Boot Protocols for GreenArrays Chips

Applicable starting GA4-1.2 and GA144-1.10

There are four stream oriented boot protocols. The stream protocols all serve to deliver a sequence of 18bit words to (or through) the boot node. Although the bit delivery format varies with the protocol, the order and interpretation of the word stream is the same. The only effects upon the stream's word content which occur due to choice of boot node and are related to the fact that the stream's route are interpreted locally, within the framework of the receiving node. This document specifies the format of a boot stream, and the delivery details of each of the supported protocols. Please refer also to the ROM documentation applicable to each chip for detailed information about the version of boot code provided in that chip.

1. The Common Boot Stream Format

The boot stream is delivered from an external source. The boot node which receives the stream will know the stream has ended because the stream will control the execution of that node in such a way as to insure stream termination. The actual means that are used within the stream to affect this control (over a chip) are not specified (parsible) by the service that must deliver it. Therefore, the delivery service outside the chip must be separately aware of what stream it intends to deliver, and of the total length of that stream.

1.1 Boot Frames

A boot stream is composed of one or more concatenated *boot frames*. Each boot frame begins with a three word header before zero or more data words in the following format.

0	Completion (jump) Address
1	Transfer (store) Address
2	Transfer Size in words
3	Data words (if size ≠0)

Only the low order 10 bits of word 0, and the low 9 bits (for F18A) or 8 bits (for F18B) of word 1 are significant. Their high order bits are ignored. The transfer size is exactly the number of data words in the boot frame; if it is zero, no data words follow. The maximum size of a boot frame is 2^{18}-1 data words. The maximum size of a boot stream is not given. All boot nodes will interpret each boot frame as follows:

1.1.1 Completion Address

The completion address is held on the return stack until the end of the boot frame, when it is used as a return address to affect a jump to that address. If the purpose of the boot frame is to transfer code to RAM in the boot node and execute it there, then the completion address will point somewhere within the loaded code. Other special actions can be affected by pointing the completion address somewhere in ROM, RAM or even a Port. The subsequent effect on the protocol can be anything from termination to redefinition or extension.

The most common use is to point the completion address at that ROM location within the boot node that will begin reading a new boot frame at the point where the previous boot frame left off. It has been arranged in the ROM of all streaming protocol boot nodes that the internal stack effect is matched when this transfer address is used. The effect on the protocol of using this address is to concatenate one more boot frame to the current boot stream. This can continue indefinitely until a boot frame appears with a different completion address. It is also potentially possible to change the protocol entirely, midstream, by loading a new driver into RAM and jumping to it, and clearly such a protocol change would require the cooperation and coordination of the external delivery service as well.

Stream compilers must be aware of the concatenation transfer address for the targeted boot node.

1.1.2 Transfer Address

The transfer address is loaded into the boot node's A register and each data word taken from the boot frame is stored there using the **!+** instruction. This means that if the address is in RAM, consecutive locations will be filled, but if the address is of a Port then incrementation is suppressed and the entire boot frame will be pumped out that port. Data words are taken from the boot frame one at a time and delivered to the destination. This has two implications for timing if the destination is a port:

- The next word, if any, for that port will certainly not be available immediately and may not be available for a long time, since all present boot media are far slower than F18 internal speeds.
- If the recipient of data on that port is not ready so that the transfer may be completed before it is necessary for the boot node to pay attention to its medium, the boot node can lose synchronization. The shortest time intervals imposed by this requirement occur in the 2-wire synchronous and 1-wire protocols; unless the sender of the boot stream inserts a time delay between words, this interval can be as short as a fraction of a bit time on the line. The only boot medium that currently can tolerate arbitrary delays is the SPI memory.

1.1.3 Transfer Count

If the count is zero, no data follow. Otherwise it is the actual number of words represented as an 18-bit value. When the indicated number of words have been transferred, the boot frame has ended and control passes to the transfer address.

1.2 Boot Stream Portability

Generally, a particular boot stream must be compiled specifically for a specific medium (because the concatenation transfer addresses must vary between different sorts of boot ROM), for a specific boot node to be used (because any effort to load other nodes must be aware of the *position* of the boot node in the array), and for a specific chip part number (because of array size, computer and I/O model, and the content and addresses in ROM.) Boot streams portability is not supported at the binary level, but rather at the source code level.

2. SPI Flash Boot

An SPI controlled flash memory device is a slave device that receives an eight bit command byte with a possible 16 (or 24) bit address parameter, both sent high bits first. During execution of the read command, the memory device emits bytes, one bit at a time, high bits first. This output is completely synchronous and can be paused and resumed, or in fact stopped at any time and between any pair of bits. This means it can be treated as a bit serial memory of arbitrary depth. The only real nods to byteness within the part are that the address parameter refers to byte locations, and in the writing protocols.

The implementation of stream structure in an SPI flash device is simply a contiguous packing of 18 bit words, high bits first. Because the SPI memory is a simple slave device, the SPI boot routine must produce its own read command and start address and pass them to the memory. In the standard SPI boot ROM, a 24-bit starting address of zero is sent to the SPI device to begin reading. The boot stream can "jump" to an SPI address by storing a short program into boot node RAM and executing it to call the function named **spi-boot** that starts stream processing from a given location. The concatenation address used for the SPI node is that location named **spi-exec** just after the read command is issued and just before reading out the header information from whatever bit position the memory device has been left at from the conclusion of the previous boot frame.

Because the SPI memory must act as a protocol slave operating at high speeds, only computers (such as native FORTH systems) in which timing may be deterministically controlled by the programmer are suitable for simulation of the SPI memory.

2.1 Protocol Details

Upon reset each F18 SPI node will check its SPI data-in pin (node.17). If the pin is high the node will not try to boot from the SPI interface and will instead go to sleep in its default multiport execute. In this case, none of the node's pins will have been changed and so all four pins will remain set for weak pulldown.

If the data-in pin from SPI memory is low, as it will be if not driven, the F18 investigates to see if it can find what appears to be a valid SPI boot device. In this case the four pins of the SPI node are assigned to the following signals:

node.17	DI	Data in (SO, serial out, from SPI device)	Tristate
node.5	DO	Data out (SI, serial in, to SPI device)	Driven
node.3	CS-	Chip select to SPI device	Driven
node.1	SCK	Serial clock to SPI device	Driven

The F18 begins, as shown in the diagram below, by selecting the device. It drives DO low, CS- and SCK high. After one-half bit time CS- is asserted by driving it low for one-half bit time. (The default half bit time used by boot ROM is 498 cycles of a micronext loop `begin . . unext` which takes ≈2.56 µS on an F18A and ≈2.30 µS on an F18B, for effective bit rates of ≈195 Kb/S and ≈217 Kb/S respectively, well within the capabilities of most conceivable devices. If a faster device is in use, a boot frame can be inserted at the start of a stream to change this parameter and resume processing the remainder of the stream at device speed.)

The F18 then drives SCK low, starting SPI in mode 3. Data is sent to the SPI Flash on the falling edge of SCK and is read by the SPI Flash on the rising edge of SCK giving the signal time to stabilize. Data is output by SPI flash to the F18 on the falling edge of SCK and is read by the F18 on the rising edge of SCK giving the Flash device time to detect the falling edge and for the signal to stabilize before it is read in the second half of the clock cycle.

In the first thirty-two cycles of the clock produced by the F18, the F18 sends a read command (03) and sends a 24-bit address of zero. Data Out from the F18 is raised only for the last two bits of the eight-bit read command. This tells the SPI flash device to start sending the contents of the flash from address zero as consecutive bits on each clock falling edge. Figure 1 shows the signals for initialization of the SPI Flash and the read command.

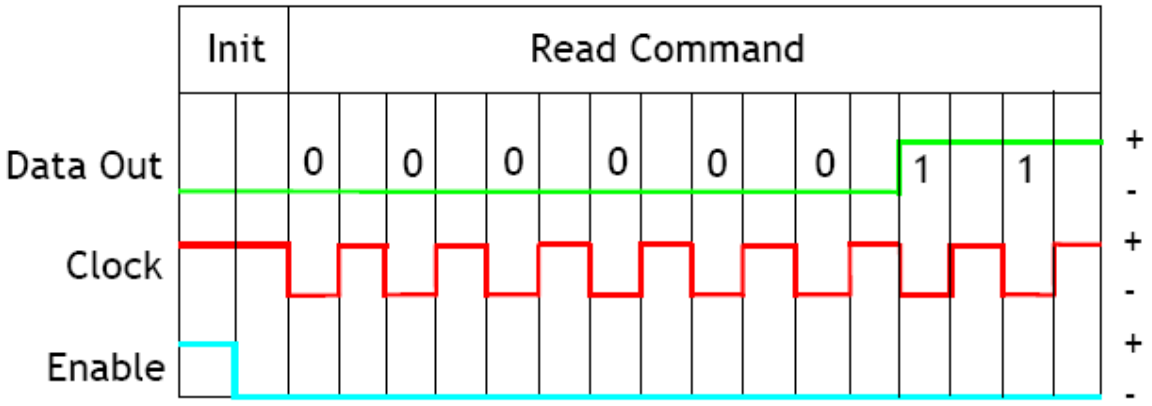

Figure 1. Eight clock cycles for command output followed by 24 more address cycles

After starting the read operation, the F18 reads the first 18-bit word from the SPI device and applies a validity check: The high order six bits of this value must lie within hex 2 and 21, inclusively. If this test fails, the F18 abandons booting and goes to sleep in its default multiport execute. Thus, any boot stream to be stored into SPI flash must have a value in this range as the high order six bits of the completion address of its first boot frame. This specifically excludes booting from erased flash (typically all ones) and from no device at all (even though pin 17 is high impedance it should read as all zeroes soon after it was reset.)

2.2 ROM Specific Details

The standard ROM has two significant characteristics that may be changed in the future. If pin 17 is high after reset, the F18 will try to access the flash, and if there actually is a flash connected it will be left in the midst of a read command regardless of whether the validity test passed. This leaves the flash driving pin 17. In order to use the SPI node's pin 17 as an I/O pin, the program will have to reissue a select operation to the flash device. This can be done by executing the following program in the SPI node: `io b! 497 dup select`

Secondly, if pin 17 is high after reset, the SPI node will be busy for a minimum of 51 clock cycles making the validity check. This is a long time at the default clock rate, ≈117-131 µS. During this time the node is not listening to any of its comm. ports, and if any other boot device tries to move data into the SPI node before it has finished this process, operation of the other boot node will be suspended until the SPI node has finished. In the case of a small chip and a fast boot protocol such as 1-wire, it is quite possible to reach the SPI node before it has finished its validity check.

3. Asynchronous Serial Boot

The asynchronous input nodes all incorporate an automatic baud rate detection that is recalibrated on every three bytes of input. Each 18 bit input word is encoded as three bytes. The data are presented low bits first, with two bits of data in the first byte and eight data bits in each of the next two bytes.The first byte begins with a 6 bit pattern that ensures that the node has time to wake up and make a clean measurement of a double wide bit cell. The time measured is then cut by half to input the remaining data. Each byte has a one bit start bit that synchronizes the input for the rest of the byte, but which does not reset the measured bit size.

The asynchronous protocol is designed to be produced by a standard UART and to allow the serial node to sleep between messages. On a standard ±12v RS232 line, the rest state is low, the start bit is high, and data bits are low if 1. Whatever method is used to adapt the 12v incoming levels to our range [0..1.8V] must do so without inversion, so that the rest state of our input is still low and the start bit is still high. This permits us to use a series resistor as a trivial RS232 line receiver.

All actual data bits must be inverted before presentation to the UART by the external delivery service so that they will be high true when read by the F18 directly from RS232 without inversion. The six bit calibration pattern in the first byte of each word is binary `010010` as presented to the UART. When combined with the UART's own start bit, passed through the inverter, and viewed low bits first, these bits will appear to the the F18 as the bit sequence `1101101` where the first two one bits will wake up the node if it is sleeping. The F18 measures this first pulse time but will discard that value in favor of what is measured for the next two bits for which it will be ready and waiting to measure beforehand. The last one bit serves as a synchronizing start bit for the first two bits of transmitted data that will fill out this first byte sent. The data appear to the F18 as follows:

```
  This is a model of input to the serial port. The data is in 8n1 format,
breaking each 18-bit data word into 3 bytes as shown. The low six bits are set
to 0x2D to ensure that the c18 can time the first start bit. The mark-space
voltages are inverted so it looks like this:


              _____________       _____________       _____ _____ _____
1st       ___/             \_____/             \_____/     \_____X_____\___
byte  stop  start  [0]   [1]   [2]   [3]   [4]   [5]    0     1   stop


              _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
2nd       ___/     \_____X_____X_____X_____X_____X_____X_____X_____X_____\___
byte  stop  start   2     3     4     5     6     7     8     9   stop


              _____ _____ _____ _____ _____ _____ _____ _____ _____ _____
3rd       ___/     \_____X_____X_____X_____X_____X_____X_____X_____X_____\___
byte  stop  start   10    11    12    13    14    15    16    17  stop
```

Other than for these details of bit measurement and assembly, each 18 bit word is interpreted in the context of exactly the same Common Boot Stream Format as is described at the start of this paper, and which applies equally to the SPI, Asynchronous and Synchronous bit level protocols. Other than bit assembly issues there is one more thing that differentiates the SPI protocol from all of the others. Only for the SPI protocol is the F18 acting as a protocol master. For all of the others it is the remote system that chooses when to send each word and how fast to send each bit. Even the stream content is remotely decided because, unlike for the SPI protocol, no read command is used to issue a starting address. This lack of speed enforcement and complete dependence upon the external source is what makes the other protocols inherently better for interactive debugging using external, software driven systems, that have a less manically diligent focus on the task at hand than what an F18 is capable of.

3.1 ROM Specific Details

Pin 17 is input and pin 1 is output. The Asynchronous Boot ROM resets to a multiport read (F18A) or execute (F18B) that includes pin high wakeup. If the pin is seen going high, signaling the leading edge of a possible start bit, the F18 starts timing its duration. If the line remains high for 262144 cycles of a timing loop (≈4.1 mS on F18B), the booting effort is abandoned and the node reverts to its default multiport execute. The present code does not listen to comm. ports while this is going on nor while it is attempting to receive boot data once it has decided to start doing so. It's also running continuously in the process, so the async receive routine can be considerably optimized in the future to reduce power requirements.

4. 2-Wire Synchronous Boot

The synchronous protocol uses two wires, both of which are under the control of the remote system. When the F18 node wakes up from reset it immediately goes back to sleep on a port read that includes the clock input pin direction. When the clock goes high (or is high at reset) then the node wakes up, reads the port bit as high and enters the synchronous protocol control loop. The F18 node will not leave this loop until told to do so by the content of the protocol itself. At the instant that the protocol so concluded, the clock line will be a high input and the data line will be an input of unspecified value. If the F18 node gets back to its "warm" loop with the pins in this state, it will re-enter the synchronous protocol control loop.

After the clock line is detected as high by the F18, the remote system will set a data value and then drive the clock line low. After about 20 to 25 ns the remote system can change the data line again and drive the clock high. After nine low pulses on the clock line, 18 bits of data will have been assembled by the F18. The data are transmitted high bits first. Between the transmission of 18 bit words there should be a delay of at least 50 ns or more to allow time for the F18 to pass a words of data off to another node and get back. The interpretation of the word data are specified in the section on Common Boot Stream Format.

4.1 ROM Specific Details

Pin 17 is clock and pin 1 is data. The Synchronous Boot ROM resets to a multiport read that includes pin high wakeup. If the clock pin is seen going high, the F18 makes a reasonableness test by measuring how long it stays high. If the line remains high for a period longer than 261000 cycles of a timing loop (≈4.1 mS on F18B), the booting effort is abandoned and the node reverts to its default multiport execute. The present code does not listen to comm. ports while this is going on nor while it is attempting to receive boot data once it has decided to start doing so, although it will absorb and ignore any port data sent to it with probable loss of synchronization with the data stream. It's also running continuously in the process, so the async receive routine can be considerably optimized in the future to reduce power requirements.

5. 1-Wire (Quasisynchronous) Boot

This new protocol is designed to internal, rather than industry, standards and takes full advantage of the timing and characteristics of F18 technology to provide a very fast and efficient way for GreenArrays chips to communicate with each other using a single wire. The single wire may be operated as a half duplex high speed communication channel; and, thanks to the high impedance and low capacitance of our inputs, it is possible to DC isolate two GreenArrays chips using nothing more than a ≈10 pF series capacitor in the 1-wire communication line. Running at lower frequencies, it is less sensitive to transmission line characteristics than is the SERDES.

Each bit consists of a short or long high pulse, followed by a low period on the line. Data are transmitted most significant bit first, 18 bits per word. The interpretation of the word data are specified in the section on Common Boot Stream Format.

Here is a reference transmit timing routine and the resulting measurements for F18A and F18B at given Vdd, using GA144-1.10 and GA4-1.2 as test cases with high/low transitions at approx 0.800 volts:

```
1312 list
1wire xmit speed test 0 org
bit n . . . . -if over dup 2/ !b !b 2* ; cr
then over 2/ !b 2* over !b ;
xmit 006 kn-k 17 for bit next drop ;
run 00b io b! -2
spin 0 -1 xmit spin ;
```

Measure	Units	Typical F18A at Vdd of			Typical F18B at Vdd of		
		1.6	1.8	2.0	1.62	1.8	2.0
0 pulse	nS	4.1	4.0	3.75	4.0	3.8	3.7
0 period	nS	52.5	45.4	40.4	43.3	38	34
0 freq	MHz	19	22	24.7	23.1	26.3	29.4
1 pulse	nS	12.7	11.5	10.5	11.1	10.1	9.2
1 period	nS	54.6	47.1	42	45.4	40	35.8
1 freq	MHz	18.3	21.2	23.8	22	25	28

Final specified values and limits will be published after we have statistical data from testing large numbers of production chips, as will tolerances of transmitter/receiver combinations across supply voltage ranges.

5.1 ROM Specific Details

Pin 17 is input. The boot ROM begins processing as soon as it sees a low to high transition on the input line. In chips based on the F18B, the boot routine is capable of executing instructions delivered on its ports at all times in this process. In those that use the F18A, rules differ and have not been fully implemented as yet.

6. SERDES Boot

Since the SERDES executes code directly from its receiver port, it does not process boot frames as such. Furthermore, there are special requirements for managing S and T during receive operations. For these reasons, the SERDES is not directly involved in these protocols.