FLAN Protocol

Rob Swan

May 3, 2021

1 Introduction

This document outlines a way to use the Megasquirt Serial Protocol to update flash on a device which has no connection besides its CAN bus to the Megasquirt. While TunerStudio doesn't include this protocol, you can use the same serial connection you use with it. A program on the laptop controls the conversation. It talks only to the Megasquirt which acts as a bridge to the CAN bus.

I wanted this for a simple dashboard which needed to be able to load new layouts or firmware updates.

It's not a difficult problem, but there were a number of things I needed to discover for myself. It would have been handy to have an outline like this showing *one way* that works.

Incidentally, FLAN stands for *flash over CAN*.

2 The Protocol

The Megasquirt Serial Protocol was designed for the transfer of data. It has the notion of passing runs of bytes at an offset in one of several fixed sized tables. It was not intended for transferring firmware images, but it is possible to build a suitable protocol on top of the *table, offset, byte stream* scheme.

All interaction will use just one table: table number 2. This allows application code on the device to use the CAN protocol for other tables, with just this table reserved for firmware duty. Table 2 was chosen on the recommendation of Jean Bélanger for consistency with other Megasquirt CAN-enabled devices.

The Serial Protocol has *r* (read) and *w* (write) operations. When it sees that the destination is a different CAN host, the Megasquirt turns them into one or more CAN *REQ* or *CMD* operations respectively. The read/REQ operation retrieves one or more bytes from a notional table and offset on the host. The write/CMD operation sends one or more bytes to a table and offset.

The *read* operation is very useful because the two-way transfer confirms success or failure of the operation. This makes it suitable as a kind of remote procedure call. The *write* operation is still necessary to transfer the data. On completing such a transfer a *read* is used to verify success.

Here are the operations:

Code	Serial	CAN	Device Action
`START`	`r C 2 0x200 1`	`REQ 2 0x200 1`	Enter stand alone flasher
`END`	`r C 2 0x201 1`	`REQ 2 0x201 1`	Exit stand alone flasher; reboot
`ERASE`	`r C 2 0x202 1`	`REQ 2 0x202 1`	Erase current page
`SUM`	`r C 2 0x203 5`	`REQ 2 0x203 5`	return success code, 16-bit checksum, 16-bit number of bytes written since last SUM operation
`FLUSH`	`r C 2 0x204 1`	`REQ 2 0x204 1`	flush buffered data, burning it to flash. Return success or fail if this operation or any preceding writes were invalid.
`READ`	`r C 2` *off len*	`REQ 2` *off len*	return *len* bytes at offset *off* from current flash page.
`PAGE`	`r C 2 0x400+`*p* 1	`REQ 2 0x400+`*p* 1	Set current page to *p*
`DATA`	`w C 2` *off len*	`CMD 2` *off len*	Write *len* bytes of data to current page at offset *off*

The device's memory is divided into pages of up to 512 bytes each. It is up to the device how it chooses to define these pages. At a minimum, a page is the smallest erasable chunk of flash memory. *PAGE* sets the current page on the device. If that page is invalid, the device should return a failure code. No other operation will succeed if a page has not been set.

Page numbers can be anything from 0 to 511, allowing up to 256k of flash to be addressed. Further extension wouldn't be too hard, for example you could continue to use table 2 for control, and use the whole of table 3 just for page numbers.

Typically *ERASE* will be performed immediately after *PAGE*, then one or more *DATA* commands will transfer data to the device. Flash memory is usually slow to write and the Megasquirt turns these DATA commands into many 8-byte transfers which happen quickly. So the device is safest to store the bytes in a RAM buffer.

Once a suitable number of bytes have been transferred (say 64, or 256, or 512), *FLUSH* is called to commit the remote buffer to flash and return a success/fail code.

Comparing the result of *SUM* with a locally evaluated checksum of the current page will confirm successful transfer.

DATA is the only *write* operation. All the other operations are encoded as special *read* requests. These requests are differentiated using special values for the *offset*.

The *READ* operation is not necessary for writing flash, but allows the existing contents of memory to be downloaded in a similar process of setting the page and reading the bytes on that page.

2.1 MSP430 Implementation

The MSP430G2553 has *program flash* from `0xc000` to `0xffff` organised as 512 byte flash pages. You can't erase fewer than 512 bytes in this memory. It also has four 64 byte *info flash* pages at `0x1000` to `0x10ff`. The fourth of these pages contains processor calibration values and should never be overwritten.

Page numbers have been assigned as follows:

address	page
`0x1000`	0
`0x1040`	1
`0x1080`	2
illegal	3
`0xC000`	4
`0xC200`	5
`0xC400`	6
`0xC600`	7
...	...
`0xF400`	30
`0xF600`	31

That accounts for all the writable flash on this processor, apart from the top 2k where the fixed monitor code (which implements this protocol) is stored and should never be overwritten. The first three pages are the small config flash area.

A similar mapping would be required to support other processor memory maps. Similar laptop software would work, but the processor firmware would need to be written from scratch.