GreenArrays®

Application Note AN010 *Revised 6/04/13*

The Snorkel Mark 1: A Programmable DMA Channel

This paper describes a simple SRAM master that moves arbitrary sequences of 18-bit or 16-bit data between one of its COM ports and specified areas of the external SRAM. This is done by a simple, software-defined special purpose processor that executes a stored program from SRAM to define the sequence of transfers.

The problem and its solution are not new. In fact, very similar Programmable DMA Channels were part of the hardware in mainframe computers of the 1960's. In later minicomputer and microprocessor designs, the DMA channels lost the ability to execute stored programs and instead single transfers were controlled with registers and main CPU software. More recently this problem is addressed by giving DMA capability to each controller attached to a common system bus such as PCI. However, the Programmable DMA Channel remains a very useful tool, and one of the nicer things about the GreenArrays architecture is the ability to implement such devices in software.

This module illustrates a number of effective techniques for minimizing the size of the F18 code; indeed, it was necessary to employ them in order to fit this functionality into only 64 words. Anyone interested in learning to program our chips well should benefit greatly from a careful study of this software.

Related Documents

AN003: SRAM Control Cluster Mark 1
AN011: Ganglia Mark 1
DB006: polyFORTH Supplement for G144A12

Contents

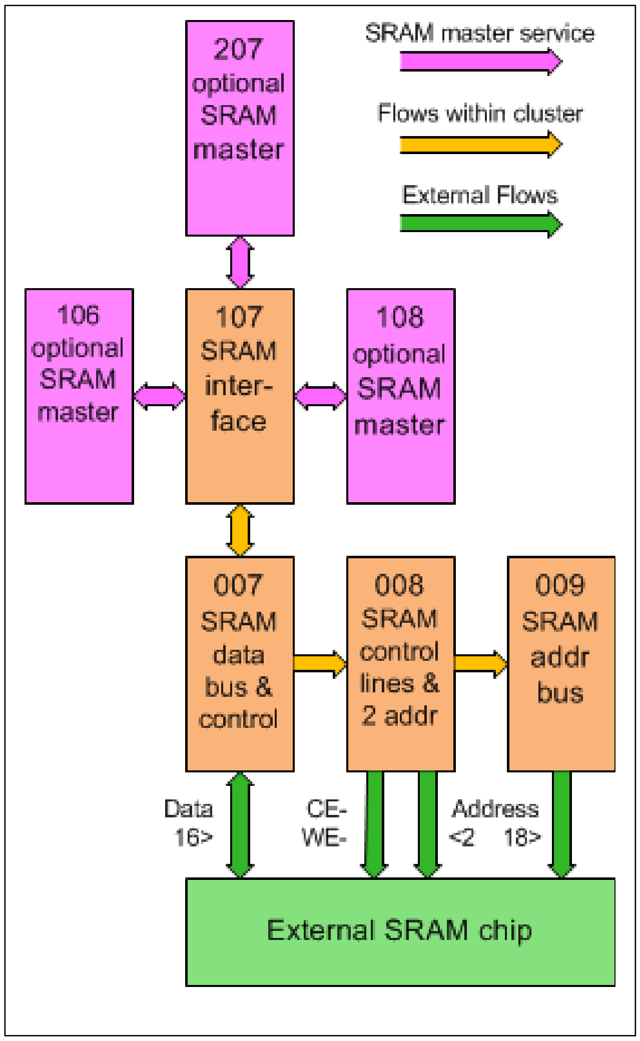
1. Problem Statement

Many applications require transactions that involve one or more movements of data between external memory and some destination or device. The transfers may include a single datum or an arbitrarily large block of data. It may be necessary to make multiple transfers for each transaction; for example, a simple SCSI operation requires sending a packet of data representing a command structure, then sending or receiving one or more blocks of data, and finally receiving a status structure. It would in many cases be inconvenient to make all of these subsidiary transfers to or from a single contiguous area of memory, even if the data direction were the same for all of them. Further, in our systems with 18-bit internal operations, some transfers will need to be moving 18-bit data to and from external memory, while others that terminate in structures or protocols based on 16-bit data will need to transfer to and from external memory in those units and alignments.

It is easy to accomplish this using programmed I/O. However, this cannot be done at memory speed in a high level external virtual machine environment, and is even slower when multiprogramming response times prevent a program from simply bursting an entire transfer in or out uninterrupted. Making transfers at memory speed generally requires something nearer the hardware than programmed I/O. We call this mechanism a Snorkel, suggesting a means whereby data or software immersed in the external SRAM may reach the "fresh air" in our fabric of high speed F18 nodes.

2. Our Solution

In the 1960s, many mainframe computers addressed this problem with *Programmable DMA Channels*. The terms varied but the basic idea was to build a hardware device that acted as a memory master and could transfer data at memory speed. The simplest just transferred one block of words or bytes in a given direction and required program intervention to transfer another. The more powerful executed what some called a *channel program.* In all cases the work of transferring data between memory and devices was offloaded from general purpose CPUs to special-purpose hardware that was optimized for making such transfers efficiently with minimal program intervention.



The speed difference between an F18 node and the external SRAM is great enough that we have the luxury of creating in software simple "devices" that fulfill roles which would otherwise require custom hardware. Because this is easy to do, there is no need to solve hypothetical future problems with current work. In general, solving only the problem at hand conserves all relevant resources, such as time, space, energy and programming labor. In addition there is no need to compose entirely hypothetical test cases.

The Snorkel is a simple, special purpose processor that executes stored *Snorkel programs* from external SRAM which tell it to perform an arbitrary sequence of 18- or 16-bit transfers to or from arbitrary areas of external memory. It occupies one of the SRAM master nodes and moves data between SRAM and any of the COM ports of the Snorkel node.

3. Implementation

This program, taking exactly 64 words of F18 RAM, implements our Snorkel in any SRAM master node shown above.

```
to activate snorkel wait till mmptr zero then
write addr of a sequence/x to it and stim.,
mmptr zeroed after stim and you may queue 2nd
x by writing and posting another stim. mmptr,
defined in idle, 2 for node 108, 4 for 207.,
,
x has 2 wd port address followed by one or mor
e function/arg groups. all x*must*be in bottom
64k of memory!,
,
function/arg group is 1 word jump adr in snork
followed by args if any. funcs are...,
x/r 16/18 count18, adr20,
fin done-flg awaken?,
,
if focusing call desired it must be first word
in first transmit. nothing about this code,
depends on using ganglia ... can source ide or
any other reasonable protocol including direct
stream or ad hoc node to node transfer.
```

```
408 list
- snorkel reclaim 207 node 0 org
mk! fp'm
/! an-a 00 push dup - !b over - !b pop !b ;
a+ 03 ap-ap+ push 1 . + 10000 over and if,
...or 1 then pop . + ;
+@ 0B a-an a+*/@ 0C a-an dup !b over !b @b ;
@w 0E a-a'u /@ 7 for 2* 2* unext,
...push +@ pop or ;
dma 14 x/r a+ @w push +@ push +@ pop begin,
...pop pop dup push over push push drop ex,
next pop drop drop drop func +@ push ;
x16 20 x dma /@ ! a+ ;
r16 23 x dma @ /! a+ ;
x18 26 x dma @w ! a+ ;
r18 29 x dma @ dup push 7 push,
...begin 2/ 2/ unext 3 and /! a+ pop /! a+ ;
fin 33 x dup /! +three dup - 8000 mk!
idle 37 begin @b 2 4 dup dup or /@ until,
...push dup /! pop over @w a! func +@ push ;,
40 1605 bin reclaim exit
```

The program is entered at **`idle`** with **`b`** initialized to the address of the port leading to node 107. For example, to load this program (note that its object code is in bin 1605) into node 207, the following descriptor suffices:

```
snorkel mk1 207 +node 1605 /ram up /b 37 /p
```

On boot, and after completing each program, this node awaits with **`@b`** a stimulus from the SRAM interface. To start a program, some other entity writes the 16-bit cell address of a Snorkel program into a cell **`mmptr`** (an agreed-upon address in external SRAM, 4 in this case) and then sends a stimulus to the Snorkel through the SRAM interface. The node swaps a zero into **`mmptr`** indicating that it has retrieved the pointer, which it then uses to fetch and execute Snorkel Program instructions. Execution continues until a **`fin`** instruction is encountered, whereupon the node stores zero where the **`fin`** instruction came from to show completion, and sends a stimulus to another node (in this case node 106 of the Virtual Machine, determined by the value **`x8000`** passed to **`mk'`** in **`fin`**) to awaken it in case the node was able to suspend while waiting for Snorkel completion. The Snorkel program structure and instruction set is as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Function
0														hi		Address of port to use (occurs once)
low																followed by one or more 5-cell instructions structured thusly:
0						Opcode: Addr of F18 routine										**Opcode:** Address of F18 routine **o16**: Send 16-bit data **i16**: Receive 16-bit data **o18**: Send 18-bit data **i18**: Receive 18-bit data **fin**: Stimulate a selected master, stop and await new program
0														hi		transfer size (words+1 thru port) 18-bit count, max 262144 words
low																
0												hi				20-bit SRAM address for transfer
low																

Decoding is simple; after reading an op-code from external memory, we simply jump to that address in internal RAM using the sequence **`push ;`** at the end of **`idle`**. We use this technique in many ways. Another technique example is the calling of **`dma`** to run a loop that invokes the code following the call each time through the loop. Such techniques reduce code size and make it feasible to express programs like this one in only 64 words.

4. Usage Examples

polyFORTH mechanism for operating a Snorkel in node 207 begins with that required for SPI flash I/O in the nucleus:

```
2496
Base mechanism for I/O done by memory mastering nodes.
Masters 0 and 1 are nodes 108 and 207 respectively.
MMPTR  double address pointer for each master's next command.
   Set zero after master fetches it.
MMBITS  mask bits for masters.  MMpF is our VM's bit.
MMASK  is the currently active mask in memory interface node.
!MMASK  sets master mask
!MMSTIM  sends stimulus to each master whose bit is set.
SUSPEND  suspends VM execution until stimulated.

+SNORK  starts a snorkel program, returning complete flag addr.
sDONE  waits till the given completion flag says done.
HOME  unmasks the snorkel so it may receive stimuli.

RELOAD  disables all other masters.  More specific shutdowns
   should be defined later (and thus executed first.)
```

```
96
( Snorkel based I/O)
HEX  8000 CONSTANT MMpF   CREATE MMBITS  800 , 200 ,
   CREATE MMASK  0 ,   2 CONSTANT MMPTR   | 4 CONSTANT MMP2
: !MMASK ( m)   DUP MMASK !  1 0 MK! ;   ( MMpF !MMASK)
: !MMSTIM ( m)   0 0 MK! ;   ( : SUSPEND   SUSPEND ;)

| 2145 CONSTANT :up    | 21D5 CONSTANT :right
| 2115 CONSTANT :down | 2175 CONSTANT :left
   20 CONSTANT o16  23 CONSTANT i16
   26 CONSTANT o18  29 CONSTANT i18  33 CONSTANT FIN
: +SNORK ( ^pgm ^fin - ^fin)   BEGIN  MMP2 @ WHILE  PAUSE REPEAT
   FIN OVER !  SWAP MMP2 !  MMBITS 1+ @ !MMSTIM ;
: sDONE ( ^fin)   BEGIN DUP @ WHILE
   ( SUSPEND)PAUSE REPEAT  DROP ;
: HOME   MMASK @ MMBITS 1+ @ OR !MMASK ;
HERE EQU reload ] MMpF !MMASK ( +RELOAD) EXIT [
```

The key functions are +SNORK and sDONE for starting Snorkel programs and waiting until they have completed. It's necessary to pass the address of each program's **fin** instruction to both of these words because **fin** is used not only as an op-code but as a flag.

+SNORK (^pgm ^fin - ^fin) spins until the preceding Snorkel program start address has been taken by the Snorkel. It then restores the **fin** op-code value in the program, sets its start address, and posts a stimulus for the Snorkel through the memory controller. If the Snorkel is executing another program at that time, the stimulus is preserved until the Snorkel node returns to its **idle** routine.

sDONE (^fin) spins until the given Snorkel program's **fin** instruction has been posted to zero.

With this logic, multiple tasks may use the mechanism without interference so long as they aren't talking to the same destination in such a way as to depend on destination state between transactions, or using the same Snorkel program or buffer areas in memory. In cases like those, it is up to the application to use appropriate facility reservation methods such as GET / RELEASE .

Some useful constants are defined here but others must wait, for space considerations, till block 28 has been loaded:

```
2428
Tools for extending the memory mastering (Snorkel/Ganglia) I/O
   that's in the nucleus.

:UP  and  :RIGHT  are port adrs for starting channel programs.

'reload  is a chain of executable code that returns when done.
+RELOAD  adds code to the reload chain.  Usage:
   HERE ] <stuff to do before existing chain>  +RELOAD [
```

```
28
( Snorkel/Ganglia)   HEX
HEX  2145 1 2CONSTANT :UP  21D5 1 2CONSTANT :RIGHT
     2115 1 2CONSTANT :DOWN  2175 1 2CONSTANT :LEFT

: 2, ( d)   , , ;   : W, ( n)   0 2, ;
: run   CONSTANT DOES  4 * R> @ OR ;
   0 run rr   1 run ll   2 run uu   3 run dd
: ,path ( n n n)   40 * OR  40 U* , OR , ;   DECIMAL

: +RELOAD ( a)   'reload @ \ AGAIN  'reload ! ;  IMMEDIATE

: DRIVE ( n)   DUP 20 < IF 2400  ELSE 480
     THEN ['] SHADOWS 1+ !  DRIVE ;
( Reset it)  OFFSET @ 1200 / DRIVE
```

The full set of constants is then:

:DOWN :RIGHT :LEFT Port addresses (calls) used to start programs for a Snorkel in node 207
:UP :RIGHT Port addresses usable to start programs for a Snorkel in node 108
o16 i16 Op-codes for transfers out of and into external SRAM as 16-bit data in RAM and as bits 15..0 of words transferred through the port. Count is number of SRAM words -1.
o18 i18 Op-codes for transfers of 18-bit data. Each 18-bit datum is stored in external memory as a polyFORTH double number (most significant cell first). Count is number of double numbers -1.
FIN Op-code for terminating a Snorkel program. The following four cells are ignored. The cell containing FIN is set to zero when FIN is executed.

Here are two examples of Snorkel programs; the first is from an application, the second is from the nucleus:

```
   ( MD5 Test) HEX
( *)HERE :DOWN 2,  1 , 2033 ,  1 rr 0 dd 0 ,path
   ( #reply) 7 W,  ( #payload) 0 W,  ( code) 1 , 2009 ,
CREATE pgm>  :DOWN 2,
   o18 ,  5 W,  ( *)W,
   i16 ,  7 W,  digest W,  HERE FIN ,
: MD5>  pgm> LITERAL +SNORK sDONE ;
: MD5 ( adr len)
   <MD5 DUP IF >MD5  ELSE  2DROP THEN  MD5> REPORT ;
: .TARE ( - d)  0 0 COUNTER D+ 2>R 2R> TIMER ;
: TARE ( - d)  0 0 COUNTER D+ 2>R 2R> COUNTER D- DNEGATE ;
: TEST ( adr len) CR ."   " 2DUP TYPE CR ."   "  TARE COUNTER
   D+ 2>R <MD5 >MD5 MD5> 2R>  TIMER REPORT ;
: -TARE ( adr len)   0 0 COUNTER D+ 2>R <MD5 >MD5 MD5> 2R>
   TIMER REPORT ;
```

```
2498
The SPI nodes, 705 with 706 and 704 available as helpers, rests
   in 3-port execute waiting for a pair of calls (focus and call
   to starting address at zero).
sGH  is a ganglion header that sets up the proper path with
   counts  gOUT  and  gIN , delivering these two calls.
Thereafter the destination node executes 16-bit SPI Programs.
   Data transfers are in bytes, on word boundaries, MSB first.
~SPI  does a transaction given snorkel program using sGH and the
   payload sizes, in words -1.  Note one word of payload in sGH.
spic  snorkel pgm for cmd/status funcs returning spiSTAT.
~sio  runs a command given channel compl cmd address, SPI
   program address, words out and words in.
~scm  sets channel program for write read and runs it.
~swr  sets for write write read and runs it.
~cmd  runs an SPI command transaction given spi program adr,
   words out and words in for whole transaction after sGH.
```

```
98
   ( SPI node 705)   HEX   2VARIABLE spiSTAT
CREATE sGH   1 , :down ,  1 , 2033 ,  2 uu 2 ll 1 uu ,path
   ( rsp) 0 , HERE 0 ,  ( pay) 0 , HERE 0 ,  ( ent) 1 , 2000 ,
   HERE sGH - 2/ CONSTANT /sGH  | CONSTANT gOUT  | CONSTANT gIN
: ~SPI ( ^pgm ^fin po pi)   gIN !  gOUT !  +SNORK sDONE ;

   3 CONSTANT sNUL    4 CONSTANT s-SEL   7 CONSTANT sCMD
   8 CONSTANT sOUT   1E CONSTANT sBYT   25 CONSTANT sWO

| CREATE sio   1 , :down ,   o18 , /sGH 1- W, sGH W,
   o16 , 0 , HERE 0 , 0 , HERE 0 ,  i16 , 0 , HERE 0 , HERE 0 W,
   HERE i16 , 0 W, spiSTAT W,  ( wrt) FIN ,  | CONSTANT sCF
| CONSTANT rda  | CONSTANT rdn  | CONSTANT cma  | CONSTANT cmn
| : ~scm ( ca pa nwo nwi)   i16 cma 1+ !  1- DUP >R rdn !
   DUP >R 1- cmn !  cma !  sio SWAP  R> R> ~SPI ;
| : ~cmd ( ca pa nwo nwi)   spiSTAT 0 rda 2!  ~scm ;
```

In the first example, pgm> is a Snorkel program. It uses the **down** port of the Snorkel node, so the transfers will be done between external SRAM and the **down** port of node 207. This means that the immediate "target" of the transfers will be node 307, communicating with that node through its **down** port. The first Snorkel instruction transfers six 18-bit values outward, into node 307, taken from 36 16-bit cells of SRAM starting at an address that was passed into the definition on the stack. The second instruction transfers eight 16-bit values inward from node 307 to eight cells of SRAM starting at the address `digest`. Finally the `FIN` instruction ends the program; HERE places its address on the compiler stack so that it may be used as the `LITERAL` in the definition `MD5>` which executes this Snorkel program and waits for it to complete. Note the use of `2,` and `W,` to lay down double and single precision numbers as 18- or 20-bit values in the Snorkel program. These words are only available after `HI`.

The second example shows a more complex Snorkel program at `sio`. It too uses node 307 as its immediate "target", performing four data transfers: An outbound transfer of `/sGH` 18-bit values starting at `sGH`; an outbound transfer of a variable number of 16-bit values from a variable address; an inbound transfer of a variable number of 16-bit values to a variable address; and an inbound transfer of one 16-bit value to SRAM at the address `spiSTAT`. Subsequent definitions in this block are used to set variable counts and addresses before executing the Snorkel program. Note that in the nucleus we use `:down` which is headless to conserve space.

All these examples are in context of Ganglion transactions, but the Snorkel is not in any way limited by Ganglion conventions. Data may be moved between memory and node adjacent to the Snorkel node for any purpose and using any protocol that has been mutually agreed upon with the adjacent node in question. Obviously, if the adjacent node is executing the port through which data are transferred to it at the time the transfer is made, then at least the first word transferred will be interpreted as an F18 instruction word.

4.1 Special Considerations

The high-order two bits of data transferred through ports on an **o16** operation will be zero on the EVB001 board because D16 and D17 pins are pulled down. These bits are not masked by the Mark 1 SRAM Cluster; if you have laid out a board you must be aware that any signals on those pins are visible to SRAM masters and in the case of the Snorkel will be moved through the port on an **o16** operation.

GreenArrays®

Application Note AN010 *Revised 6/04/13*

IMPORTANT NOTICE