GreenArrays™

World Leader toward Efficiency

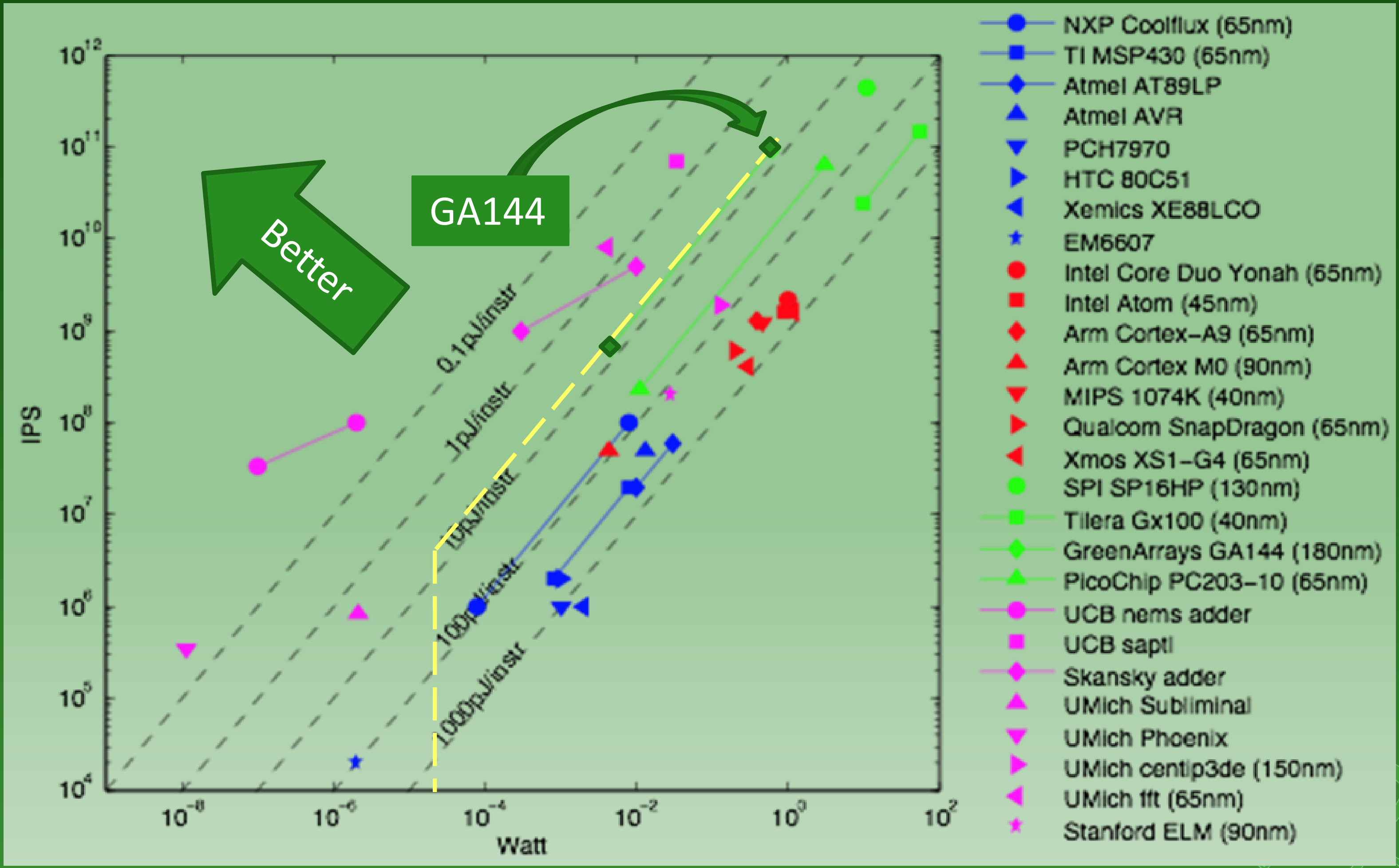
Choreographing Teams of Fast, Low-Power Computers

GreenArrays® Staff

SVFIG Forth Day

17 November 2012

The Best Known Energy Efficiency in a Commercially Available Chip.



Graph courtesy of Per Ljung, Nokia Research

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

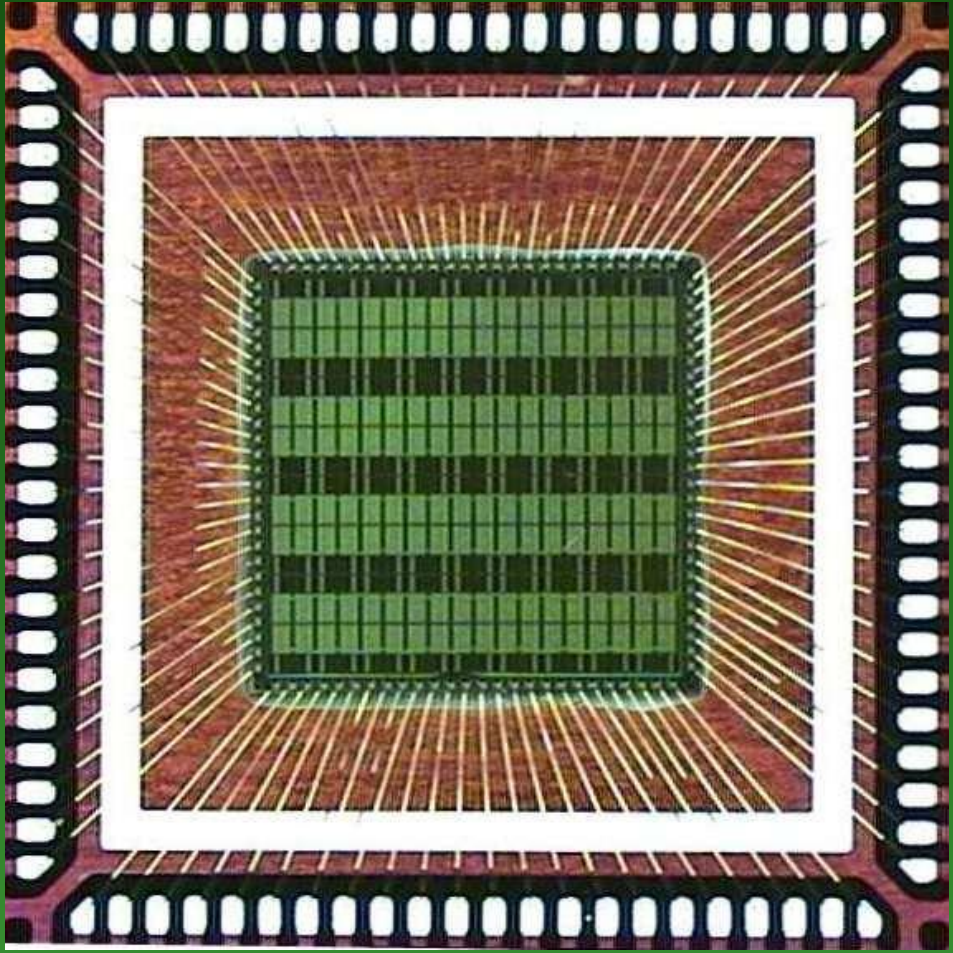
- Primary Website
  - http://www.greenarraychips.com
- arrayForth Institute
  - http://school.arrayforth.com
- Announcement Blogs
  - Business http://www.greenarraychips.com/blog1
  - Technical http://www.greenarraychips.com/blog2
- Tech Support on e-mail, Skype, Phone

GreenArrays™
World Leader toward Efficiency

F18A Architecture Review

John Rible and Greg Bailey

The Production G144A12 Chip

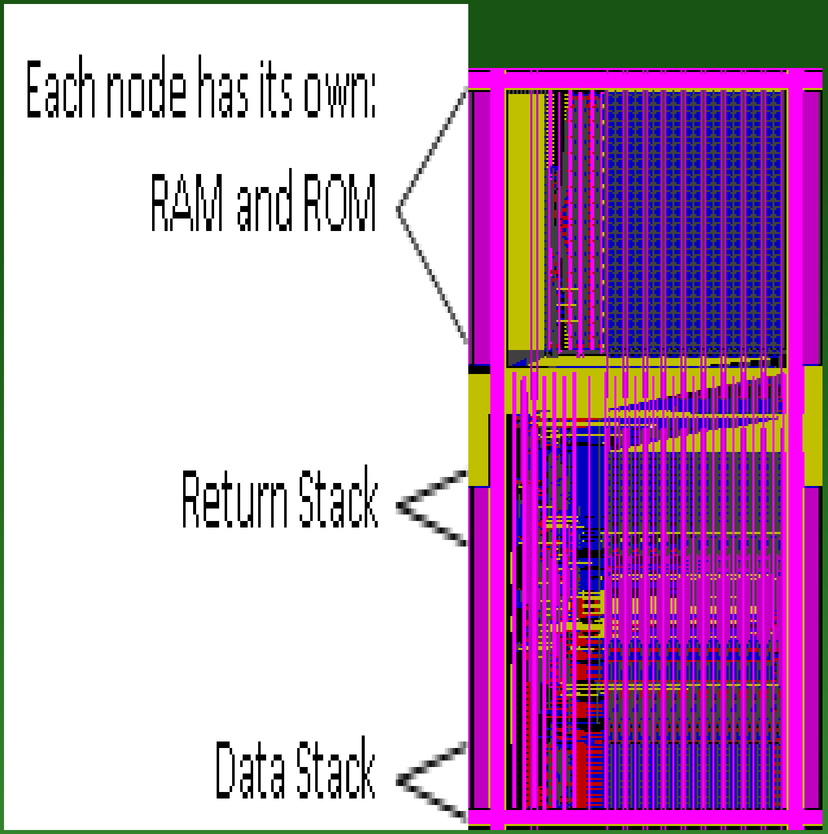


- 144 F18A computers in 8 rows, 18 columns
- Each talks to its adjacent neighbors
- 22 edge nodes have I/O pads
- Max ≈ 100 GOPS at <1 Watt
- In production. Sample kits and chips shipped for 1 year, quantity orders welcome

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 µm x 523 µm 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)

GreenArrays™
World Leader toward Efficiency

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

GreenArrays™

World Leader toward Efficiency

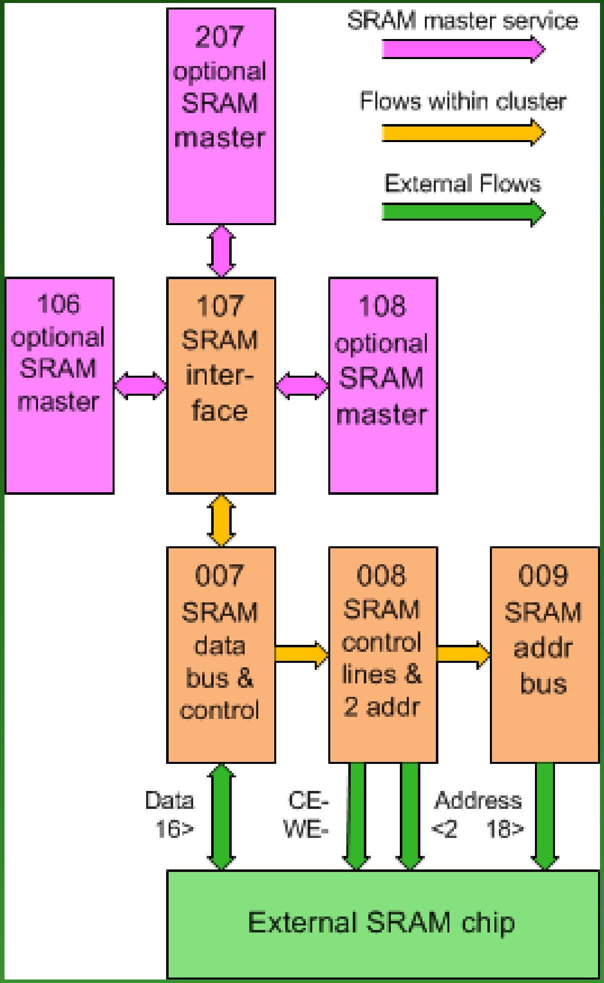
A Team to Control an SRAM

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:

Words Received				Reply Sent	Function Performed
1st	2nd	3rd	4th		
+p4	+a16	---	---	w16	**e@ Read a word from SRAM at p:a**
-p4	-a16	w16	---	---	**e! Write a word into SRAM at p:a**
-n16	+p4	a16	w16	f16	**Compare-and-exchange. Write w to SRAM iff current value = n Return x0FFFF if stored or 0 if not.**
+x	-0	m16	---	---	mk! Set master enable mask
+x	-1	m16	---	---	mk! Post stimuli for master(s)

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

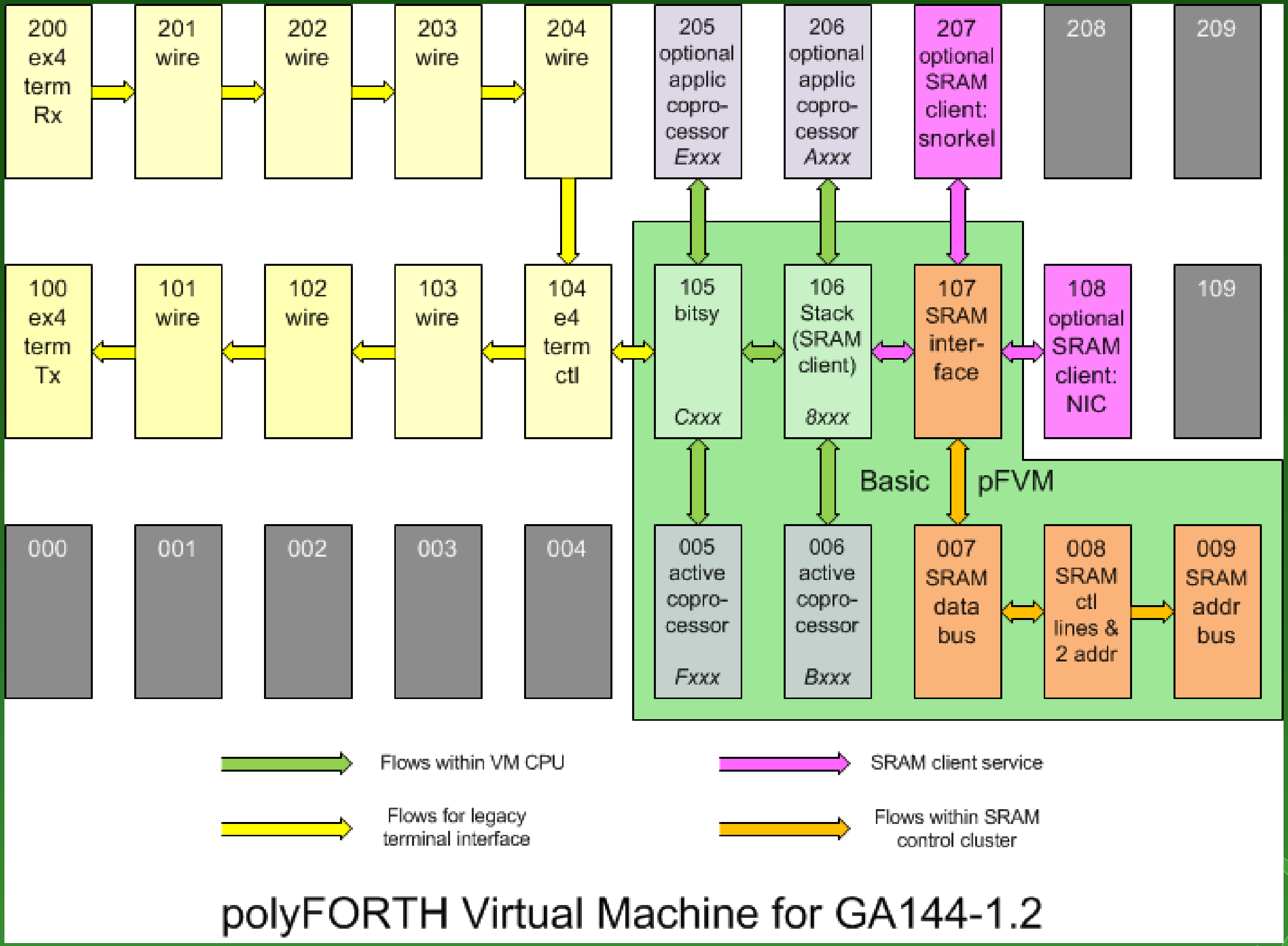
GreenArrays™

World Leader toward Efficiency

A Team to Implement a Virtual Computer

John Rible and Greg Bailey

eForth/polyFORTH VM



200 ex4 term Rx
201 wire
202 wire
203 wire
204 wire
205 optional applic copro-cessor Exxx
206 optional applic copro-cessor Axxx
207 optional SRAM client: snorkel
208
209
100 ex4 term Tx
101 wire
102 wire
103 wire
104 e4 term ctl
105 bitsy Cxxx
106 Stack (SRAM client) 8xxx
107 SRAM inter-face
108 optional SRAM client: NIC
109
Basic pFVM
000
001
002
003
004
005 active copro-cessor Fxxx
006 active copro-cessor Bxxx
007 SRAM data bus
008 SRAM ctl lines & 2 addr
009 SRAM addr bus
Flows within VM CPU
SRAM client service
Flows for legacy terminal interface
Flows within SRAM control cluster

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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Function
0	High Level Execution token (external RAM address)															Call pFVM code in low external RAM
1	1	0	0	00		ea	F18 RAM/ROM/Port address									Call F18 definition in Bitsy node 105
1	1	1	0													Call F18 definition in node 205 through **up** port of bitsy node
1	1	1	1													Call F18 defn in node 005 **down** bitsy
1	0	0	0													Call F18 definition in Stack node 106
1	0	1	0													Call F18 defn in node 206 **up** stack
1	0	1	1													Call F18 defn in node 006 **down** stack

For More Information on These Virtual Machines

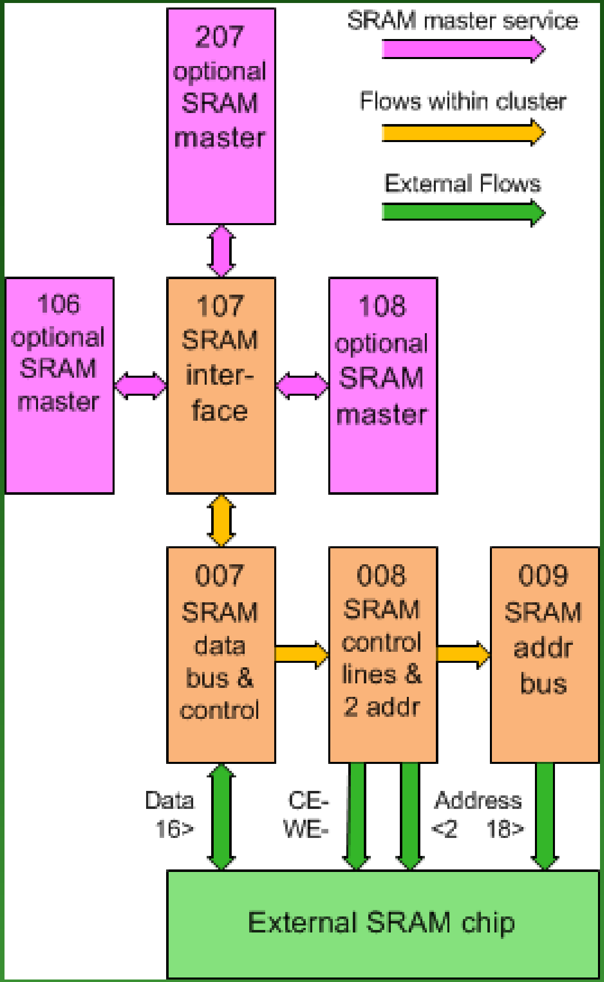
- Documentation on website
  - DB006: polyFORTH Supplement for G144A12
- Source code
  - arrayFORTH 2a blocks [360..478] for pF VM
  - arrayFORTH 2a blocks [1080..1198] for eF VM

GreenArrays™
World Leader toward Efficiency

The Snorkel: A Programmable DMA Channel

Greg Bailey

Goals for the Snorkel



- Move arbitrary 16- or 18-bit data beween 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

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) followed by one or more 5-cell instructions as follow:
low																
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		18-bit transfer size (words thru port)
low																
0												hi				20-bit SRAM address for transfer
low																

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]

GreenArrays™
World Leader toward Efficiency

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

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	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 - Header stripped when delivering payload at destination
Encoded path remaining (list of direction/distance)																		
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]

GreenArrays™
World Leader toward Efficiency

Implementing polyFORTH on a Virtual Computer

Greg Bailey

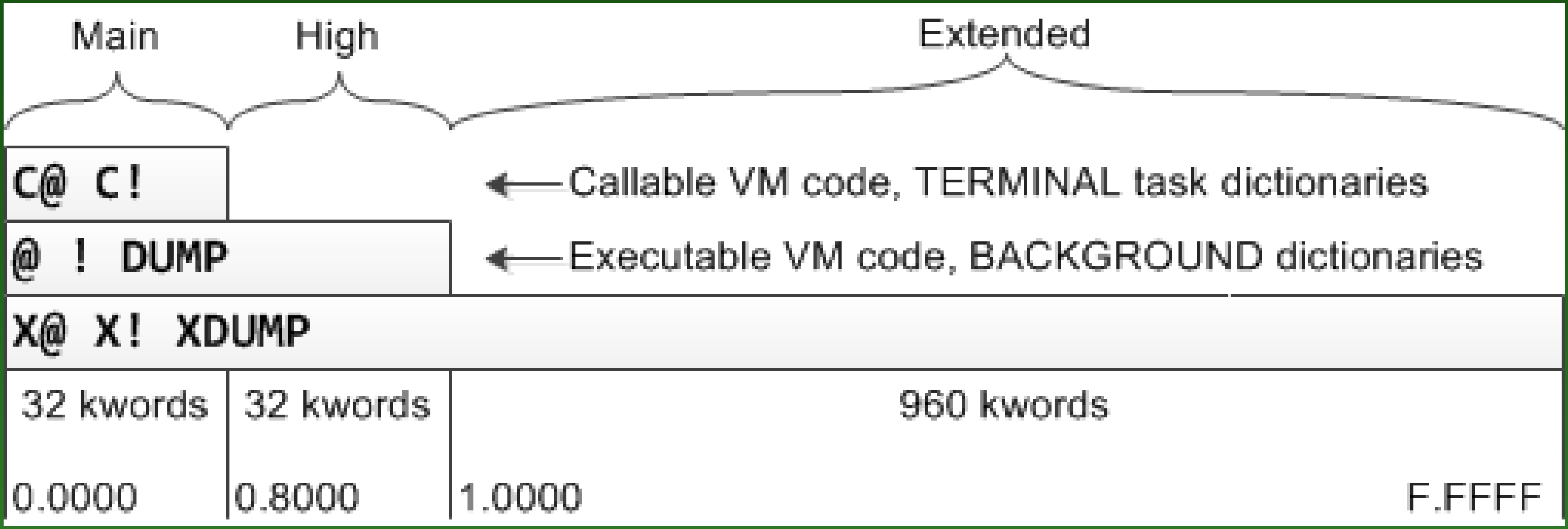
Motivation

- 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



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

GreenArrays™
World Leader toward Efficiency

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

GreenArrays™

World Leader toward Efficiency

Using polyFORTH to Explore a 3-Axis Accelerometer

Peter Milford

For More Information on This Exercise

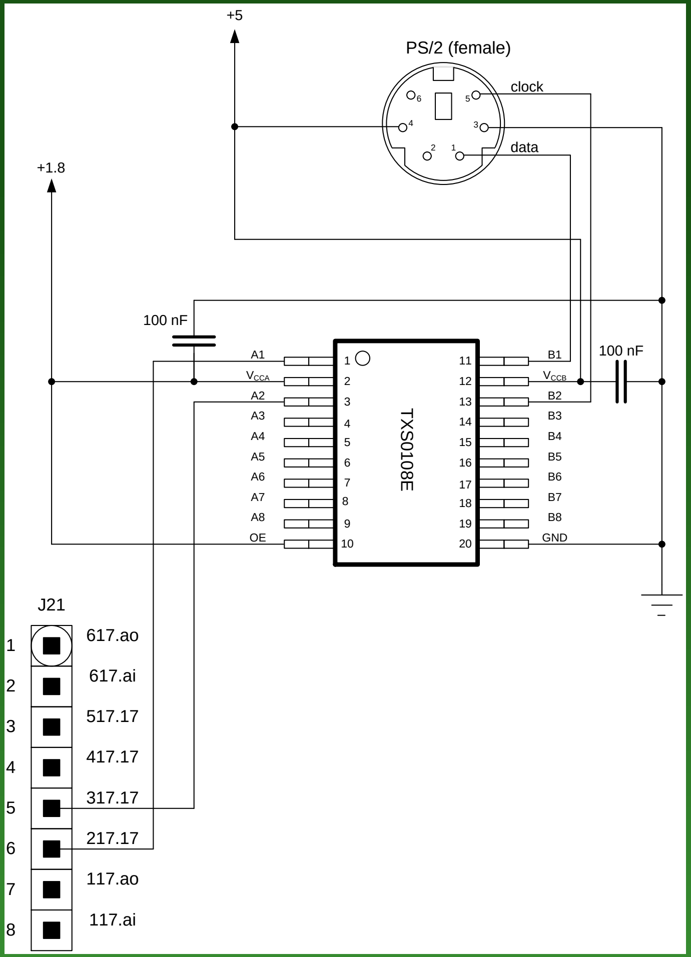
- Documentation on website
  - AN008: *Exploring a 3-Axis Accelerometer*
- arrayForth Institute
  - APP0100: *Application Notes*
- Source code
  - polyFORTH 2a block [142]

GreenArrays™
World Leader toward Efficiency

Using polyFORTH to Develop a Software-Defined Interface

Stefan Mauerhofer
presented by Greg Bailey

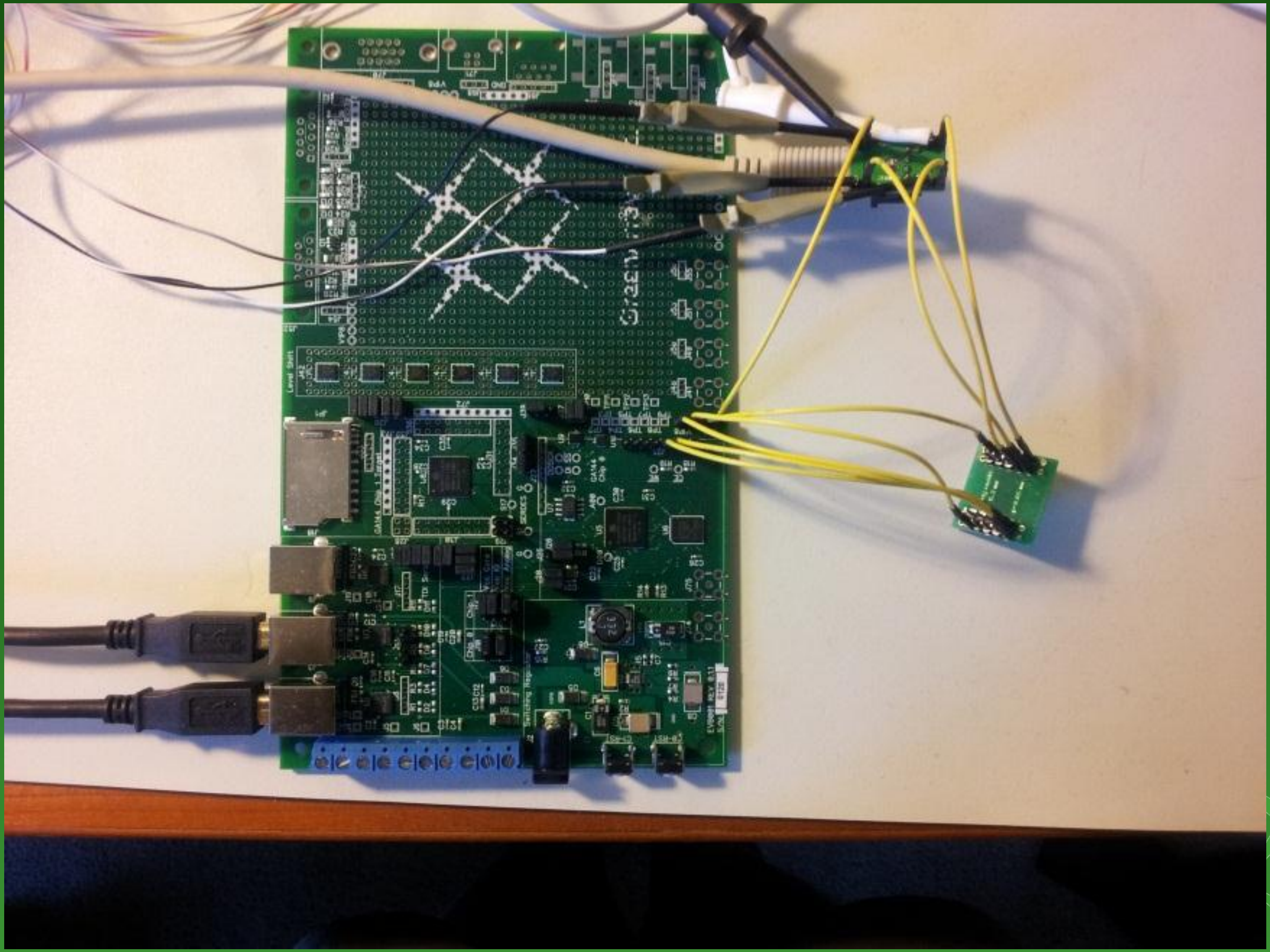
Talking with an AT Keyboard



- 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]

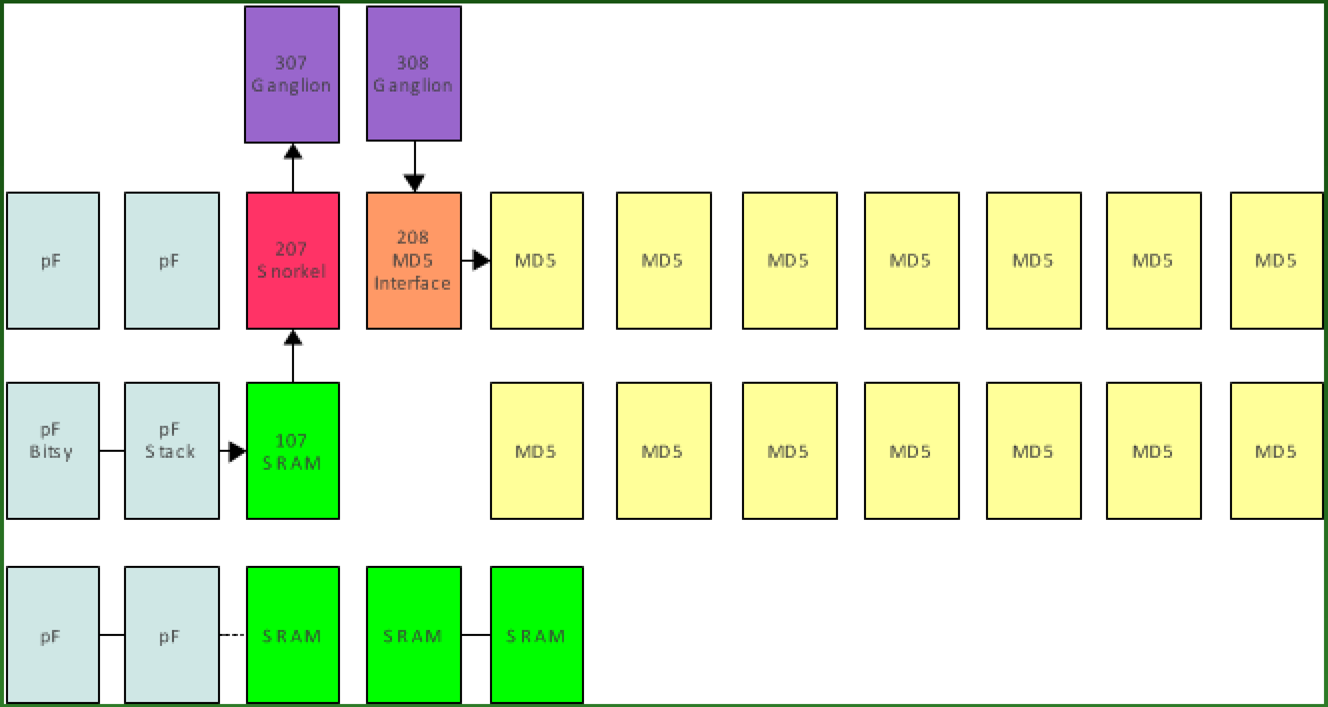
GreenArrays™

World Leader toward Efficiency

Using polyFORTH to Test and Validate an MD5 Team

Charley Shattuck

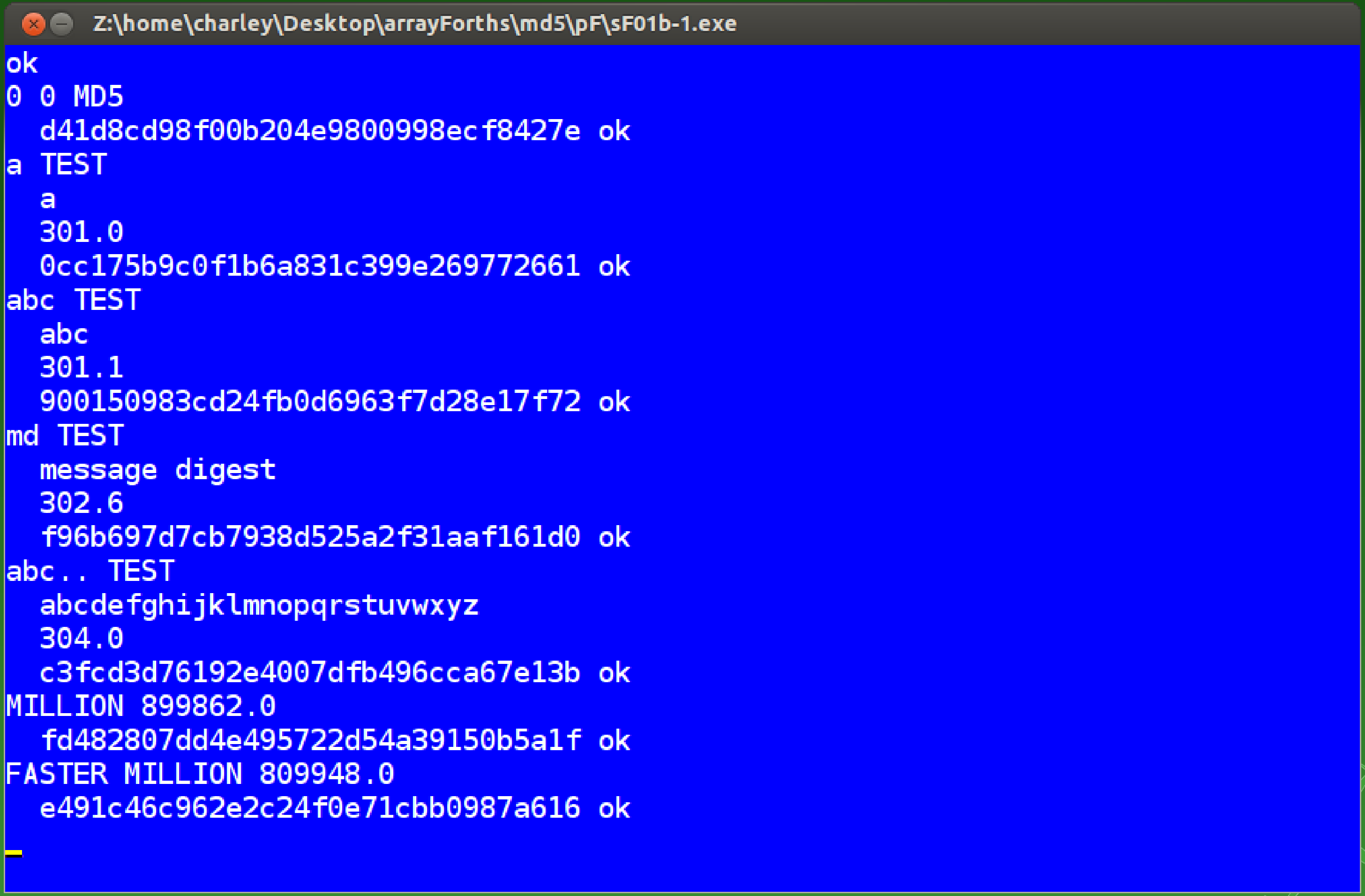
Node Diagram of MD5 Module



: MD5 (a n) <MD5 >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


Z:\home\charley\Desktop\arrayForths\md5\pF\sF01b-1.exe

```
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)

GreenArrays™

World Leader toward Efficiency

Choreographing a Memory Mastering Ethernet NIC

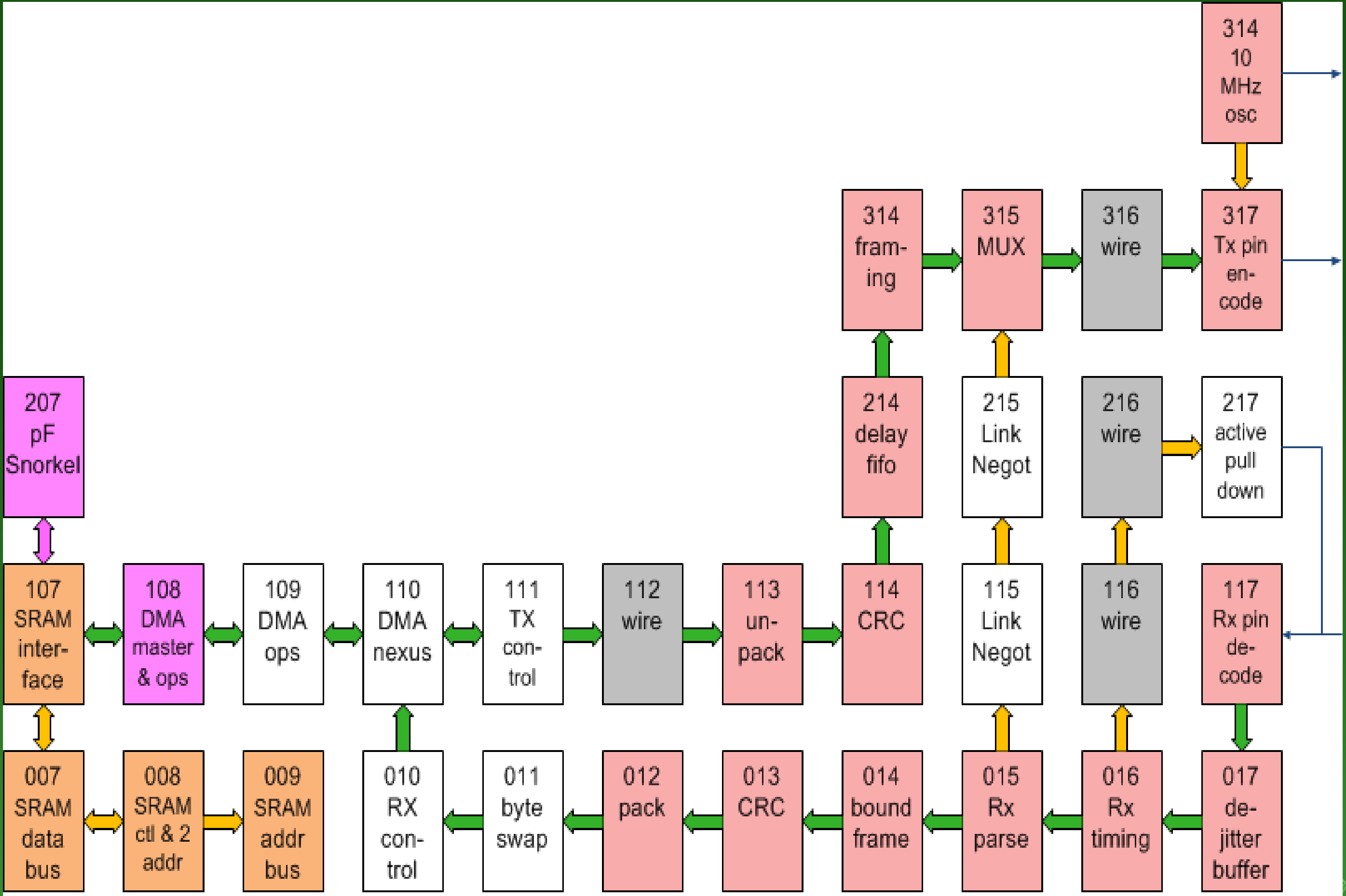
Greg Bailey

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

Shared Memory Structure

Name	Content	
t.cm	Command to TX pipeline, zero when taken.	
t.pf	^ next descr to process, = t.rx if none	RX Descriptor Pool
t.rx	^ next descr to be filled by RX, = t.ep if none	
t.ep	^ next empty descriptor for freed buffer	
t.lk	Latest Link Status Word	
t.dp	32-bit count of packets dropped for no buffers	
t.sk	^ USER AREA TO AWAKEN for TX/RX completions	
t.wk	Value of WAKE to store into user areas	
t.xa	20-bit TX buffer address	
t.xn	Length of TX buffer in octets; negative to force link down; 0 when done	
t.pp	Poll period for commands	
t.tt	^ USER AREA TO AWAKEN for timer prodding	
t.rxd	RX Descriptor table: +0 20-bit store address; +2 ^ Buffer Structure; +3 Unused.	



10 Mbit Full Duplex Ethernet NIC

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)

GreenArrays™

World Leader toward Efficiency

Thank You!

For more information, please visit

http://www.greenarraychips.com