “n” to “nbr” of new sprites starting a number “m”. Copied sprites share the same loaded image as

	the original to save memory
SPRITE HIDE [#]n	Removes sprite n from the display and replaces the stored background. To restore a screen to a previous state sprites should be hidden in the opposite order to which they were written "LIFO"
SPRITE INTERRUPT sub	Specifies the name of the subroutine that will be called when a sprite collision occurs. See Appendix C for how to use the function SPRITE to interrogate details of what has collided
SPRITE LOAD [#]n, fname$ [,colour]	Loads the file fname$ as a sprite into buffer number n. The file must be in PNG format RGB888 or RGBA888. If the file extension .PNG is omitted then it will be automatically added. The parameter “colour” specifies the background colour for the sprite. Pixels in the background colour will not overwrite the background when the sprite is displayed. Colour defaults to zero
SPRITE MOVE	Actions a single atomic transaction that re-locates all sprites which have previously had a location change set up using the SPRITE NEXT command. Collisions are detected once all sprites are moved and reported in the same way as from a scroll
SPRITE NEXT [#]n, x, y	Sets the X and Y coordinate of the sprite to be used when the screen is next scrolled or the SPRITE MOVE command is executed. Using SPRITE NEXT rather than SPRITE SHOW allows multiple sprites to be moved as part of the same atomic transaction.
SPRITE NOINTERRUPT	Disables collision interrupts
SPRITE READ [#]n, x , y, w, h	Reads the display area specified by coordinates x and y, width w and height h into buffer number n. If the buffer is already in use and the width and height of the new area are the same as the original then the new command will overwrite the stored area.
SPRITE SCROLLH n [,col]	Scrolls the background and any sprites on layer 0 n pixels to the right. n can be any number between -31 and 31. Sprites on any layer other than zero will remain fixed in position on the screen. By default the scroll wraps the image round. If “col” is specified the colour will replace the area behind the scrolled image
SPRITE SCROLLR x, y, w, h, delta_x, delta_y [,col]	Scrolls the region of the screen defined by top-right coordinates “x” and “y” and width and height “w” and “h” by “delta_x” pixels to the right and “delta_y” pixels up. By default the scroll wraps the background image round. If “col” is specified the colour will replace the area behind the scrolled image. Sprites on any layer other than zero will remain fixed in

	position on the screen. Sprites in layer zero where the centre of the sprite (x+ w/2, y+ h/2) falls within the scrolled region will move with the scroll and wrap round if the centre moves outside one of the boundaries of the scrolled region.
SPRITE SCROLLV n [,col]	Scrolls the background, and any sprites on layer 0, n pixels up. n can be any number between -MM.VRES-1 and MM.VRES-1. Sprites on any layer other than zero will remain fixed in position on the screen. . By default the scroll wraps the image round. If “col” is specified the colour will replace the area behind the scrolled image
SPRITE SHOW [#]n, x, layer, [orientation]	Displays sprite n on the screen with the top left at coordinates x, y. Sprites will only collide with other sprites on the same layer, layer zero, or with the screen edge. If a sprite is already displayed on the screen then the SPRITE SHOW command acts to move the sprite to the new location. The display background is stored as part of the command such that it can be replaced when the sprite is hidden or moved further. The orientation is an optional parameter, valid values are: 0 - normal display (default if omitted) 1 - mirrored left to right 2 - mirrored top to bottom 3 - rotated 180 degrees (= 1+2)
SPRITE WRITE [#]n, x y	Overwrites the display with the contents of sprite buffer n with the top left at coordinates x, y. SPRITE WRITE overwrites the complete area of the display. The background that is overwritten is not stored so SPRITE WRITE is inherently higher performing than SPRITE SHOW but with greater functional limitations.
SERVO 1 [, freq], 1A [,1B] [,1C] [,1D] SERVO 2 [, freq], 2A [,2B] [,2C] [,2D]	See description of the SERVO command in the Micromite User Manual. This command allows the specification of a frequency for extra channels
SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]	Calculates pitch, roll and yaw angles from accelerometer and magnetometer inputs. Valid fusion types are MAHONY and MADGWICK. Usage is described in Appendix A

SPI3 OPEN speed, mode, bits SPI READ nbr, array() SPI WRITE nbr, data1, data2, data3, … etc or SPI WRITE nbr, string$ or SPI WRITE nbr, array() SPI CLOSE	See Appendix D of the Micromite User Manual.
TRIANGLE X1, Y1, X2, Y2, X3, Y3 [, C [, FILL]]	All parameters can now be expressed as arrays and the software will plot the number of boxes as determined by the dimensions of the smallest array. x1, y1, x2, y2, x3,and y3 must all be arrays or all be single variables /constants otherwise an error will be generated c and fill can be either arrays or single variables/constants. See the Micromite Plus manual for full details of parameter usage.
TTS [PHONETIC] "text" [,speed] [,pitch] [,mouth] [,throat] [, interrupt]	Outputs text as speech on the DAC outputs. See http://www.retrobits.net/atari/sam.shtml for details of parameter usage. The speech is played in the background. 'interrupt' is optional and is the name of a subroutine which will be called when the speech has finished playing.

TURTLE RESET	Clears the screen and re-initialises the turtle system. The turtle is located at MM.HRES\2, MM.VRES\2. The default pen colour is white and the fill colour is green. The initial heading is up (0 degrees)
TURTLE DRAW TURTLE	Draws a turtle at the current location and in the current orientation
TURTLE PEN UP	Lifts the pen so moves do not write to the screen
TURTLE PEN DOWN	Lowers the pen so moves do write to the screen
TURTLE FORWARD n	Moves the turtle n pixels in the current heading
TURTLE BACKWARD n	Moves the turtle n pixels opposite the current heading
TURTLE DOT	Draw a 1-pixel dot at the current location, regardless of pen status
TURTLE TURN LEFT deg	Turn the turtle to the left (anti-clockwise) by the specified number of degrees.
TURTLE TURN RIGHT deg	Turn the turtle to the right (clockwise) by the specified number of degrees.
TURTLE BEGIN FILL	Start filling. Call this before drawing a polygon to activate the bookkeeping required to run the filling algorithm later.
TURTLE END FILL	End filling. Call this after drawing a polygon to trigger the fill algorithm. The filled polygon may have up to 128 sides.
TURTLE HEADING deg	Rotate the turtle to the given absolute heading (in degrees). 0 degrees means facing straight up. 90 degrees means facing to the right.
TURTLE PEN COLOUR col	Set the current drawing colour. Colours are specified as per normal Micromite drawing commands
TURTLE FILL COLOUR col	Set the current fill colour. Colours are specified as per normal Micromite drawing commands
TURTLE MOVE x, y	Move the turtle to the specified location, drawing a straight line if the pen is down.

TURTLE DRAW PIXEL x, y	Draw a 1-pixel dot at the given location using the current draw colour, regardless of current turtle location or pen status.
TURTLE FILL PIXEL x, y	Draw a 1-pixel dot at the given location using the current fill colour, regardless of current turtle location or pen status.
TURTLE DRAW LINE x1, y1, x2, y2	Draw a straight line between the given coordinates, regardless of current turtle location or pen status.
TURTLE DRAW DIRCLE x, y, r	Draw a circle at the given coordinates with the given radius, regardless of current turtle location or pen status.

WS2812 pinno, colours%()	This command outputs the required signals needed to drive WS2812 LED drivers on the pin specified. The colours%() array should be sized to have exactly the same number of elements as the number of LEDs to be driven. Each element in the array should contain the colour in the normal RGB888 format (0 - &HFFFF) e.g.
	`dim b%(6)=(rgb(red), rgb(green), rgb(blue), rgb(Yellow), rgb(cyan), rgb(magenta), rgb(white))`
	`setpin 1,dout`
	`ws2812 1,b%()`
	will output the specified colours to an array of 7 WS2812 LEDs

Functions (Armmite H7 Only)

BIN2STR$(type, value [,BIG])	This function returns a string containing the binary representation of 'value'. Valid values of "type" are: INT64: returns an 8 byte string containing a signed 64-bit integer UINT64: returns an 8 byte string containing a unsigned 64-bit integer INT32: returns a 4 byte string containing a signed 32-bit integer UINT32: returns a 4 byte string containing an unsigned 32-bit integer INT16: returns a 2 byte string containing a signed 16-bit integer UINT16: returns a 2 byte string containing an unsigned 16-bit integer INT8: returns a 1 byte string containing a signed 8-bit integer UINT8: returns a 1 byte string containing an unsigned 8-bit integer SINGLE: returns a 4 byte string containing a single precision floating point number DOUBLE: returns an 8 byte string containing a double precision floating point number By default the string contains the number in little-endian format. i.e. the least significant byte is the first one in the string. Setting the third parameter to 'BIG' will return the string in big-endian format i.e. the most significant byte is the first one in the string In the case of the integer conversions, an error will be generated if the 'value' cannot fit into the 'type' e.g. an attempt to store the value 400 in a INT8 This function makes it easy to prepare data for efficient binary file I/O or for preparing numbers for output to sensors and saving to flash memory. See also the function STR2BIN
DATETIME$(n)	Returns the date and time corresponding to the epoch number n (number of seconds that have elapsed since midnight GMT on January 1, 1970). The format of the returned string is "dd-mm-yyyy hh:mm:ss". Use the text NOW to get the current datetime string, i.e. ? DATETIME$(NOW)
DAY$(date$)	Returns the day of the week for a given date as a string "Monday", "Tuesday" etc. The format for date$ is "dd-mm-yyyy". Use NOW to get the day for the current date, e.g. ? DAY$(NOW)
EPOCH(DATETIME$)	Returns the the epoch number (number of seconds that have elapsed since midnight GMT on January 1, 1970) for the supplied DATETIME$ string. The format for DATETIME$ is "dd-mm-yyyy hh:mm:ss". Use NOW to get the epoch number for the current date and time, i.e. ? EPOCH(NOW)
GPS(ALTITUDE) GPS(DATE)	returns current altitude if sentence GGA enabled returns the normal date string corrected for local time e.g. "12-

	01-2017”
GPS(DOP)	returns DOP (dilution of precision) value if sentence GGA enabled
GPS(FIX)	returns 0=no fix, 1=fix, etc. if sentence GGA enabled
GPS(GEOID)	Returns the geoid-ellipsoid separation. if sentence GGA enabled
GPS(LATITUDE)	returns the latitude in degrees as a floating point number, values are –ve for South of equator
GPS LONGITUDE)	returns the longitude in degrees as a floating point number, values are –ve for West of the meridian
GPS(SATELLITES)	returns number of satellites in view if sentence GGA enabled
GPS(SPEED)	returns the ground speed in knots as a floating point number
GPS(TIME)	returns the normal time string corrected for local time e.g. “12:09:33”
GPS(TRACK)	returns the track over the ground (degrees true) as a floating point number
GPS(VALID)	returns: 0=invalid data, 1=valid data. **ALWAYS CHECK THIS VALUE TO ENSURE DATA IS VALID BEFORE USING OTHER GPS() FUNCTION CALLS**
JSON$(array%(),string$)	Returns a string representing a specific item out of the JSON input stored in the longstring array%() e.g. JSON$(a%(), “name”) JSON$(a%(), “coord.lat”) JSON$(a%(), “weather[0].description”) JSON$(a%(),”list[4].weather[0].description Examples taken from api.openweathermap.org
LCOMPARE(array1%(), array2%())	Compare the contents of two long string variables array1%() and array2%(). The returned is an integer and will be -1 if array1%() is less than array2%(). It will be zero if they are equal in length and content and +1 if array1%() is greater than array2%(). The comparison uses the ASCII character set and is case sensitive.
LGETSTR$(array%(), start, length)	Returns part of a long string stored in array%() as a normal MMBasic string. The parameters start and length define the part of the string to be returned.

LINSTR(array%(), search$ [,start])	Returns the position of a search string in a long string. The returned value is an integer and will be zero if the substring cannot be found. array%() is the string to be searched and must be a long string variable. Search$ is the substring to look for and it must be a normal MMBasic string or expression (not a long string). The search is case sensitive. Normally the search will start at the first character in 'str' but the optional third parameter allows the start position of the search to be specified.
LIST COMMANDS	Lists all valid commands
LIST FUNCTIONS	Lists all valid functions and operators
LLEN(array%())	Returns the length of a long string stored in array%()
MM.DEVICE$	Returns "Armmite "
MM.INFO$(AUTORUN)	Returns “On” or “Off” depending on the status of OPTION AUTORUN
MM.INFO$(CPUSPEED)	Returns the CPU speed as a string
MM.INFO$(LCDPANEL)	Returns the name of the LCD panel configured or a blank string
MM.INFO$(PIN pinno)	Returns the status of I/O pin “pinno”. Valid returns are: “Invalid”, “Reserved”, “In Use”, and “Unused”
MM.INFO$(SDCARD)	Returns the status of the SDcard. Valid returns are: “Disabled”, “Not present”, “Ready, and “Unused”
MM.INFO$(TOUCH)	Returns the status of the Touch controller. Valid returns are: “Disabled”, “Not calibrated”, and “Ready”
MM.PERSIST	Returns the value of a user variable that is not impacted by CPU RESTART or a watchdog timeout
MOVEMENT(sensitivity)	This function combines capturing a new image from an OV7670 camera with a comparison with the currently displayed image. It returns the number of pixels that are different between the images. The sensitivity parameter controls whether or not a pixel is considered to be different. The algorithm works by converting each image to a 6-bit greyscale image. The sensitivity is then the difference between the pixels that will be counted as a change. The top 20 and bottom 20 rows are ignored in the comparison. This allows the user to use these for other display purposes without artificially triggering the movement detection.
OWSEARCH(pin, flag [,serial_number_mask])	Returns the onewire device serial number as an INTEGER. Flags are:

	0 - Continue an existing search 1 - start a new search 4 - Continue an existing search for devices in the requested family 5 - start a new search for devices in the requested family (the MSB of the serial_number_mask) 8 - skip the current device family and return the next device 16 - verify that the device with the serial number in serial_number_mask is available
SPI3(n)	See Appendix D of the Micromite User Manual
SPRITE(…)	
SPRITE(C, [#]n)	Returns the number of currently active collisions for sprite n. If n=0 then returns the number of sprites that have a currently active collision following a SPRITE SCROLL command
SPRITE(C, [#]n, m)	Returns the number of the sprite which caused the “m”th collision of sprite n. If n=0 then returns the sprite number of “m”th sprite that has a currently active collision following a SPRITE SCROLL command
SPRITE(H,[#]n)	Returns the height of sprite n. This function is active whether or not the sprite is currently displayed (active).
SPRITE(L, [#]n)	Returns the layer number of active sprites number n
SPRITE(N)	Returns the number of displayed (active) sprites
SPRITE(N,n)	Returns the number of displayed (active) sprites on layer n
SPRITE(S)	Returns the number of the sprite which last caused a collision. NB if the number returned is Zero then the collision is the result of a SPRITE SCROLL command and the SPRITE(C…) function should be used to find how many and which sprites collided.
SPRITE(W, [#]n)	Returns the width of sprite n. This function is active whether or not the sprite is currently displayed (active).
SPRITE(X, [#]n)	Returns the X-coordinate of sprite n. This function is only active when the sprite is currently displayed (active). Returns 10000 otherwise.
SPRITE(Y, [#]n)	Returns the Y-coordinate of sprite n. This function is only active when the sprite is currently displayed (active). Returns 10000 otherwise

STR2BIN(type, string$ [,BIG])	This function returns a number equal to the binary representation in 'string$'. Valid values of 'type' are: INT64: takes an 8 byte string containing a signed 64-bit integer, returns an MMBasic INTEGER UINT64: takes an 8 byte string containing an unsigned 64-bit integer, returns an MMBasic INTEGER INT32: takes a 4 byte string containing a signed 32-bit integer, returns an MMBasic INTEGER UINT32: takes a 4 byte string containing an unsigned 32-bit integer, returns an MMBasic INTEGER INT16: takes a 2 byte string containing a signed 16-bit integer, returns an MMBasic INTEGER UINT16: takes a 2 byte string containing an unsigned 16-bit integer, returns an MMBasic INTEGER INT8: takes a 1 byte string containing a signed 8-bit integer, returns an MMBasic INTEGER UINT8: takes a 1 byte string containing an unsigned 8-bit integer, returns an MMBasic INTEGER SINGLE: takes an 4 byte string containing single precision floating point number, returns an MMBasic FLOAT DOUBLE: takes an 8 byte string containing single precision floating point number, returns an MMBasic FLOAT By default the string must contain the number in little-endian format. i.e. the least significant byte is the first one in the string. Setting the third parameter to 'BIG' will interpret the string in big-endian format i.e. the most significant byte is the first one in the string. This function makes it easy to read data from binary data files or interpret numbers from sensors or efficiently read binary data from flash memory chips. An error will be generated if the string is the incorrect length for the conversion requested See also the function BIN2STR$

Appendix A
Sensor Fusion

The Armmite H7 supports the calculation of pitch, roll and yaw angles from accelerometer and magnetometer inputs.

For information on this technology see https://github.com/kriswiner/MPU-6050/wiki/Affordable-9-DoF-Sensor-Fusion

The SENSORFUSION command supports both the MADGWICK and MAHONY fusion algorithms. The format of the command is:

SENSORFUSION type ax, ay, az, gx, gy, gz, mx, my, mz, pitch, roll, yaw [,p1] [,p2]

Type can be MAHONY or MADGWICK

Ax, ay, and az are the accelerations in the three directions and should be specified in units of standard gravitational acceleration.

Gx, gy, and gz are the instantaneous values of rotational speed which should be specified in radians per second.

Mx, my, and mz are the magnetic fields in the three directions and should be specified in nano-Tesla (nT)

Care must be taken to ensure that the x, y and z components are consistent between the three inputs. So , for example, using the MPU-9250 the correct input will be ax, ay,az, gx, gy, gz, **my, mx, -mz** based on the reading from the sensor.

Pitch, roll and yaw should be floating point variables and will contain the outputs from the sensor fusion.

The SENSORFUSION routine will automatically measure the time between consecutive calls and will use this in its internal calculations.

The Madwick algorithm takes an optional parameter p1. This is used as beta in the calculation. It defaults to 0.5 if not specified

The Mahony algorithm takes two optional parameters p1, and p2. These are used as Kp and Ki in the calculation. If not specified these default to 10.0 and 0.0 respectively.

A fully worked example of using the code is given on the BackShed forum at

http://www.thebackshed.com/forum/forum_posts.asp?TID=9321&PN=1&TPN=1

Appendix B
Sprites

See the SPRITE commands and functions for syntax details.
The concept of the sprite implementation is as follows:

1. Sprites are full colour and of any size. The collision boundary is the enclosing rectangle.
2. Sprites are loaded to a specific number (1-50)
3. Sprites are displayed using the SPRITE SHOW command
4. For each SHOW command the user must select a "layer". This can be between 0 and 10.
5. Sprites collide with sprites on the same layer, layer 0, or the screen edge
6. Layer 0 is a special case and sprites on all other layers will collide with it
7. The SCROLL commands leave sprites on all layers except layer 0 unmoved
8. Layer 0 sprites scroll with the background and this can cause collisions
9. There is no practical limit on the number of collisions caused by SHOW or SCROLL commands
10. The sprite function allows the user to fully interrogate the details of a collision
11. A SHOW command will overwrite the details of any previous collisions for that sprite
12. A SCROLL command will overwrite details of previous collisions for ALL sprites
13. To restore a screen to a previous state sprites should be removed in the opposite order to which they were written "LIFO"

Because moving a sprite or, particularly, scrolling the background can cause multiple sprite collisions it is important to understand how they can be interrogated.

The best way to deal with a sprite collision is using the interrupt facility. A collision interrupt routine is set up using the SPRITE INTERRUPT command.

e.g. SPRITE INTERRUPT collision

The following is a pro-forma for identifying all collisions that have resulted from either a SPRITE SHOW command or a SCROLL command

```
'
' This routine demonstrates a complete interrogation of collisions
'
sub collision
  local integer i
' First use the SPRITE(S) function to see what caused the interrupt
  if sprite(S) <> 0 then 'collision of specific individual sprite
    'sprite(S) returns the sprite that moved to cause the collision
          print "Collision on sprite ",sprite(S)
          process_collision(sprite(S))
          print ""
'
  else '0 means collision of one or more sprites caused by background move
```

```
    ' SPRITE(C, 0) will tell us how many sprites had a collision
    print "Scroll caused a total of ",sprite(C,0)," sprites to have collisions"
    for i=1 to sprite(C,0)
          ' SPRITE(C, 0, i) will tell us the sprite number of the “I”th sprite
      print "Sprite ",sprite(C,0,i)
      process_collision(sprite(C,0,i))
    next i
    print ""
  endif
end sub
' get details of the specific collisions for a given sprite
sub process_collision(S as integer)
  local integer i ,j
  'sprite(C, #n) returns the number of current collisions for sprite n
  print "Total of ",sprite(C,S)," collisions"
   for i=1 to sprite(C,S)
          ' SPRITE(C, S, i) will tell us the sprite number of the “I”th sprite
    j=sprite(C,S,i)
    if j=100 then
      print "collision with left of screen"
    else if j=101 then
      print "collision with top of screen"
    else if j=102 then
      print "collision with right of screen"
    else if j=103 then
      print "collision with bottom of screen"
    else
      print "Collision with sprite ",sprite(C,S,i) 'sprite(C, #n, #m) returns details of the mth collision
    endif
  next i
end sub
```

Appendix C
Nucleo Pin Definitions

The following provides constants to access the Nucleo pins by a variety of relevant naming conventions

```
Const PE2=1,D56=1,CN9_14=1,CN11_46=1
Const PE3=2,D60=2,CN9_22=2,CN11_47=2
Const PE4=3,D57=3,CN9_16=3,CN11_48=3
Const PE5=4,D58=4,CN9_18=4,CN11_50=4
Const PE6=5,D59=5,CN9_20=5,CN11_62=5
Const PC13=7,CN11_23=7
Const PC14=8,CN11_25=8
Const PC15=9,CN11_27=9
Const PF0=10,D68=10,CN9_21=10,CN11_53=10
Const PF1=11,D69=11,CN9_19=11,CN11_51=11
Const PF2=12,D70=12,CN9_17=12,CN11_52=12
Const PF3=13,A3=13,CN9_7=13,CN12_58=13
Const PF4=14,A8=14,CN10_11=14,CN12_38=14
Const PF5=15,A4=15,CN9_9=15,CN12_36=15
Const PF6=18,CN11_9=18
Const PF7=19,D62=19,CN9_26=19,CN11_11=19
Const PF8=20,CN11_54=20
Const PF9=21,D63=21,CN9_28=21,CN11_56=21
Const PF10=22,A5=22,CN9_11=22,CN12_42=22
Const PH0=23,CN11_29=23
Const PH1=24,CN11_31=24
Const PC0=26,A1=26,CN9_3=26,CN11_38=26
Const PC1=27,CN11_36=27
Const PC2=28,A7=28,CN10_9=28,CN11_35=28
Const PC3=29,A2=29,CN9_5=29,CN11_37=29
Const PA0=34,D32=34,CN10_29=34,CN11_28=34
Const PA1=35,CN11_30=35
Const PA2=36,CN12_35=36
Const PA3=37,A0=37,CN9_1=37,CN12_37=37
Const PA4=40,D24=40,CN7_17=40,CN11_32=40
Const PA5=41,D13=41,CN7_10=41,CN12_11=41
Const PA6=42,D12=42,CN7_12=42,CN12_13=42
Const PA7=43,D11=43,CN7_14=43,CN12_15=43
Const PC4=44,CN12_34=44
Const PC5=45,CN12_6=45
Const PB0=46,D33=46,CN10_31=46,CN11_34=46
Const PB1=47,A6=47,CN10_7=47,CN12_24=47
Const PB2=48,D27=48,CN10_15=48,CN12_22=48
Const PF11=49,CN12_62=49
Const PF12=50,D8=50,CN7_20=50,CN12_59=50
Const PF13=53,D7=53,CN10_2=53,CN12_57=53
```

```
Const PF14=54,D4=54,CN10_8=54,CN12_50=54
Const PF15=55,CN12_60=55
Const PG0=56,CN11_59=56
Const PG1=57,D64=57,CN9_30=57,CN11_58=57
Const PE7=58,D41=58,CN10_20=58,CN12_44=58
Const PE8=59,D42=59,CN10_18=59,CN12_40=59
Const PE9=60,D6=60,CN10_4=60,CN12_52=60
Const PE10=63,D40=63,CN10_24=63,CN12_47=63
Const PE11=64,D5=64,CN10_6=64,CN12_56=64
Const PE12=65,D39=65,CN10_26=65,CN12_49=65
Const PE13=66,D3=66,CN10_12=66,CN12_55=66
Const PE14=67,D38=67,CN10_28=67,CN12_51=67
Const PE15=68,CN12_53=68
Const PB10=69,D36=69,CN10_32=69,CN12_25=69
Const PB11=70,D35=70,CN10_34=70,CN12_18=70
Const PB12=73,D19=73,CN7_7=73,CN12_16=73
Const PB13=74,D18=74,CN7_5=74,CN12_30=74
Const PB14=75,CN12_28=75
Const PB15=76,D17=76,CN7_3=76,CN12_26=76
Const PD8=77,CN12_10=77
Const PD9=78,CN11_69=78
Const PD10=79,CN12_65=79
Const PD11=80,D30=80,CN10_23=80,CN12_45=80
Const PD12=81,D29=81,CN10_21=81,CN12_43=81
Const PD13=82,D28=82,CN10_19=82,CN12_41=82
Const PD14=85,D10=85,CN7_16=85,CN12_46=85
Const PD15=86,D9=86,CN7_18=86,CN12_48=86
Const PG2=87,D49=87,CN8_14=87,CN11_42=87
Const PG3=88,D50=88,CN8_16=88,CN11_44=88
Const PG4=89,CN12_69=89
Const PG5=90,CN12_68=90
Const PG6=91,CN12_70=91
Const PG7=92,CN12_67=92
Const PG8=93,CN12_66=93
Const PC6=96,D16=96,CN7_1=96,CN12_4=96
Const PC7=97,D21=97,CN7_11=97,CN12_19=97
Const PC8=98,D43=98,CN8_2=98,CN12_2=98
Const PC9=99,D44=99,CN8_4=99,CN12_1=99
Const PA8=100,CN12_23=100
Const PA9=101,CN12_21=101
Const PA10=102,CN12_33=102
Const PA11=103,CN12_14=103
Const PA12=104,CN12_12=104
Const PA13=105,CN11_13=105
Const PA14=109,CN11_15=109
Const PA15=110,D20=110,CN7_9=110,CN11_17=110
Const PC10=111,D45=111,CN8_6=111,CN11_1=111
```

```
Const PC11=112,D46=112,CN8_8=112,CN11_2=112
Const PC12=113,D47=113,CN8_10=113,CN11_3=113
Const PD0=114,D67=114,CN9_25=114,CN11_57=114
Const PD1=115,D66=115,CN9_27=115,CN11_55=115
Const PD2=116,D48=116,CN8_12=116,CN11_4=116
Const PD3=117,D55=117,CN9_10=117,CN11_40=117
Const PD4=118,D54=118,CN9_8=118,CN11_39=118
Const PD5=119,D53=119,CN9_6=119,CN11_41=119
Const PD6=122,D52=122,CN9_4=122,CN11_43=122
Const PD7=123,D51=123,CN9_2=123,CN11_45=123
Const PG9=124,D0=124,CN10_16=124,CN11_63=124
Const PG10=125,CN11_66=125
Const PG11=126,CN11_70=126
Const PG12=127,CN11_65=127
Const PG13=128,CN11_68=128
Const PG14=129,D1=129,CN10_14=129,CN12_61=129
Const PG15=132,CN11_64=132
Const PB3=133,D23=133,CN7_15=133,CN12_31=133
Const PB4=134,D25=134,CN7_19=134,CN12_27=134
Const PB5=135,D22=135,CN7_13=135,CN12_29=135
Const PB6=136,D26=136,CN10_13=136,CN12_17=136
Const PB7=137,CN11_21=137
Const PB8=139,D15=139,CN7_2=139,CN12_3=139
Const PB9=140,D14=140,CN7_4=140,CN12_5=140
Const PE0=141,D34=141,CN10_33=141,CN12_64=141
Const PE1=142,CN11_61=142
```