GreenArrays™ Application Brief AB004 *Revised 10/21/14*

Port Execution

Communication Between two GA144 Nodes

Learning to understand and use port execution is perhaps the most important rite of passage when learning to program a GA144. The arrayForth program below is a simple demo showing how one node can use a neighbor's memory for data storage.

```
820 list
64 word array in neighbor 608 node 0 org
set 00 a @p ! ! ; .. @p a! ..
@next 02 -n @p ! @ ; .. @+ !p ..
!next 04 n @p ! ! ; .. @p !+ ..
fetch 06 a-n set @next ;
store 08 na set !next ;,
,
test 0A 0 dup set
loop dup 1 . + push,
..63 for dup !next 1 . + next,
..63 for @next next pop loop ; 1A
```

Concentrate on the word `set` first. The first four instructions fill a single 18 bit cell and are all that this word does. `@p` reads the next 18 bit cell and places it on the stack. In this case that happens to be an instruction word containing `@p a! ..` . This is not executed here but is to be passed as data to a neighbor node.

The next instruction is `!` which writes the instruction word already placed on the stack into the port addressed by register A. The neighbor who shares this port can either read it as data or execute it, depending on whether it is suspended on a read of the port or on a jump to the port. In this case we expect that the neighbor has jumped to the port, awaiting instructions to be executed from that port only (the neighbor must not be executing a multiport address).

The third instruction is also `!` . This writes the address that was already on the stack into the shared port. If the neighbor reads that shared port then the address will be transferred onto the neighbors stack.

The final instruction is `;` which returns to the word which called `set` . Note the yellow `..` after `;` . This is to ensure that the next instruction starts in slot zero of the next available word. Being yellow is a convention that tells us the next word is meant to be executed in a neighbor rather than in this node. In arrayForth, as opposed to PC colorForth, the yellow and green words behave in the same way so a green `..` would have worked just as well.

The instruction word `@p a! ..` when executed by the neighbor will receive the address from the port and place it into the A register. The word `set` then is used to set a neighbor's A register. Notice that all code executed by the neighbor comes through the port, none of it resides in the neighbors memory. This leaves all 64 words of neighbor memory free to be used for data storage.

Now that you understand `set` it should be clear how `@next` and `!next` work. `@next` executes `@p ! @ ;` sending an instruction word via `!` and receiving a value via `@` . The neighbor executes `@+ !p ..` , which fetches a value from memory via the A register, increments the A register, and sends the value back through the port. `!next` is nearly the same, but in this case a value is first sent to the neighbor, then stored via the neighbor's A register, which is post incremented.

The words `fetch` and `store` use the previously defined words `set` , `@next` , and `!next` to set the neighbor's A register before sending or receiving a value. The word `test` can be used in the simulator to watch memory being written and read using these techniques. In order to run this in softsim you'll need to compile the code, provide boot descriptors, load them from block 216 as shown below, and of course run softsim by typing "so".

```
200 list
user f18 code 820 load exit

822 list
boot descriptors,
609 +node right /p,
608 +node right /a 608 /ram a /p

216 list
softsim configuration 822 load exit
```

For more information see http://www.greenarraychips.com

In particular, DB004 arrayForth User's Manual.