# Mini Memory Tester

*Beta Documentation*

By Dagen Brock

# Table of Contents



## Description

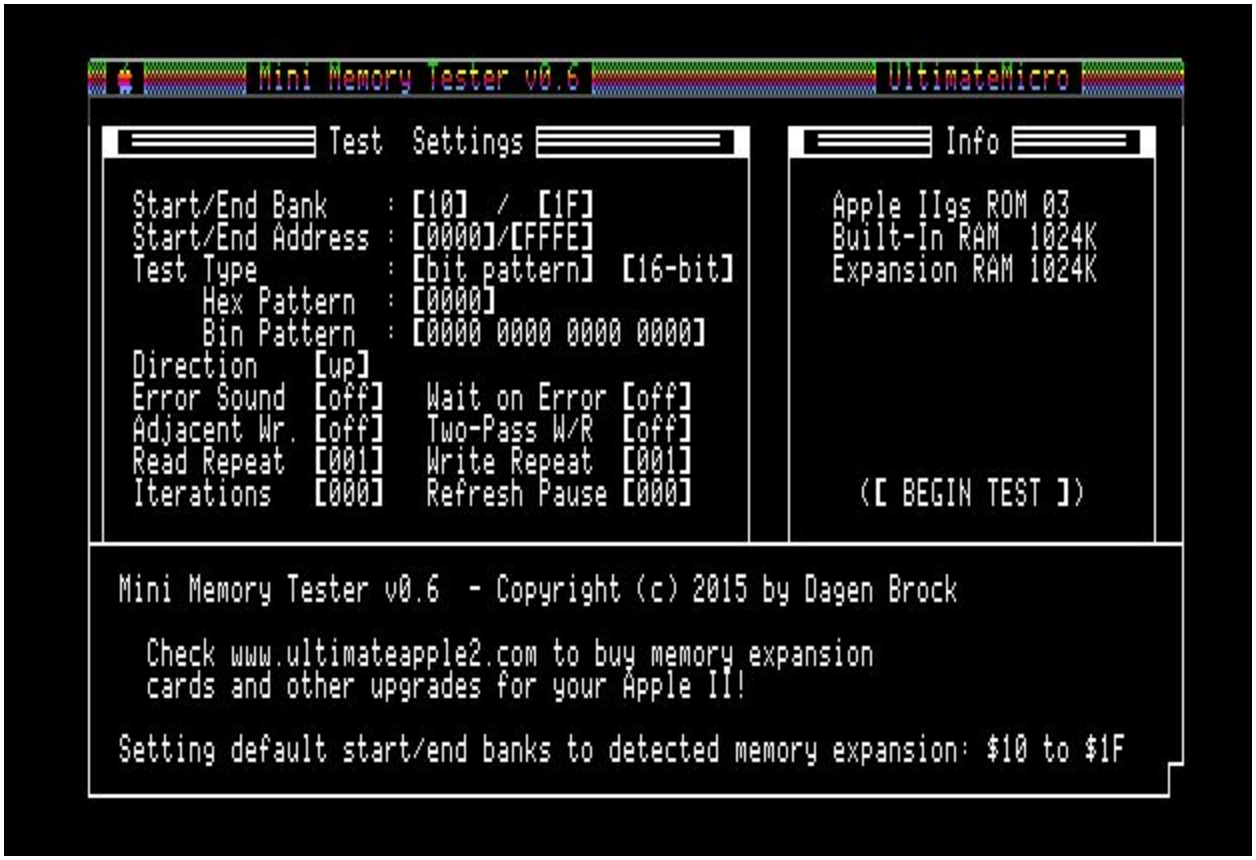
Mini Memory Tester is a small, but powerful, testing program for Apple IIgs memory cards.

## Project

You can visit the project, including all sources, online at http://github.com/digarok/mmt

# Main Menu

Here's a look at the main screen.




The screen is separated into 3 areas.

1. The main "Test Settings" area on the left side is where you configure your test.
2. The "Info" pane on the right shows RAM information and displays test information while a test is running.
3. The "Log" or console area at the bottom shows a log of messages regarding the test.

## Navigation

Use the up and down arrows, or the left and right arrows, to select the menu item you want (indicated with blinking angle brackets), and hit enter to edit it, or if "BEGIN TEST" is selected, it will begin testing memory using the specified parameters.

When entering values, it expects hexadecimal numbers (0 through F) for addresses, and integers (0-9) for any other numeric values, such as "Read Repeat" or "Iterations".

You do not need to hit enter when typing values. Text entry is complete when you type the appropriate length of hexadecimal number or integer. You can hit ESC to cancel the current number entry.

*Note that the program will let you set whatever values you want, and that may cause your computer to crash, blow up, run away, or worse. I am not responsible for anything bad that happens. But it's probably okay.*

## Test Parameters

Some test parameters should be fairly obvious, such as "Start/End Bank" to set the starting bank and ending bank for your test. Here's a description of each parameter and what it does.

- Start Bank - End Bank
    - Range of memory banks that will be tested
- Start Address - End Address
    - Hex address to test at within banks
- Test Type
    - Bit Pattern
        - This is whatever you enter in the "Hex Pattern" box. If you enter "1234", it will write "1234123412341234" to the memory during the test.
    - Bit Walk 1
        - This writes and reads a "walking" 1 bit. For example, in 8-bit mode the first test uses, %10000000, then %01000000, then %00100000, etc., until it gets to %00000001.
        - When using a Bit Walk mode, it is slower because it must test each possible pattern at that memory location.
    - Bit Walk 0
        - The inverse (bitwise) of the above Bit Walk 1. This writes and reads a "walking" 0 bit. For example, in 8-bit mode the first test uses, %011111111, then %10111111, then %11011111, etc., until it gets to %11111110.
    - Random
        - This starts with a new random seed each bank, and then each value in the bank is chosen via a pseudo-random number generator.
- 16-bit/8-bit
    - This affects whether the test uses a 16-bit or 8-bit data size.
    - 16-bit
        - In this mode, tests write and read 16-bits using a single long indexed store instruction (e.g. STAL $050000,x). This should mean that you are using all the data lines, and address lines including the multiplexed bank lines.
    - 8-bit
        - In this mode, long instructions are still used, but we are only writing 8-bit values. I'm no CPU expert, but I think this is only using 8 of the data lines.

- Hex Pattern
    - Set this to the value you want to write during your test. (ONLY WORKS FOR TEST TYPE “BIT PATTERN”!)
      *NOTE: In other test modes, this field will show the last value tested.*
- Bin Pattern
    - This shows the binary representation of the Hex Pattern above. They are the same number. Unfortunately, you cannot enter a binary number at this time.
- Direction
    - Up - This will start at the low Bank & Address and finish at the high Bank & Address
    - Down - This will start at the high Bank & Address and finish at the low Bank & Address
- Error Sound
    - This makes the program play 3 error chimes any time it encounters a read error – helpful for unattended usage.
- Wait On Error
    - Wait for a keypress after an error message is printed. (Otherwise, continuously test without intervention.)
- Adjacent Write
    - This writes to the destination location, as well as its neighbors above and below. Example, if testing in 16-bit mode, and the test is currently writing to 08/2006, then it will simultaneously (not literally) write the same value at address 08/2004 and 08/2008. Remember, 16-bit mode moves by 2 bytes per word.
    - You can’t use Adjacent Write with Two-Pass mode.
- Two-Pass W/R
    - Normally, a test will write to a location, and then read the location to see if the value is correct. In two-pass mode, the test will write to all specified addresses in a bank, and then read all of those addresses in a second pass.
    - You can’t use Two-Pass mode with Adjacent Write.
- Write Repeat
    - Set this to write the value this many times to the destination address.
- Read Repeat
    - Set this to read the value this many times from the destination address. Any single read failure will trigger an error.
- Iterations
    - How many test passes to run before stopping. 0 = Infinite.
- Refresh Pause
    - Micro Pause between writes and reads. I’m still adjusting this one.

# Testing

Once you begin the test, it will loop forever unless you have a set number of iterations, but you can cancel the test at any time by pressing “Q” or the “ESC” key.  You can also pause the test with the “P” key.

## Test Address Ranges

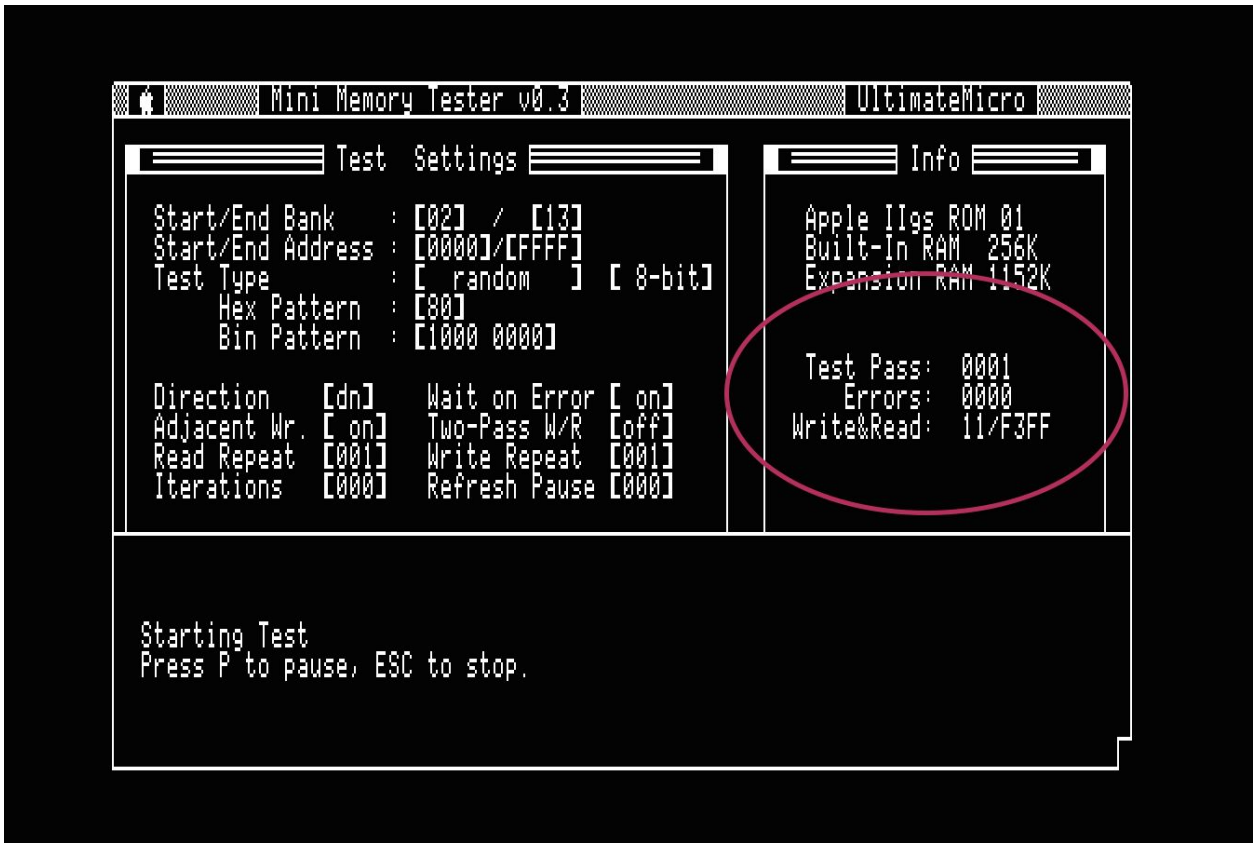
If you specify a range of, say “2000-8000” across multiple banks, then it will only test that range. Example, if you say banks 10 through 11, with the address range above then it will only test:

10/2000 … 10/8000
11/2000 … 11/8000

Just be aware that it’s not contiguous between banks.

## Running Tests

Running tests show the current pass, the total number of errors encountered, and the current address chunk being written or read.




After each pass of writing and reading, the pass number is incremented, and the next pass starts.

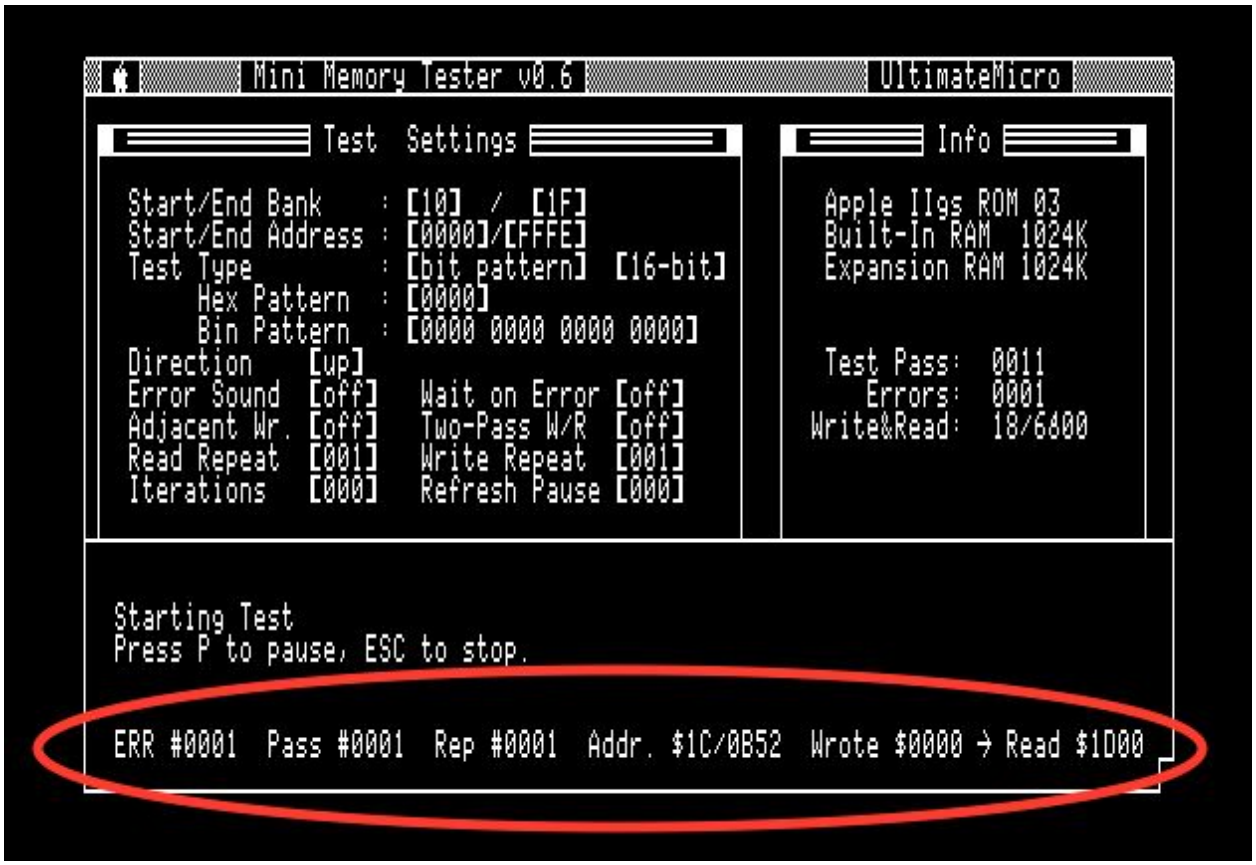
**Note: Press P to pause testing and ESC to stop.**

# Memory Errors

When the test encounters an error, it will show an error message in the Console Log window. The Console Log will show the last 6 errors. You can see a total number of errors in the Info window.

## Generating memory corruption

It is helpful (particularly to test the program itself!) to be able to corrupt some memory during the test to see that things are working properly. To do this, simply press “C” to “corrupt” the current spot being written. Then when the read pass occurs, it should print an error message. (Sometimes, keypresses don’t get to the corruptor, so you may have to hold it down for a second.)




As you can see, it shows the error in the bottom of the Console Log. The error message includes the error number (ERR), the test pass, the read repeat it occurred on (Rep), the address, the expected value, and the value we got when we read that location.

## Other Notes

The program is designed to test **EXPANSION RAM**, not internal RAM which is built-in to the system. Can you set the program to test internal banks such as $00, $01, $E0 or $E1? Sure. Go ahead. It will probably not end well, but I allow you to completely configure start and end addresses and you can use this to also test your internal RAM.

## Credits

Thanks to Henry Courbis and Anthony Martino of UltimateMicro for helping with the feature design, testing and debugging. It would not be as thorough (relatively) of a piece of software, had they not motivated me through a long list of feature requests! ;)

- https://ultimateapple2.com/

Thanks to Brutal Deluxe for the Merlin32 compiler which made this so much faster to write, and Cadius, which I used for both code formatting and disk images.

- http://brutaldeluxe.fr/products/crossdevtools/merlin/index.html
- http://www.brutaldeluxe.fr/products/crossdevtools/cadius/

Thanks to Kent Dickey and the GSPort maintainers for the emulator used to write this. I crashed it many times writing this program! :D

- http://gsport.sourceforge.net/

## Release Notes

1.0 - Let's release this sucker! (November 27, 2015)

- Fix Corruptor bug

0.6 - "Beta" release to Ultimate Micro (November 12, 2015)

- Fix ROM03 crash
- Fix Wait on Error functionality
- Add audio alert

0.5 - "Beta" release to Beta Mailing List (October 30, 2015)

- Rewrote RAM detection routine - now works properly on my main IIGS

0.4 - "Beta" release to Ultimate Micro (October 25, 2015)

- Entirely new feature set. See this Doc.

0.3 - Internal Only - Massive Rewrite Done (October 15, 2015)

0.2 - Internal Only - Start Massive Rewrite (September 16, 2015)

0.1a - Second Alpha (March 11, 2015)

- Fix 0 second delay bug
- Add "P"ause
- New Layout
- Display ROM revision (not tested)
- Add logging console
- Display errors in console instead of pop-up, removing key wait
- Error counter

“Alpha” - Initial release (March 4, 2015)