# Beehive Overview

A "soft" multicore computer from our friends at Microsoft Research, Silicon Valley:

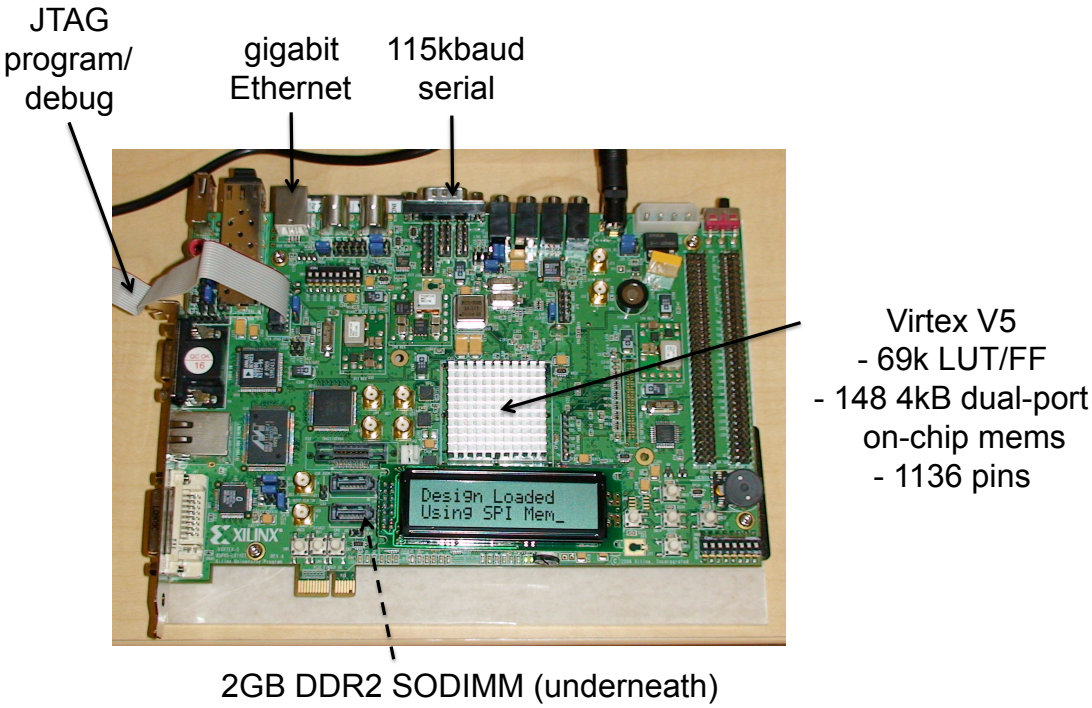
Chuck Thacker
Andrew Birrell
Tom Rodeheffer



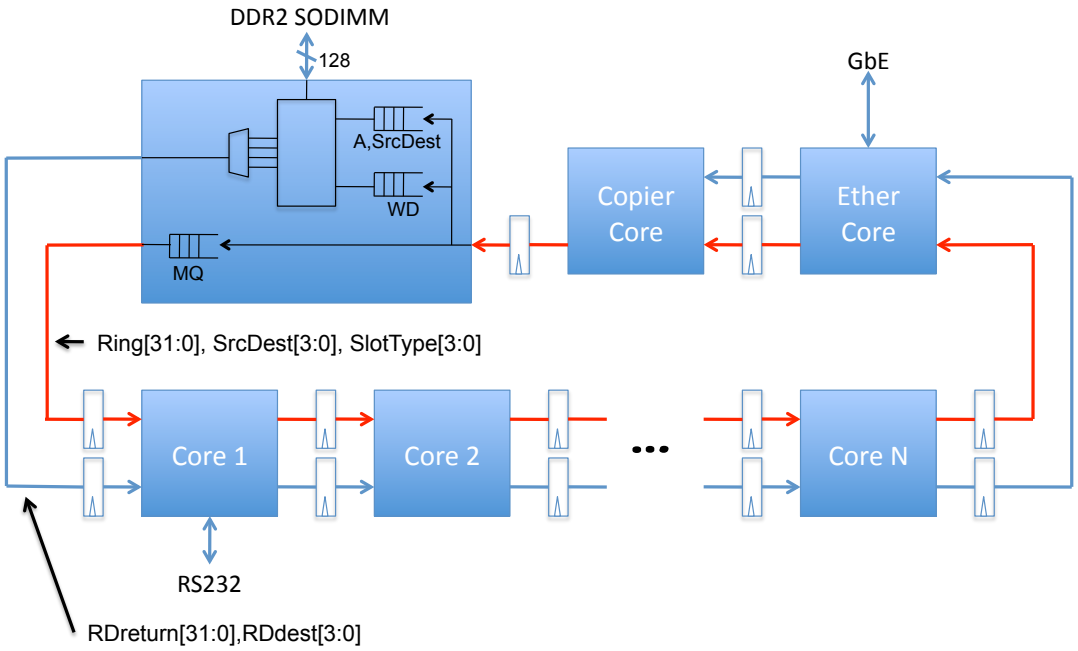
Hardware: synthesized from Verilog description, runs on Xilinx XUPV5 FPGA board:
- ring interconnect with RISC cores
- HW support for messages & locks

Software: simple multicore runtime written (mostly) in C. Tool chain includes GCC, assembler, linker, library support for HW features.

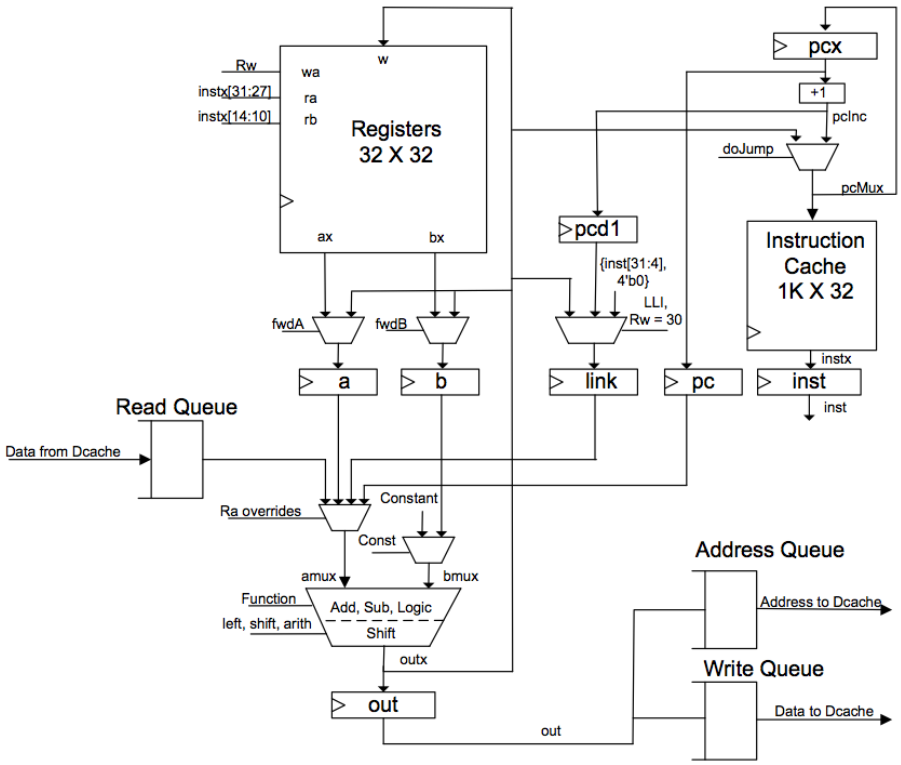
# The Hardware



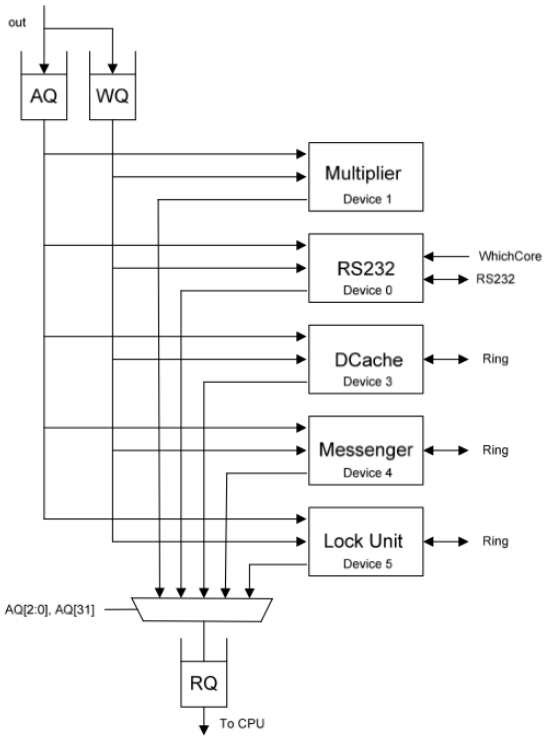
# The Beehive Architecture




# RISC CPU



# RISC "I/O" Modules




# Cache Module

- I & D caches
  - 1k word organized as 128 8-word lines
  - Direct mapped with valid bit (+ dirty bit for D cache)
  - Initial contents is boot code
    - Core 1 (master): simple TFTP client to load SODIMM
    - Cores 2-N (slaves): wait for message
- Initiate memory request on miss, eviction of dirty line
  - If write, use 8 ring slots to send 8 words of cache line
    - Ring[31:0] = data to be written
    - SlotType[3:0] = WriteData
  - Then use one ring slot to send line address to be read/written
    - Ring[31:0] = I/D, R/W, Addr[30:3]
    - SrcDest[3:0] = local core # (used when returning data)
    - SlotType[3:0] = Address
  - Dirty read miss: Read addr, 8 words of write data, Write addr
  - On reads, monitor RDdest[3:0] for our core number, capture RDreturn[31:0] as next incoming word

# Lock Module

- 64 1-bit locks
  - each core keeps a local copy of lock states
- CPU releases lock N (via write)
  - Set local_lock[N] = 0, no other action required
- CPU requests lock (via read)
  - If local_lock[N] == 1, return 2
  - Else use ring slot to send lock request around ring
    - Ring[31:0] = lock number
    - SrcDest[3:0] = local core number
    - SlotType[3:0] = Preq
  - Other cores rewrite Preq to Pfail if they hold the lock
  - If incoming slot type is Preq or Pfail with our core number
    - If Pfail, lock request failed, return 0
    - If Preq, lock request succeeds, local_lock[N] = 1, return 1
    - Rewrite slot type to Null, removing request from ring

# Lock Module (SW)

```
int icSema_tryP(int n);
  // Try to acquire lock n
  //   return 0 = fail: local lock is 0, lock held by another core
  //   return 1 = success: local lock is now 1
  //   return 2 = held: local lock was already 1
  // Assumes n is in [0..63].

static void icSema_P(int n)
  // Acquire lock n, spinning until it is acquired

void icSema_V(int n);
  // Release lock n
  // Assumes n is in [0..63].
```

# Message Module

- Can send message of 0 to 63 words to destination core
  - Use one ring slot for message header
    - Ring[31:0] = source core #, message type, length in words (N)
    - SrcDest[3:0] = destination core #
    - SlotType = Message
  - Use following N ring slots for message body
    - Ring[31:0] = next word of message
    - SrcDest[3:0] = destination core #
    - Slot Type = Message
- Can receive up to 1024 message words into a fifo
  - Monitor ring, copying Ring[31:0] to fifo when slot is a message directed to us.
  - Received Message slots are rewritten as Null.
  - 0-length messages are treated as directions for Debug Unit
  - CPU can poll to read next word from fifo (or get empty indication)

# Message Module (SW)

```
typedef unsigned int IntercoreMessage[63];

void message_send(unsigned int dest, unsigned int type,
                  IntercoreMessage *buf, unsigned int len);
// Send a message to core number "dest", using "len" words at "buf".
// Note that message lengths are measured in words, not bytes.

unsigned int message_recv(IntercoreMessage *buf);
// If there's a message available to receive, place its body in *buf,
// and return its status word.  Otherwise return 0.

unsigned int message_srce(unsigned int s);
// Given a message status word, return the source core number

unsigned int message_type(unsigned int s);
// Given a message status word, return the message "type" field

unsigned int message_len(unsigned int s);
// Given a message status word, return the message body length (in words)
```

# The Ring Interconnect

- Ring slots are seen in the same order by all cores
  - Can determine which slot was "first" or "last"
- Memory controller initiates a "train" of slots by sending a token
  - Ring[7:0] = 0 (initial count of slots following this token)
  - SlotType[3:0] = Token
  - Subsequent slots have a slot type of Null
- To use some ring slots, a core
  - Waits until Token slot arrives, adds number of slots it will use to Ring[7:0], forwards updated Token to next core
  - Forwards the original number of slots
  - Adds its slot values to the end of the train
- Memory controller
  - Consumes Token, Null, Address, WriteData slots
  - Saves other slots in fifo, add all saved slots to next departing train
  - Starts new train when current train has arrived in its entirety