GreenArrays™ Application Brief AB005 *Revised 11/06/17*

Delay Lines (or buffers)

Using one or more Nodes as 64-word Storage Elements

Most of the examples of controlling a chain of nodes through port execution, beginning with the example of boot streams, use a nest of streams whose overall length increases linearly with the number of nodes in the chain. This follows from the idea that the instructions and data given to each node must be part of the original stream.

We had a need for a long delay line buffer of 1920 words using 30 nodes. The application timing would not tolerate such nesting so we needed to find another way. (Delay line was preferable because there is no need in a straight buffer for random addressing, which would massively complicate the structure; it is sequential access all the way after all.)

It is easy to use all the memory in a node as a FIFO buffer for 64 words of data without using any code at all in ROM or RAM. This was recognized long ago and we do it all the time with a single node. However, we have learned that it is also possible to concatenate an arbitrary number ***n*** of such nodes for a FIFO of ***64n*** words, still without code in any of their ROM or RAM, and without incurring a time penalty in words per second throughput for any practical value of ***n*** .

The secret in this method is to have each node do all of its work via port execution, and to store the instructions that must be passed to the next node in sequence on the stack of each node so they do not have to be contained in the stream from predecessors. To insert a word into the FIFO it is written into a port of the first node surrounded by two instruction words. When that is done, the first node sends its "oldest" data word to the next node, surrounded by two instruction words identical with those inserted into it. At the end of the delay line, the "oldest" word in the entire delay line is passed to a receiving node through a COM port, surrounded by two instruction words that are simply discarded by the receiver. Thanks to circular stacks, each node initialized as a delay line has an infinite supply of these two instruction words on its stack for transmission to the next node. Maximum throughput is on the order of 5 memory cycles or 25 ns per payload word pushed through the delay line. In one application sustained rates on the order of 50 ns per word were easily achieved. To move data, the source node must insert its three words, and the destination node must remove them.

Each delay line node is initialized with **P** pointing to the source(s) of data, **A** set to zero for RAM pointer, and **B** pointing to the next node in the line or to the final data receiver. Each node's stack is initialized with instruction word pairs, W1 and W2 below, with W1 on top. The sequence pushed into the first node consists of the following, in arrayForth-3 Syntax:

```
A[ !b @p @ ]]  W1 (x9D0A)  Sends W1 from stack to the next node, fetches new word from
       port and oldest word from SRAM.
<the new data word>
A[ !b !+ !b ]]  W2 (x9822)  Sends oldest word to next node, replaces oldest slot in RAM
       with new word, advances SRAM pointer and sends W2 from stack to the next node.
```

Code to push a word into the delay line might look like this, with **A** pointing to the entrance COM port:

```
: >dly ( n)   A[ !b @p @ ]] lit !  !  A[ !b !+ !b ]] lit ! ;
```

At the receiving end, code to receive a data word via the COM port **A** points at might look like this:

```
: <dly ( -n)   @ drop  @  @ drop ;
```

A host machine FORTH definition to make boot descriptors for a delay line node ***nn*** with input port ***i*** and output ***o*** might be:

```
: +DLY ( i o nn)   DUP +NODE  /B /P  A[ !b !+ !b ]]  A[ !b @p @ ]]
   2OVER 2OVER  2OVER 2OVER  2OVER 2OVER  2OVER 2OVER  10 /STACK  0 /A ;
```

The boot descriptor for a node like 308 receiving input from 307 and sending output to 309 might be:

```
LEFT RIGHT 308 +DLY
```

There are other arrangements of opcodes that may be used in the two key instruction words above. These two particular arrangements were optimized for throughput by avoiding back-pressure in the pipeline.