UNI/O Serial EEPROM Driver

The Microchip 11XX series of EEPROMs are perfectly suited to recording small amounts of data in a Micromite based project. They are available in a range of capacities up to 2 Kbytes and come in an easy to use TO-92 package (a thru hole package that looks like a standard transistor) with only one pin used for communicating with the Micromite.

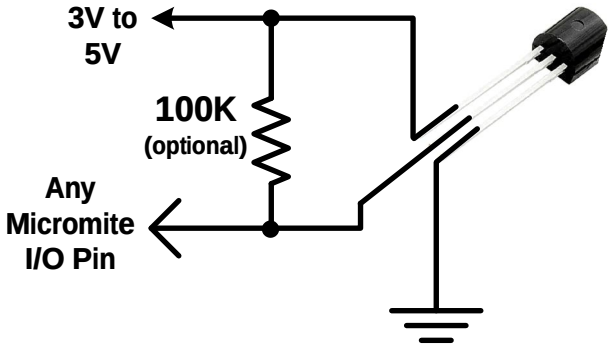
They draw very little current (1µA when not being accessed) and any memory location can be rewritten up to a million times. This last characteristic means that it is possible to log a value (for example, a temperature) once every few seconds in sequential memory locations and the EEPROM would not exceed its endurance specification even after a hundred years.

The 11XX series is also low cost, about 20 to 30 cents for a single unit direct from Microchip.

Typical part numbers that work well with the Micromite are the **11AA160-I/TO** (16Kbit TO-92 package) or the **11AA160T-I/TT** (16Kbit SOT-23 package). For more details visit the Microchip website: http://www.microchip.com/wwwproducts/en/11AA160

Connections

The 11XX EEPROM will work off 3.3V or 5V and can be directly connected to any I/O pin on the Micromite. A pullup resistor can be used to reduce noise and errors on the signal line but normally is not required. Note that if the device is powered from 5V then a 5V tolerant I/O pin must be used.



The 11XX series are sensitive to noise on the power and signal pins so clean power is important (with decoupling capacitors) and the signal line should be short and routed away from other signal lines.

The 16 Kbit version comes in two types with different UNI/O addresses and they can be connected in parallel using the same I/O pin. Depending on the command parameter supplied to the UNIO driver either can be accessed on demand. Any number of UNIO EEPROMs can be connected using other I/O pins to expand the capacity at low cost.

Usage

To add the UNIO command to MMBasic you need to insert the code contained in UNIO.BAS somewhere in your BASIC program. The exact spot is not important but at the end of the program is normal.

This will work on the Micromite (28 or 44 pins) and the Micromite Plus (64 or 100 pins). The CPU speed must be 20MHz or higher and the operation can take from 5 to 200 milliseconds (depending on the amount of data read/written) to complete.

The function takes five parameters and returns a status value.

```
status = UNIO(pin, cmd, addr, nbr, data)
```

The parameters are:

PIN	The pin number used to access the chip.
CMD	The command. This consists of two decimal numbers described below.
ADDR	The address of the first byte to be read/written (addresses in any one EEPROM start at zero and range up to 2047 (for the 16 Kbit version).
NBR	Number of bytes to read/write (described further below).
DATA	The variable holding data to be read/written. This can be a variable or an array. In the case of a write it can also be an expression.

The status value returned by the function will be 1 (or true) for success or 0 (false) if an error occurred. Errors can be caused by too low a CPU speed, no EEPROM detected, noise on the signal line, etc. The 11XX series of EEPROMs are sensitive to noise on the power supply or signal lines so if the UNIO function returns zero you could retry the operation a number of times until you receive a true (indicating success) return. Also try reading/writing smaller amounts of data.

The CMD parameter consists of two decimal digits (for example 12). The first digit is the UNI/O address for the chip. Most chips use zero for this but the 11AA161 part number uses 1. Thus it is possible to parallel a 11AA160 and a 11AA161 on the same I/O pin and access them separately.

The second digit is the operation to perform as follows:

Second Digit	Operation
1	Read data from the chip.
2	Write data to the chip.
3	Erase the entire chip to all zero values The last three parameters of the CFunction will be ignored and can be left off (see the example below).
4	Erase the entire chip filling every byte with &HFF (all ones). As above, the last three parameters can be left off.

As examples:

01 will read from most versions of the 11XX chip (with UNI/O address of 0).

11 will read from a 11AA161 chip (with UNI/O address of 1).

02 will write to most versions of the 11XX chip (with UNI/O address of 0).

03 will erase the chip to all zero (assuming the chip has a UNI/O address of 0).

The NBR parameter will determine how many bytes will be written or read. For a floating point number this should be 4 and for an integer it should be 8. With integers it is possible to write less bytes if you are sure that the value of the integer will always be within a limited range. If the value of the integer will be 0 to 255 you only need to read/write one byte, for 0 to 65535 you can read/write two bytes and for numbers 0 to 4294967295 you can read/write 4 bytes. For negative numbers or if in doubt use the full 8 bytes for integers.

When reading/writing strings you can limit the number of bytes written to suit the maximum expected length of the string plus one. For example, if you know that the strings written will be 20 characters or less you can write 21 bytes (the extra byte is used by MMBasic to track the actual length of the string).

You can write arrays to the EEPROM by passing the array with empty brackets. For example dat(). You need to calculate how many bytes there are in the array and use this number for the NBR parameter. For example, if a floating point array is defined as DIM dat(10) you will need to write 44 bytes (arrays normally start from zero so there 11 elements with 4 bytes each).

Examples

Write the value of the floating point variable TmpRdg to address 1000 in an EEPROM on pin 15.

```
status = UNIO(15, 02, 1000, 4, TmpRdg)
```

The same only this time writing to a 11AA161 EEPROM (UNI/O address 1) connected to pin 15.

```
status = UNIO(15, 12, 1000, 4, TmpRdg)
```

Reading the value at address 4C (hex) into the floating point variable TmpRdg.

```
status = UNIO(15, 01, &H4C, 4, TmpRdg)
```

Writing the integer array dimensioned as IntArr%(5) to the same location.

```
status = UNIO(15, 02, &H4C, 48, IntArr%())
```

Writing the string Name$ which will have a maximum of 30 characters.

```
status = UNIO(15, 02, &H4C, 31, Name$)
```

Erase the entire chip to zeros. The last three parameters are not required and can be left off.

```
status = UNIO(15, 03)
```

More on Saving Strings

Saving strings to a UNIO EEPROM can be a little complicated so this section attempts to shed some light on this subject. Normally the detail of how a string is stored by MMBasic is hidden from the programmer but when storing them in an external EEPROM a deeper understanding helps because you are writing to physical memory locations.

To begin, you need to know how strings are stored in MMBasic. Take this example:

```
D$ = "abcd"
```

While the string is four characters long it is actually stored in memory as five bytes. The first byte is the length (ie, 4), the next byte is a, the third byte is b, and so on.

When writing a string to an EEPROM you would normally allow a fixed amount of the EEPROM for each string, this is so that you can easily find a string when reading from the EEPROM. For example, allowing 16 bytes per string, the first string would be at &H0000, the second at &H0010, the third at &H0020, etc. So, the command to write the variable D$ in the third slot would be:

```
S = UNIO(14, 02, &H0020, 16, D$)
```

and the code to read the string in the third slot would be:

```
S = UNIO(14, 01, &H0020, 16, E$)
```

Note that we do not care how long the string in D$ is, it could be one byte or five. The reason we do not care is because the first byte of the string is the length byte and that will be written to the EEPROM and read back in a read command. The only significance of the length byte is that it occupies one of the 16 bytes that we are writing to the EEPROM so that the maximum length string that you can write is 15 bytes. In fact, the code should look like this:

```
CONST MaxStr = 15
IF LEN(D$) > MaxStr THEN ERROR "String too long"
S = UNIO(14, 02, &H0020, MaxStr + 1, D$)
```

Also note that in the above examples we used 16 bytes to save each string but you could use 20, or 40, or how many bytes you need depending on the maximum length string that you expect to store.

Adding the UNIO command to the Library

If you add this CFunction to the MMBasic library it will act exactly the same as a built in command and you will not have to add the above code to any program that uses this command. It will take up a minimum of memory and it will not be erased when you use NEW or load a new program. To do this you can follow these steps:

- Load the CFunction code listed above into the Micromite using either AUTOSAVE, XMODEM or the MMEDIT program. Do not include any other program lines unless you also want them stored in the library.
- Enter the command LIBRARY SAVE

This will transfer the CFunction code to the library area and delete it from the main memory. You can now use the command in your programs exactly the same as if it was a built in command. The only way it can be removed is by using the command LIBRARY DELETE or by reprogramming the chip with the Micromite firmware.