I2C LCD module 20 x 4

Connections

1	GND
2	VCC (+5V)
3	SDA
4	SCL

The LCD has a Funduino LCM1602 soldered on the reverse side.

The I2C bus address can be set using 3 jumpers (111 means open all three jumpers)

```
0x20 000 connect all
0x21 001
0x22 010
0x23 011
0x24 100
0x25 101
0x26 110
0x27 111 open all
```

Default as delivered is all jumpers open or 111 meaning address is 0x27.

Library to use is the `LiquidCrystal_I2C.h` (see Hobby Components forum for download). The main functions are:

```
LiquidCrystal_I2C lcd(address(hex), cols, rows); // create lcd object
```

```
lcd.init(); // initialise the display
lcd.backlight(); // turn on backlight, do this is loop()
lcd.clear(); // clear the display, cursor 0,0
lcd.home(); // cursor to 0,0. Leave displayed charaters
lcd.setCursor(col, row); // rows 0 & 1, cols 0 - 15
lcd.print(value); // decimal or string
lcd.write(value); // raw write of value
```

I2C connections on Arduino

```
A4 = SDA
A5 = SCL
```

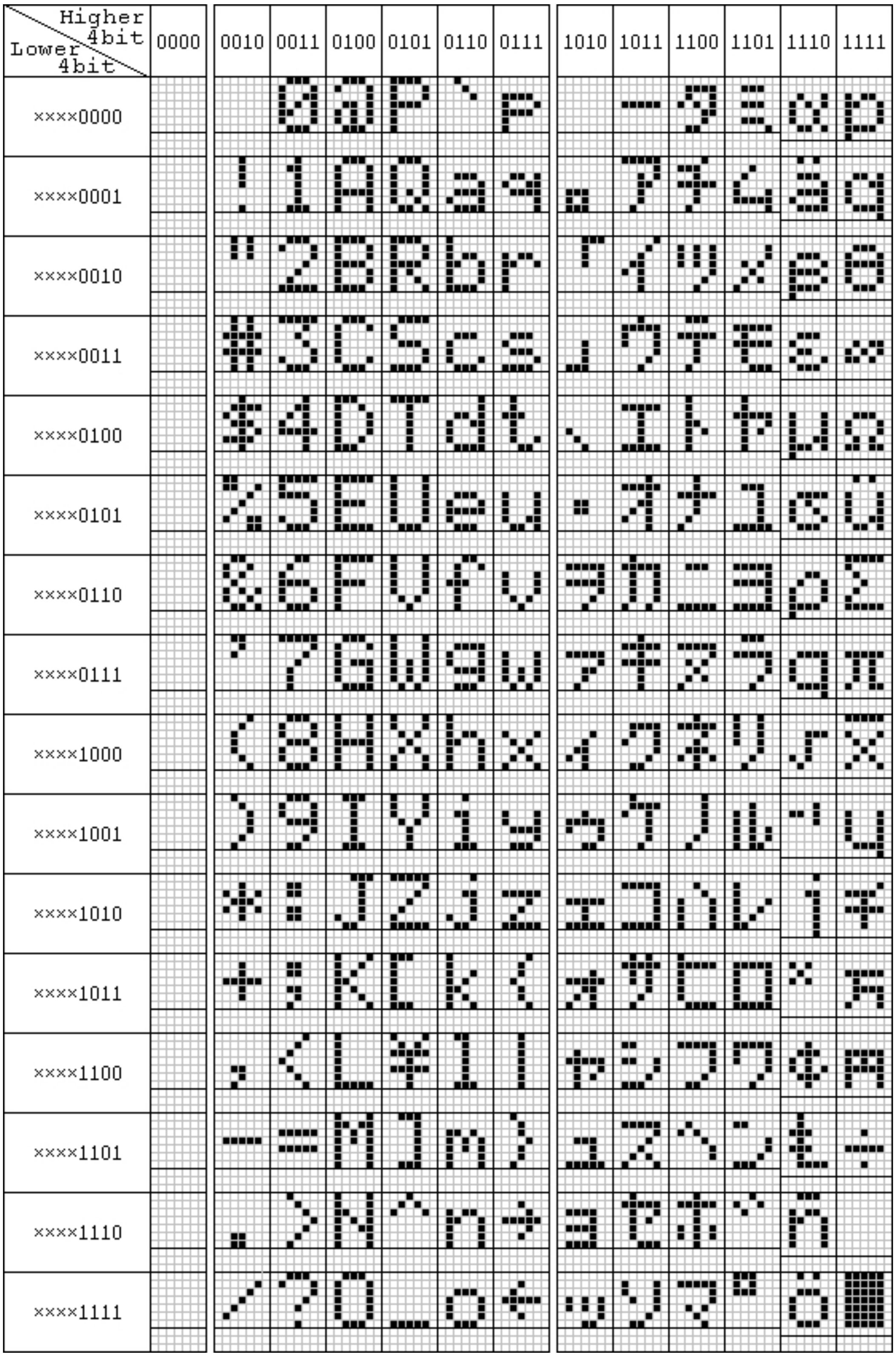
Code

Make SURE you get the LiquidCrystal_I2C library from Hobbycomponents. Many of the others on the web have been modified and DON’T work.

```
/* Include the SPI/IIC Library */
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

/* Initialise the LiquidCrystal library. The default address is 0x27
   and this is a 16 x 4 line display */
LiquidCrystal_I2C lcd(0x27,16,2);


void setup()
{
  /* Initialise the LCD */
  lcd.init();
}

/* Main program loop */
void loop()
{
  /* Make sure the backlight is turned on */
  lcd.backlight();

  /* Display some text inside the border */
  while (1)
  {
    lcd.setCursor(0,0);
    lcd.print(“GANYMEDEHAM”);
    lcd.setCursor(0,1);
    lcd.print(“M6KWH”);
    delay(500);
    delay(500);
  }
}
```

LCD Character Set

The character set for the LCD module is shown in the following figure



Higher 4bit / Lower 4bit	0000	0010	0011	0100	0101	0110	0111	1010	1011	1100	1101	1110	1111
xxxx0000			0	@	P	`	p		ー	タ	ミ	α	p
xxxx0001		!	1	A	Q	a	q	。	ア	チ	ム	ä	q
xxxx0010		"	2	B	R	b	r	「	イ	ツ	メ	β	θ
xxxx0011		#	3	C	S	c	s	」	ウ	テ	モ	ε	∞
xxxx0100		$	4	D	T	d	t	、	エ	ト	ヤ	μ	Ω
xxxx0101		%	5	E	U	e	u	・	オ	ナ	ユ	σ	ü
xxxx0110		&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111		'	7	G	W	g	w	ァ	キ	ヌ	ラ	g	π
xxxx1000		(	8	H	X	h	x	ィ	ク	ネ	リ	√	x̄
xxxx1001		)	9	I	Y	i	y	ゥ	ケ	ノ	ル	⁻¹	y
xxxx1010		*	:	J	Z	j	z	ェ	コ	ハ	レ	j	千
xxxx1011		+	;	K	[	k	{	ォ	サ	ヒ	ロ	ˣ	万
xxxx1100		,	<	L	¥	l	|	ャ	シ	フ	ワ	¢	円
xxxx1101		-	=	M	]	m	}	ュ	ス	ヘ	ン	£	÷
xxxx1110		.	>	N	^	n	→	ョ	セ	ホ	゛	ñ	
xxxx1111		/	?	O	_	o	←	ッ	ソ	マ	゜	ö	█

Custom Characters

The display module has 64 bytes of character generator RAM, which provide 8 user-definable characters. These characters correspond to character codes 00h through 07h.

Character codes 08h through 0Fh also correspond to the eight user-defined characters. Each user-defined character is a 5X8 character in an 8X8 block. The mapping between the CG RAM locations and the displayed character is shown in the following figure. If the 5X10 dot display arrangement is used, only 4 user-defined characters are supported and the characters are defined with 11 bytes each.

```
CG Address      CG Data
                D7-----------D0
x+0             . . . 1 1 1 1 1         *****
x+1             . . . 1 0 0 0 0         *
x+2             . . . 1 0 0 0 1         *   *
x+3             . . . 1 1 1 1 1    =    *****
x+4             . . . 1 0 0 0 1         *   *
x+5             . . . 1 0 0 0 0         *
x+6             . . . 1 0 0 0 0         *
x+7             . . . 0 0 0 0 0         (cursor line)
```

Where "x" is the base address for the character: CG0=00h, CG1=08h, CG2=10h, CG3=18h, CG4=20h, CG5=28h, CG6=30h, CG7=38h.
As you can see in the above figure, each byte of CG RAM represents 1 row of pixels in the character, with D0 being the rightmost pixel, and D4 being the leftmost (the upper 3 bits are not displayed). The first address for a character is the topmost row, while the 7th is the bottom. (The 8th is ORed with the cursor underbar). A `1' in a pixel position becomes a dark pixel on the display. CG RAM occupies a separate address space from the data display (DD) RAM. One must set the (first) CG address before writing any CG data. Each write automatically increments/decrements the CG address pointer to the next location. Remember to set the display back to DD addressing mode before trying to write characters to be displayed.
As an example, we'll define character #3 to be the above symbol by writing the CG address to the start of the character's RAM then writing the successive rows of pixel data.

```
RS R/W D7----D0
0  0   01011000         Set CG RAM address to char #3
1  0   00011111         Write top row of pixels
1  0   00010000                :
1  0   00010001                :
1  0   00011111                :
1  0   00010001                :
1  0   00010000                :
1  0   00010000         Write bottom row of pixels
1  0   00000000         Write cursor row (descender)
```