Choreographing Teams of Fast, Low-Power Computers

GreenArrays® Staff
SVFIG Forth Day
17 November 2012
The Best Known Energy Efficiency in a Commercially Available Chip.
Progress Since Nov 2011

- Many evaluation boards and chips shipped
- Numerous new documents published
- arrayForth® Institute created
- polyFORTH® system, running in Fall 2011, documented and released in Sep 2012
- Ethernet NIC implemented in Jan 2012, most of TCP/IP stack converted in Feb 2012, life testing underway
- Chip design improvements continue
Promising Developments

- Engagement with technologists developing new manufacturing methods for the “Internet of Things”
- Excellent study by a major company confirms the GA144 to be vastly superior to even the latest TI MSP430 in energy efficiency
- Engagements with several potential customers who need higher performance at lower power than is otherwise possible
Problems Encountered

- Slow acceptance of novel hardware
- Lack of funds limits marketing efforts
- Legal harassment by Daniel E. Leckrone has wasted our time and money
Plans for 2013

- Development roadmap (see website)
- Moving development tools onto the chip – two approaches, polyFORTH based and Chuck’s etherForth
- Selective creation of product companies and funding them (VC, Angel, KickStarter)
Acknowledgments

- App notes created by Peter Milford and Stefan Mauerhofer
- Advanced programmer interface concepts by Robert Patten
- Documentation reviews by David Stubbs who has also salvaged all of Jeff Fox’s disk drives
Plan for the Afernoon

- Quick review of GreenArrays Architecture
- Short (10 minute), sweet topics with Q&A encouraged after each topic
  - High-level design for several applications
  - High-level development and debugging techniques
- Chuck’s Fireside Chat
For More Information on GreenArrays

- **Primary Website**
  - [http://www.greenarraychips.com](http://www.greenarraychips.com)

- **arrayForth Institute**
  - [http://school.arrayforth.com](http://school.arrayforth.com)

- **Announcement Blogs**
  - Business [http://www.greenarraychips.com/blog1](http://www.greenarraychips.com/blog1)
  - Technical [http://www.greenarraychips.com/blog2](http://www.greenarraychips.com/blog2)

- **Tech Support on e-mail, Skype, Phone**
F18A Architecture Review

John Rible and Greg Bailey
The Production G144A12 Chip

- 144 F18A computers in 8 rows, 18 columns
- Each talks to its adjacent neighbors
- 22 edge nodes have I/O pads
- Max \( \approx 100 \) GOPS at \(<1\) Watt
- In production. Sample kits and chips shipped for 1 year, quantity orders welcome
F18A Technology

- Easily configured arrays of computers and I/O
- Each 18-bit asynchronous computer is self-contained
  - RAM, ROM and registers in a single address space
- Instant suspension/resumption per computer
- High performance (≈666 MIPS/node)
- Low energy per unit work (≈7 pJ/instruction)
  - No power or energy cost per MIP, only per unit work; typically low duty cycle
- Multilevel programming
F18A Computer

- 241 \( \mu \text{m} \times 523 \, \mu \text{m} \) in 180nm CMOS process
- Dual stack architecture
- 8-element circular stacks
- Archtypical Forth ALU
- 5 specialized registers
- Memory balanced for speed and power
- Up to 5 comm ports
- Optional I/O
Coordinating Computers

- Fast, simple, synchronized comm ports
  - Passing instructions and/or data bidirectionally
  - Transparent handshaking
    - Automatic suspension with no races
  - Port execution
    - Simple protocols – use instructions, not codes
- Multiport operations
  - Up to five other computers when rules are followed
  - Multiplexing / demultiplexing data streams
Managing I/O

- Software-defined pin behavior
  - GPIO Pins: Fine control, reads actual pin state
    - Nodes may have up to four pins
  - Bidirectional parallel buses
  - Analog I/O

- High speed SERDES (~600 Mbit)
For More Information on Architecture and Chips

- Documentation on website
  - DB001: *F18A Technology Reference*
  - DB002: *G144A12 Chip Reference*

- arrayForth Institute
  - PROG0100: *F18A Architecture and Instruction Set*
  - PROG0200: *F18A Programming Techniques*
    (course not released yet)
Multilevel Programming

Greg Bailey
Three Basic Methods

- Microcode: Application modules consisting of native F18 code residing in one (or more) computers.
- Streamed port execution of larger programs fed by a memory resource
- Virtual machines running from external memory, implemented by a team of computers
Code Generation

- Applicable to any of the three basic programming methods
  - Hand-crafted software
  - Semiautomatic programming
    - Interactive synthesis: Analog block diagrams
    - Automatic programming: Ras Bodik and colleagues
Today’s Emphasis

- Organizing teams of computers
- Using high level tools like polyFORTH for debugging and managing of such teams

Previously we have concentrated on the fine details of F18 programming...

Today we take a macroscopic view of creating applications
For More Information on Programming Methods

- Documentation on website
  - DB004: arrayForth User’s Manual
  - DB005: polyFORTH Reference Manual
  - DB006: polyFORTH Supplement for G144A12
  - Paper on Boot Protocols
  - Paper on Getting Started with eForth
A Team to Control an SRAM

John Rible and Greg Bailey
Goals for This Particular Model

- Enabling external memory access
- Support for multiple masters
- Not specialized ... random access sequences
- Not fully optimized
Four Nodes, Three Clients

- Three worker nodes dictated by chip geography
- Interface extended to node 107 to facilitate large teams
- Tradeoffs – more clients increase latency
Functions

- Arbitrates between up to three clients and provides five atomic functions:

<table>
<thead>
<tr>
<th>Words Received</th>
<th>Reply Sent</th>
<th>Function Performed</th>
</tr>
</thead>
<tbody>
<tr>
<td>+p4</td>
<td>+a16</td>
<td>w16 e@ Read a word from SRAM at p:a</td>
</tr>
<tr>
<td>-p4</td>
<td>-a16</td>
<td>w16 e! Write a word into SRAM at p:a</td>
</tr>
<tr>
<td>-n16</td>
<td>+p4</td>
<td>w16 f16 Compare-and-exchange. Write w to SRAM iff current value = n Return x0FFFF if stored or 0 if not.</td>
</tr>
<tr>
<td>+x</td>
<td>-0</td>
<td>m16 mk! Set master enable mask</td>
</tr>
<tr>
<td>+x</td>
<td>-1</td>
<td>m16 mk! Post stimuli for master(s)</td>
</tr>
</tbody>
</table>
For More Information on External RAM Control

- Documentation on website
  - AN003: SRAM Control Cluster Mark 1

- Source code
  - arrayForth 2a blocks [270..280] SRAM
  - arrayForth 2a blocks [1320..1328] Partial ROM support for SDRAM
A Team to Implement a Virtual Computer

John Rible and Greg Bailey
eForth/polyFORTH VM

polyFORTH Virtual Machine for GA144-1.2
Choreography

- Stack node responsible for data stack, memory access and operations predominantly using the data stack.
- Bitsy node responsible for return stack, instruction stream fetch and decode, and ops predominantly related to these things.
- Neighbor “buds” available for expansion or application specific instructions.
### Instruction Set

<table>
<thead>
<tr>
<th>Function</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Call pFVM code in low external RAM</td>
<td>00ea</td>
<td>High Level Execution token (external RAM address)</td>
</tr>
<tr>
<td>Call F18 definition in Bitsy node 105</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>Call F18 definition in node 205 through up port of bitsy node</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>Call F18 defn in node 005 down bitsy</td>
<td>1111</td>
<td>F18 RAM/ROM/Port address</td>
</tr>
<tr>
<td>Call F18 definition in Stack node 106</td>
<td>0000</td>
<td></td>
</tr>
<tr>
<td>Call F18 defn in node 206 up stack</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>Call F18 defn in node 006 down stack</td>
<td>0011</td>
<td></td>
</tr>
</tbody>
</table>
For More Information on These Virtual Machines

- Documentation on website
  - DB006: polyFORTH Supplement for G144A12

- Source code
  - arrayFORTH 2a blocks [360..478] for pF VM
  - arrayFORTH 2a blocks [1080..1198] for eF VM
The Snorkel: A Programmable DMA Channel

Greg Bailey
Goals for the Snorkel

- Move arbitrary 16- or 18-bit data between external SRAM and one of Snorkel’s ports
- Independently execute a simple program from SRAM
- Started by stimulus from another master
- Gets program start address from an agreed SRAM cell
- Implements streamed port execution programming method
### Snorkel Program Structure

<table>
<thead>
<tr>
<th>Function</th>
<th>Opcode: Addr of F18 routine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Address of port to use (occurs once) followed by one or more 5-cell instructions as follow:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Opcode: Address of F18 routine</td>
</tr>
<tr>
<td>0</td>
<td>o16: Send 16-bit data</td>
</tr>
<tr>
<td>0</td>
<td>i16: Receive 16-bit data</td>
</tr>
<tr>
<td>0</td>
<td>o18: Send 18-bit data</td>
</tr>
<tr>
<td>0</td>
<td>i18: Receive 18-bit data</td>
</tr>
<tr>
<td>0</td>
<td>fin: Stimulate a selected master, stop and await new program</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>18-bit transfer size (words thru port)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>20-bit SRAM address for transfer</td>
</tr>
</tbody>
</table>
For More Information on the Snorkel Mark 1

- Documentation on website
  - AN010: *Not yet published.*

- Source code
  - arrayFORTH 2a block [408]
  - polyFORTH 2a blocks [96, 28]
A Surface of Ganglia: Dynamic Message Routing

Greg Bailey
Problem Statement

- Route messages between arbitrary nodes
- Each message is a simple exchange between SRAM and a given port of a given node
  - Deliver x-word payload, receive y-word reply
- Employ unoccupied nodes for routing
- Path exists only during exchange
- Support long (262k-word) messages
  - Any combination of code and/or data
Chosen Solution

- Define a frame whose header holds source routing, updated as the frame moves incrementally between nodes.

- Program each node by default with the ganglion program that interprets header and:
  - Updates header and moves frame to next node.
  - Or delivers payload and receives reply back down same path.

- Resulting surface of ganglia can connect any two ports if a path is possible.
## Ganglion Frame Structure

<table>
<thead>
<tr>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Function</td>
<td></td>
</tr>
</tbody>
</table>

- **Function:**
  - Focusing call (updated for each port crossing)
  - Included in payload delivery
  - Call to pump routine in ganglia
  - Header - Focusing call and encoded path updated at each step
  - Encoded path remaining (list of direction/distance)
  - Header stripped when delivering payload at destination
  - reply count, Y-1
  - payload count, X-1
  - Payload (X words)
  - Sent with outbound frame and delivered after focusing call at destination
  - Reply (Y words)
  - Transferred from destination back to originator

- **Path encoding will be upgraded for versatility**
Procedure for Use

- Build snorkel program that transmits frame and receives reply
- Build frame and execute program
- Snorkel program may assemble (gather) frame from components and may disassemble (scatter) reply as desired
For More Information on Mark 1 Ganglia

- Documentation on website
  - Mark 1 will be obsoleted, probably by 2b.
  - AN011: *Mark 2 Ganglia, not yet published.*

- Source code
  - arrayFORTH 2a blocks [404..406]
  - polyFORTH 2a blocks [97, 28]
Implementing polyFORTH on a Virtual Computer

Greg Bailey
Motivation

- Automated Test Equipment (ATE) testing of a chip with 144 computers
- Develop and test software on chip without dependency on a host computer
- Cross-develop into target chips with very high speed and the ability to generate test stimuli and probe responses at low cost
Chosen Solution

- Port polyFORTH to a VM running on chip
  - Thanks to FORTH, Inc. for its kind permissions
  - Well-documented, robust development system
  - Suitable for any application
  - Target compiler and full source provided

- Best tradeoffs for versatile 16-bit model
  - Very similar to our system for original Novix chip
  - Will be able to compile and debug F18 code
polyFORTH Memory Model

Callable VM code, TERMINAL task dictionaries
Executable VM code, BACKGROUND dictionaries

<table>
<thead>
<tr>
<th>Main</th>
<th>High</th>
<th>Extended</th>
</tr>
</thead>
<tbody>
<tr>
<td>C@ C!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>@ ! DUMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X@ X! XDUMP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

32 kwords 32 kwords 960 kwords
0.0000 0.8000 1.0000 F.FFFF
Resulting System

- Solid development platform
- Performance governed by memory access time and choice of “instruction set”
  - True of any VM, note benefits of dual stack arch
  - Result on the order of DEC 11/73 or VAX 780 at cost of about 5 mA when polling nodes eliminated
- VM may be extended with application specific instructions
- Plenty of room for enhancement with VM instructions and improved memory subsystem
For More Information on G144A12 polyFORTH

- **Documentation on website**
  - DB005: *polyFORTH Reference Manual*
  - DB006: *polyFORTH Supplement for G144A12*

- **Source code**
  - arrayFORTH 2a blocks [360..478] Virtual Machine
  - polyFORTH 2a full system and utilities
    - blocks [51..59] Target Compiler
    - blocks [60..119] Nucleus Source
Memory Mastering I/O for polyFORTH

Greg Bailey
Trivial I/O to Manipulate Pins

- Use Simple Ganglion frames to set io register using port execution, no RAM code required in the nodes owning the pins

- Examples are:
  - Setting pin 600.17 to select SPI devices
  - Resetting target chip using pin 500.17
Ad Hoc Fetch/Store

- R@ (d a) R! (a − d) R!@ (d a − d) in any listening node
- Memory, ports, io register
- Used for simple I/O exploration, see app notes
- Operational use when speed unimportant
SPI Flash and MMC

- Bus support code in node 705 specific to device protocol type (protocols very different)
- Macro operations defined by streaming port execution using ganglion frames
- When switching devices drop new code into node 705 using a ganglion frame
Other Bus Masters

- Ideal I/O for this sort of implementation uses shared memory structures and stimuli
- Present chips require a polling node whenever sources are combined

Lesson Learned
- Future chips will poll at very low power
- “Warp” ports, for more flexibility in floor planning, are probably worth their cost
For More Information on I/O Using Snorkel & Ganglia

- Documentation on website
  - DB006: *polyFORTH Supplement for G144A12*

- Source code
  - arrayFORTH 2a blocks [410..412, 774..776]
  - polyFORTH 2a blocks [98..101] SPI mass storage
    - blocks [31..32, 121..122] external frequency refs
    - block [142] ad hoc memory/register fetch/store
Using polyFORTH to Explore a 3-Axis Accelerometer

Peter Milford
For More Information on This Exercise

- Documentation on website
  - AN008: *Exploring a 3-Axis Accelerometer*

- arrayForth Institute
  - APP0100: *Application Notes*

- Source code
  - polyFORTH 2a block [142]
Using polyFORTH to Develop a Software-Defined Interface

Stefan Mauerhofer

presented by Greg Bailey
Talking with an AT Keyboard

- 5V open-collector bi-directional interface
- Select a suitable level shifter chip
- Take 5V supply from convenient USB interface on EVB001 Evaluation Board
Hardware Prototype
Used arrayForth IDE to Explore

- With a scope attached to IO pins 317.17 and 217.17 the keyboard can be exercised and its behavior observed
- ArrayForth code is written and tested to allow reading and writing of the keyboard using nodes 316 and 216 as buffers
- Simple testing is performed “by hand”
Higher Level Testing with polyFORTH

- Snorkel/Ganglia path to buffer nodes
- Read and write buffers using ganglia words R! and R!@ with paths to nodes 315 and 215
- Build up and test a higher level interface using keycode tables and meta keys
- Use a polyFORTH background task to control the keyboard
- Replace serial terminal ‘KEY vector with the attached PS/2 keyboard
For More Information on the AT Keyboard Project

- Documentation on website
  - AN009: Attaching a PS/2 Keyboard

- Source code
  - polyFORTH 2a block [142]
Using polyFORTH to Test and Validate an MD5 Team

Charley Shattuck
Node Diagram of MD5 Module
: MD5 (a n)  MD5 MD5> REPORT ;

- polyFORTH code is factored into three snorkel/ganglia transactions plus REPORT.
- <MD5 gets the module started
- >MD5 feeds a string of bytes to the module
- MD5> stops the module and reads back the message digest
- REPORT displays the message digest in a standard format
Timing Test Results

```
 ok 0 0 MD5
d41d8cd98f00b204e9800998ecf8427e ok
a TEST
 a 301.0
 0cc175b9c0f1b6a831c399e269772661 ok
abc TEST
 abc 301.1
 900150983cd24fb0d6963f7d28e17f72 ok
md TEST
 message digest
 302.6
 f96b697d7cb7938d525a2f31aaf161d0 ok
abc.. TEST
 abcdefghijklmnopqrstuvwxyz
 304.0
c3fcd3d76192e4007dfb496cca67e13b ok
MILLION 899862.0
 fd482807dd4e495722d54a39150b5a1f ok
FASTER MILLION 809948.0
e491c46c962e2c24f0e71cbb0987a616 ok
```
Timing Test

- Hashes 1 million bytes in : 900 ms
- With virtual CPU turned off : 810 ms
- 386DX25 : 1303 ms
- 386DX40 : 776.5 ms
- 386 code was generated inline by assembler macros, no loops, no conditionals, VERY FAST!
- See 386 source in arrayForth terminal emulator blocks 156 through 161
For More Information on Testing the MD5 Hash

- Documentation on website
  - AN001: *An Implementation of the MD5 Hash*
    (Updated version not published yet)
Choreographing a Memory
Mastering Ethernet NIC

Greg Bailey
Problem Statement

- Support Ethernet for all good reasons
  - Direct communication with rest of world
  - IP transport without writing host drivers
- Existing NICs expensive to use
  - Price of silicon and support chips (flash)
  - Cost of interface (typically a PCI bus)
- Prove practicality of high speed bit-banged communications complying with standards
<table>
<thead>
<tr>
<th>Name</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>t.cm</td>
<td>Command to TX pipeline, zero when taken.</td>
</tr>
<tr>
<td>t.pf</td>
<td>^ next descr to process, = t.rx if none</td>
</tr>
<tr>
<td>t.rx</td>
<td>^ next descr to be filled by RX, = t.ep if none</td>
</tr>
<tr>
<td>t.ep</td>
<td>^ next empty descriptor for freed buffer</td>
</tr>
<tr>
<td>t.lk</td>
<td>Latest Link Status Word</td>
</tr>
<tr>
<td>t.dp</td>
<td>32-bit count of packets dropped for no buffers</td>
</tr>
<tr>
<td>t.sk</td>
<td>^ USER AREA TO AWAKEN for TX/RX completions</td>
</tr>
<tr>
<td>t.wk</td>
<td>Value of WAKE to store into user areas</td>
</tr>
<tr>
<td>t.xa</td>
<td>20-bit TX buffer address</td>
</tr>
<tr>
<td>t.xn</td>
<td>Length of TX buffer in octets; negative to force link down; 0 when done</td>
</tr>
<tr>
<td>t.pp</td>
<td>Poll period for commands</td>
</tr>
<tr>
<td>t.tt</td>
<td>^ USER AREA TO AWAKEN for timer prodding</td>
</tr>
<tr>
<td>t.rxd</td>
<td>RX Descriptor table: +0 20-bit store address; +2 ^ Buffer Structure; +3 Unused.</td>
</tr>
</tbody>
</table>
Building TCP/IP Stack

- Port well tested ATHENA stack for polyFORTH on 32-bit machines to 16-bit environment
  - Extended memory functions optimized for simple code
  - Locating code and structures in upper memory, and buffers in extended memory, to minimize footprint on low memory
For More Information on Ethernet NIC

- Documentation on website
  - AN007: *A Bit-banged 10baset NIC*
  - DB008: *polyFORTH TCP/IP Package*
    (neither is yet published)

- Source code
  - arrayFORTH 2a blocks [720..778] Working NIC
  - polyFORTH 2a blocks [540..840] TCP/IP package
    (partially converted, working)
Thank You!

For more information, please visit
http://www.greenarraychips.com