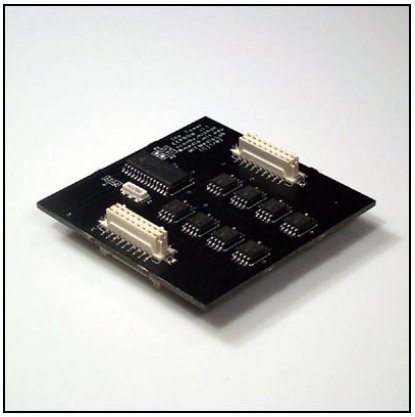
EEPROM Layer



Description

The EEPROM layer contains a bank of eight, 256-kilobit EEPROM chips, for a total memory of 256 kilobytes. The memory is useful storing values in data-collection applications.

Hardware Detail

There are eight, individually addressable EEPROM chips on the layer. While all chips communicate via the I2C serial protocol, it is important to note that they are not on the main Tower serial bus. To avoid address conflicts with the rest of the system, the PIC processor on the layer independently communicates with them using two of its standard I/O lines.

Layer Code

The include file for the EEPROM layer contains two functions, one for writing a byte to EEPROM, and one for reading a byte. *(All of these functions are written for the PIC Foundation. The include files used with other foundations differ slightly, due to the presence of local variables.)*

The **ee-write** function takes three arguments, the chip number, memory location, and the data value to be written. The chip number is given as a value from 0 to 7, and the memory location can be anywhere from 0 to 32768. A total of five arguments are being sent to the EEPROM layer. First a “0” is sent, indicating that a write function is being performed. Then, the chip number, memory location, and data values are sent, with the memory location argument being split into high and low byte portions to allow addressing of all memory locations. Please note that it does take some time to write to the EEPROM chip, and errors may occur if you issue two ee-write calls back-to-back.

```
to ee-write :chipnumber :addr :data
    i2c-start
    i2c-write-byte $0c
    i2c-write-byte 5
    i2c-write-byte 0
    i2c-write-byte :chipnumber
    i2c-write-byte highbyte :addr
    i2c-write-byte lowbyte :addr
    i2c-write-byte :data
    i2c-stop
end
```

The **ee-read** function operates similarly to the write function, sending the chip number and memory location as arguments to the layer. The first argument in this case however, is a "1", signifying a memory read. After the request is sent to perform the memory read, the result is read out of the layer's transmit buffer as a single byte. Actually two bytes are read out, but the first is ignored, as it is known to be a "1", representing the number of arguments to follow.

```
to ee-read :chipnumber :addr
    i2c-start
    i2c-write-byte $0c
    i2c-write-byte 4
    i2c-write-byte 1
    i2c-write-byte :chipnumber
    i2c-write-byte highbyte :addr
    i2c-write-byte lowbyte :addr
    i2c-stop
    i2c-start
    i2c-write-byte $0d
    ignore i2c-read-byte 1
    seti2c-byte i2c-read-byte 0
    i2c-stop
    output i2c-byte
end
```

Examples of Use

To use the EEPROM layer, a user would first want to write a value or string of values to memory, and then read them out at a later time. A simple example of how to write a number to memory is shown below:

```
ee-write 7 23 80
```

This code will write the value 80 to memory location 23, on chip 7. To test that the write worked correctly, one could do something like this to read the value back out:

```
print ee-read 7 23
> 80
```

The result of the read operation, which in this case should be 80, will then be passed to the print function, which then prints the value to the console. If you want to take a number of sensor readings, write them to memory, and then later read them out, a sample of how to do so is given here:

```
setn 0
repeat 5 [ee-write 0 n ((sensor 1) / 16) wait 10 setn n + 1]

setn 0
repeat 5 [print ee-read 0 n setn n + 1]
> 24
> 81
> 126
> 187
> 244
```

For this sample code to work, a Sensor layer must also be present on the Tower. In this example, the global variable "n" is first set to zero, then 5 values of the sensor in port 1 are read, and stored to

memory, with “n” representing the memory location being written to, which is incremented after each iteration. Since the sensor reading is a 12-bit number, we are dividing by 16 in order to store meaningful data in the 8-bit memory locations. The wait in the loop causes a half-second delay between sensor readings, to allow for a more interesting variety of sensor values. With the current delay, it will take 5 seconds to capture the 5 sensor values. After the data has been stored, we again reset the value of the global “n”. With another repeat loop of 5 iterations, we print out the stored values to the console one at a time, using n as our memory location counter as in the write loop.