

0

Search Adafruit

SHOP

BLOG

LEARN

FORUMS


VIDEOS

SIGN IN

CLOSE MENU

0 Items

Sign In



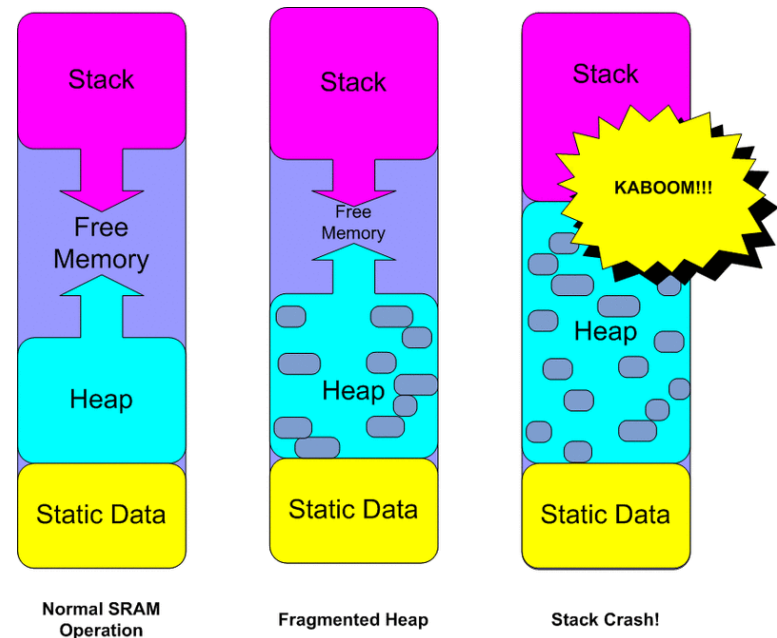
SHOP

BLOG

LEARN

FORUMS

VIDEOS



Memories of an Arduino

How to get the most from your Arduino Memory

You know you have a memory problem when...

Memory Architectures

- Arduino Memories
- Arduino Memory Comparison

Measuring Memory Usage

Large Memory Consumers

Solving Memory Problems

- Optimizing Program Memory
- Optimizing SRAM
- Using EEPROM

Single Page

Download PDF

Contributors

Bill Earl

Feedback? Corrections?

LEARN ARDUINO

Optimizing SRAM

by Bill Earl

SRAM is the most precious memory commodity on the Arduino. Although SRAM shortages are probably the most common memory problems on the Arduino. They are also the hardest to diagnose. If your program is failing in an otherwise inexplicable fashion, the chances are good you have crashed the stack due to a SRAM shortage.

There are a number of things that you can do to reduce SRAM usage. These are just a few guidelines to get you started:

Remove Unused Variables

If you are not sure whether a variable is being used or not, comment it out. If the sketch still compiles, get rid of it!

F() Those Strings!

(Park the char* in Harvard PROGMEM)

Literal strings are repeat memory offenders. First they take up space in the program image in Flash, then they are copied to SRAM at startup as static variables. This is a horrible waste of SRAM since we will never be writing to them.

Paul Stoffregen of PJRC and Teensyduino fame developed the F() macro as a super-simple solution to this problem. The F() macro tells the compiler to keep your strings in PROGMEM. All you have to do is to enclose the literal string in the F() macro.

https://learn.adafruit.com/memories-of-an-arduino/optimizing-sram

For example, replacing this:
[Copy Code](#)

```
1. Serial.println("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-a-am sram. Lovely sram! Lovely sram! Lovely sram! Lovely sram! Lovely sram! Sram sram sram sram!");
```

with this:
[Copy Code](#)

```
1. Serial.println(F("Sram sram sram sram. Lovely sram! Wonderful sram! Sram sra-a-a-a-a-am sram sra-a-a-a-a-am sram. Lovely sram! Lovely sram! Lovely sram! Lovely sram! Lovely sram! Sram sram sram sram!"));
```

Will save you 180 bytes of wonderful SRAM!

Reserve() your strings

The Arduino string library allows you to reserve buffer space for a string with the reserve() function. The idea is you can prevent String from fragmenting the heap by using reserve(num) to pre-allocate memory for a String that grows.

With the memory already allocated, String doesn't need to call realloc() if the string grows in length. In most usages, lots of other little String objects are used temporarily as you perform these operations, forcing the new string allocation to a new area of the heap and leaving a big hole where the previous one was (memory fragmentation). Usually all you need to do is use reserve() on any long-lived String objects that you know will be increasing in length as you process text.

You can do better with C strings, but if you just follow these guidelines for String objects, they work nearly as efficiently and using them is so much easier.

Move constant data to PROGMEM.

Data items declared as PROGMEM do not get copied to SRAM at startup. They are a little less convenient to work with, but they can save significant amounts of SRAM. The basic Arduino reference for PROGMEM is [here](#). And there is a more detailed tutorial on the subject [here](#).

Reduce Buffer Sizes

Buffer and Array Allocations:
If you allocate a buffer, make sure it is no bigger than it needs to be.

Buffers in Libraries:
Also be aware that some libraries allocate buffers behind the scenes that may be candidates for trimming as well.

System Buffers:
Another buffer hidden deeply in the system is the 64 byte serial receive buffer. If your sketch is not receiving a lot of high-speed serial data, you can probably cut this buffer size in half - or maybe even less.

The Serial buffer size is defined in HardwareSerial.cpp. This file can be found in your Arduino install directory:

```
....\Arduino-1.x.x\hardware\arduino\cores\arduino\HardwareSerial.cpp
```

Look for the line:

```
#define SERIAL_BUFFER_SIZE 64
```

And change it to 32 or less.

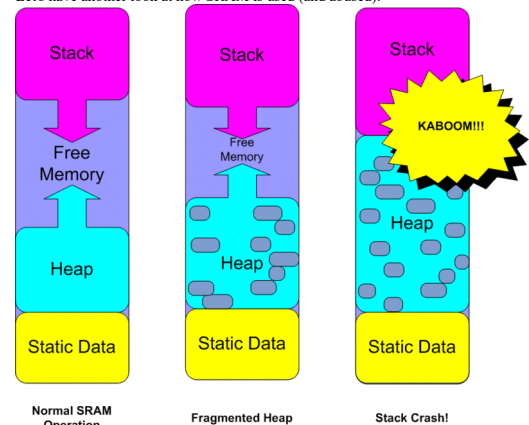
Reduce Oversized Variables

Don't use a float when an int will do. Don't use an int when a byte will do. Try to use the smallest data type capable of holding the information.

Data Types	Size in Bytes	Can contain:
boolean	1	true (1) or false (0)
char	1	ASCII character or signed value between -128 and 127
unsigned char, byte, uint8_t	1	ASCII character or unsigned value between 0 and 255
int, short	2	signed value between -32,768 and 32,767
unsigned int, word, uint16_t	2	unsigned value between 0 and 65,535
long	4	signed value between -2,147,483,648 and 2,147,483,647
unsigned long, uint32_t	4	unsigned value between 0 and 4,294,967,295
float, double	4	floating point value between -3.4028235E+38 and 3.4028235E+38 (Note that double is the same as a float on this platform.)

Think Globally. Allocate Locally.

Let's have another look at how SRAM is used (and abused):



Global & Static Variables

Global and Static variables are the first things loaded into SRAM. They push the start of the heap upward toward the stack **and they will occupy this space for all eternity.**

Dynamic Allocations

Dynamically allocated objects and data cause the heap to grow toward the stack. Unlike Global and Static variables, these variables can be de-allocated to free up space. **But this does not necessarily cause the heap to shrink!** If there is other dynamic data above it in the heap, the top of the heap will not move. When the heap is full of holes like swiss cheese we call it a "fragmented heap".

Local Variables

Every function call creates a stack frame that makes the stack grow toward the heap. Each stack frame will contain:

- All parameters passed to the function
- All local variables declared in the function.

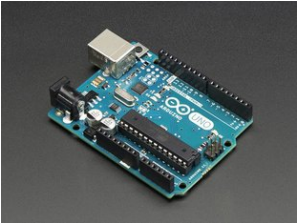
This data is usable within the function, but **the space is 100% reclaimed when the function exits!**

The Takeaway?

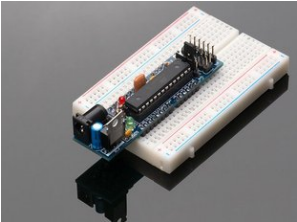
- **Avoid dynamic heap allocations** - These can quickly fragment the limited heap-space.
- **Prefer local to global allocation** - Stack variables only exist while they are being used. If you have variables that only are used in a small section of your code, consider making that code into a function and declaring the variables local to the function.

OPTIMIZING PROGRAM MEMORY USING EEPROM

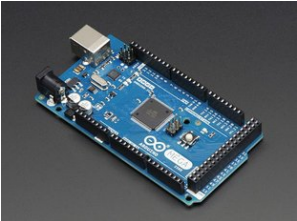
Last updated on 2016-04-17 at 06.36.31 PM Published on 2013-08-02 at 10.19.49 AM



Arduino Uno R3 (Atmega328 - assembled)
\$24.95 [Add To Cart](#)



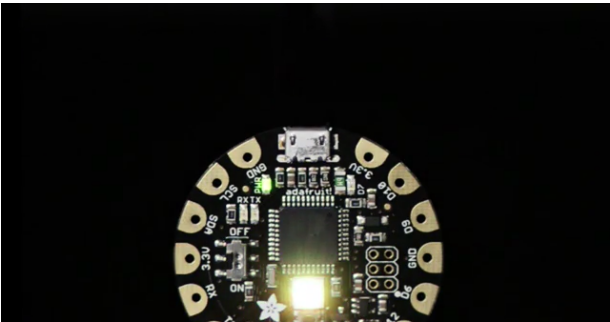
DC Boarduino (Arduino compatible) Kit (w/ATmega328)
\$17.50 [Add To Cart](#)

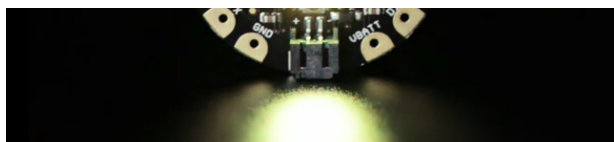


Arduino Mega 2560 R3 (Atmega2560 - assembled)
\$45.95 [Add To Cart](#)



Arduino Mega R3 Android Accessory Development Kit (ADK) Board
\$84.95 [Add To Cart](#)





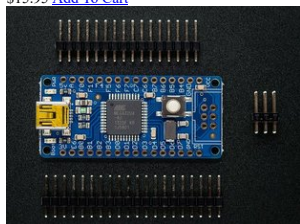
FLORA - Wearable electronic platform: Arduino-compatible
\$19.95 [OUT OF STOCK \(NOTIFY ME\)](#)



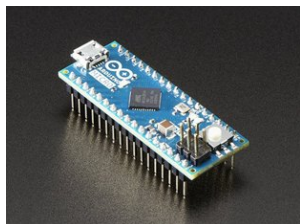
Arduino Leonardo ATmega32u4 without headers
\$22.50 [OUT OF STOCK \(NOTIFY ME\)](#)



Teensy (ATmega32u4 USB dev board) 2.0
\$15.95 [Add To Cart](#)



Atmega32u4 Breakout Board
\$19.90 [Add To Cart](#)



Arduino Micro with Headers - 5V 16MHz - (ATmega32u4 - assembled)
\$24.95 [Add To Cart](#)



Arduino Uno Ethernet
\$65.00 [OUT OF STOCK \(NOTIFY ME\)](#)
[ADD ALL TO CART](#)

RELATED GUIDES

[Metal Inlay Capacitive Touch Buttons](#)

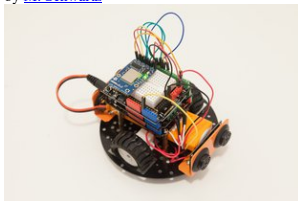
[Make pretty things you can touch and make bleepy noises](#)
by [Todd Treece](#)



[Learn how to grind up aluminum foil into a fine metal powder for use as capacitive touch buttons.](#)

[WiFi Controlled Mobile Robot](#)

[Control your robot via WiFi using Arduino & the CC3000 WiFi chip](#)

by [M. Schwartz](#)

Ever wanted to control your mobile robot remotely from your computer or your smartphone ? Now it is possible using Arduino and the CC3000 WiFi chip. In this guide you'll learn how to install the required components on a robot, and how to set the software environment so you can control the robot right from your browser

[FEATURED](#)[Light-Up Angler Fish Embroidery](#)

Deep sea stitching with FLORA

by [Becky Stern](#)

Make this simple circuit with a single glowing lure on a menacing embroidered angler fish and dress up your shorts for summer!

[2.8" TFT Touchscreen](#)[320x240 pixels in 16 bit color with a touchscreen](#)by [lady ada](#)

Add some jazz & pizzazz to your project with a color touchscreen LCD. This TFT display is big (2.8" diagonal) bright (4 white-LED backlight) and colorful (16-bit 262,000 different shades)! 240x320 pixels with individual pixel control, this has way more resolution than a black and white 128x64 display. As a bonus, this display has a resistive touchscreen attached to it already, so you can detect finger presses anywhere on the screen. Learn how to use this LCD with an Arduino.

[x](#)

OUT OF STOCK NOTIFICATION

YOUR NAME YOUR EMAIL [NOTIFY ME](#)

- [CONTACT](#)
- [SUPPORT](#)
- [DISTRIBUTORS](#)
- [EDUCATORS](#)
- [JOBS](#)
- [FAQ](#)
- [SHIPPING & RETURNS](#)
- [TERMS OF SERVICE](#)
- [PRIVACY & LEGAL](#)
- [ABOUT US](#)

ENGINEERED IN NYC Adafruit ®

"Art is I; science is we" - [Claude Bernard](#)