PatientIO

How I *think* Kestrel-3 will use RapidIO.

Who am I?

Samuel A. Falvo II <kc5tja@arrl.net>
http://sam-falvo.github.io/kestrel

!!! DISCLAIMER !!!

I reserve the right to change anything, at any time, for any reason.

Parts

PART I

Why I Need Intelligent I/O

Ken's Challenge

Use Kestrel-3 for:

- five full business days
- at work for work.

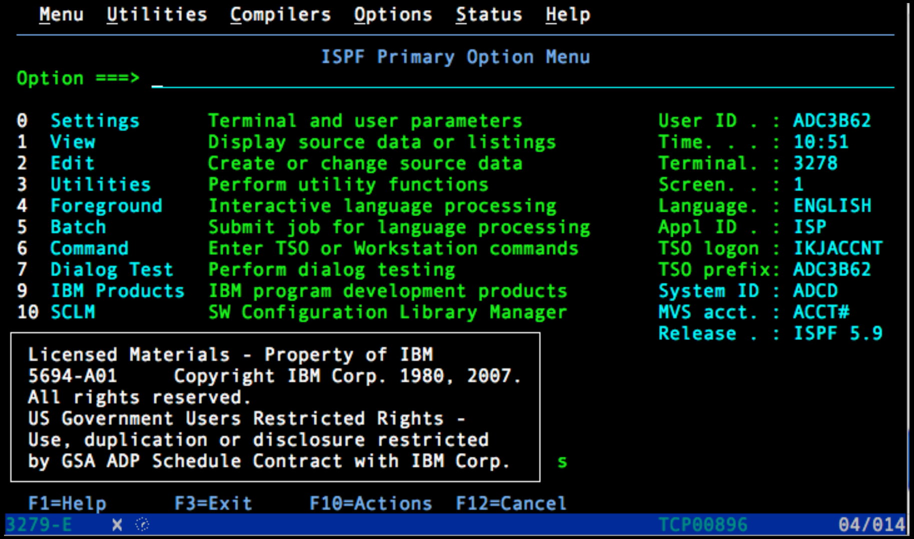
Image by Lisa Yarost, CC BY 2.0;

primary source: http://tinyurl.com/pe7meyu

My Plan

Use the Kestrel-3:

- as a VNC terminal
- to a cloud-hosted server.



```
 Menu  Utilities  Compilers  Options  Status  Help
 ------------------------------------------------------------------------------
                          ISPF Primary Option Menu
Option ===> _

0  Settings      Terminal and user parameters            User ID . : ADC3B62
1  View          Display source data or listings         Time. . . : 10:51
2  Edit          Create or change source data            Terminal. : 3278
3  Utilities     Perform utility functions               Screen. . : 1
4  Foreground    Interactive language processing         Language. : ENGLISH
5  Batch         Submit job for language processing      Appl ID . : ISP
6  Command       Enter TSO or Workstation commands       TSO logon : IKJACCNT
7  Dialog Test   Perform dialog testing                  TSO prefix: ADC3B62
9  IBM Products  IBM program development products        System ID : ADCD
10 SCLM          SW Configuration Library Manager        MVS acct. : ACCT#
                                                         Release . : ISPF 5.9
 Licensed Materials - Property of IBM
 5694-A01     Copyright IBM Corp. 1980, 2007.
 All rights reserved.
 US Government Users Restricted Rights -
 Use, duplication or disclosure restricted
 by GSA ADP Schedule Contract with IBM Corp.  s

 F1=Help      F3=Exit      F10=Actions  F12=Cancel
3279-E    X                                         TCP00896       04/014
```

Peripheral Output Requirements

(Highest bandwidth)

Video.

SD Card.

Ethernet Adapter.

(Lowest bandwidth)

Peripheral Input Requirements

(Highest bandwidth)

Ethernet Adapter.

SD Card.

Keyboard and Mouse.

(Lowest bandwidth)

Limitations

16-bit path to RAM.

70ns access time (~14MHz).

28MB/s peak.

12.5MHz clock, 25MB/s expected.

CPU @ 12.5MHz yields 6.25 MIPS peak.

Slow, but still 2x faster than 12.5MHz MC68000. :)

Bandwidth Budget: Video Refresh

640*480 px/f = 307,200 px/f

 * 8 b/px = 2,457,600 b/f

 * 60 f/s = 147,456,000 b/s

 / 16 b/t = 9,216,000 t/s

 * 2 B/t = 18,432,000 B/s (18.5 MB/s)

Bandwidth Budget: Network

Video leaves only 6.5MB/s memory bandwidth left over for network I/O and CPU combined.

That’s 52Mbps for fastest possible network transfer.

Bandwidth Budget: CPU

CPU wants to fetch 32-bit instructions.

Since it won’t have a cache, it *really* wants 25MBps to RAM.

Bandwidth Budget: Results

CPU performance will vary between 0.0 MIPS and 6.25 MIPS, depending on network traffic and video retrace activity.

Expected average: 1.6 MIPS.

(Slightly slower than 8MHz MC68000.)

Software Expectations, redux.

1.6 MIPS is:

- *plenty* fast enough for keyboard and mouse event handling.
- *barely* fast enough for framebuffer updates. I expect sub-Atari 520ST performance: usable, but irksome.

Software Expectations, redux.

Bit-banging SD and network I/O, therefore, is *clearly* out of the question.

I need I/O that can work on its own, without CPU intervention, as much as possible.

PART II

Channel I/O

Channel I/O

Channels are programmable DMA engines. They execute *their own instruction set*.

First appearing in the IBM 709, they *really* came into their own with System/360, and has remained a fixture to this very day.

Channel I/O

Master/slave.

Periodically poll to see if device need service.

With clever programming, polling can be hidden from CPU via an endless channel program.

Channel I/O

Channel programs are particularly ideal for taking care of medium- to high-level protocol functionality.

READ, WRITE, (addresses units)
SENSE, CONTROL, (addresses controllers)
TRANSFER-IN-CHANNEL.

Channel I/O

Channel I/O for SPI needs more instructions than IBM S/360 to account for lower level protocols.

RECV, SEND, EXCH, WAIT, DROP, SBYTE, SSEL, RATE, etc.

SD Card Read Protocol

Send command block (6 bytes).

Wait for R1 response byte.

Wait for Data Token byte.

Read in 512 data bytes.

Read (and perhaps ignore) 2-byte CRC.

Channel Program: SD Card Read

```
SEND  commandBlock, 6
WAIT  $FF, $FF, $FE, $00
READ  $FF, bufferAddr, 512
DROP! $FF, 2
```

SD Card Write Protocol

Send command block (6 bytes).

Wait for R1 response byte.

Send Data Token byte.

Send 512 data bytes.

Send 2-byte CRC.

Wait for R1 response byte.

Wait for card busy status to clear.

Channel Program: SD Card Write

```
SEND    commandBlock, 6
WAIT    $FF, $FF, $FF, $00
SBYTE   $FE
SEND    bufferAddress, 512
SBYTE   crc1
SBYTE   crc2
WAIT!   $FF, $FF, $FF, $00
```

Channel I/O

Because channels run independently of CPU, with buffers as a first-class data type, data exchange can reach up to 50Mbps peak.

A channel program can interrupt the CPU to let it know it has finished.

Channel I/O

I’m evaluating an S16X4-derived MISC core for use as the channel processor.

```
# ## ## ##        commandBlock (64-bits)
# SEND NOP        commandLength (16-bits)
# # WAIT          sendByte andMask||xorMask
...
```

Channel I/O

I’m evaluating an S16X4-derived MISC core for use as the channel processor.

It’s Turing-complete.

It’s a proven architecture (CPU for Kestrel-2!).

PART III

PatientIO

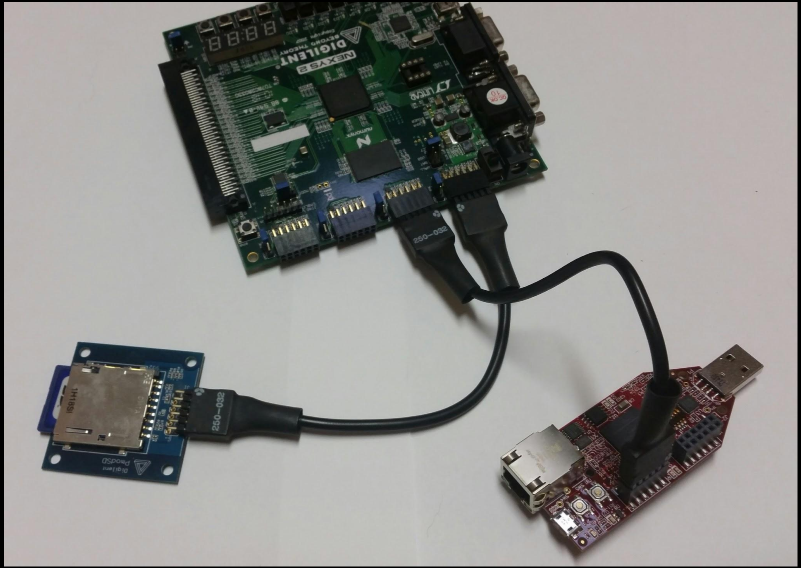
Why RapidIO

Channel I/O is *great* for master/slave type devices, like SD and MMC memory cards, ADCs and DACs, SPI flash memories, etc.

Digilent Nexys2 only has 4 PMod ports!

AVNet board only has 2 PMod ports!

Why RapidIO



Kestrel-3 motherboard

Ethernet
(PatientIO)

SD card slot
(Channel I/O)

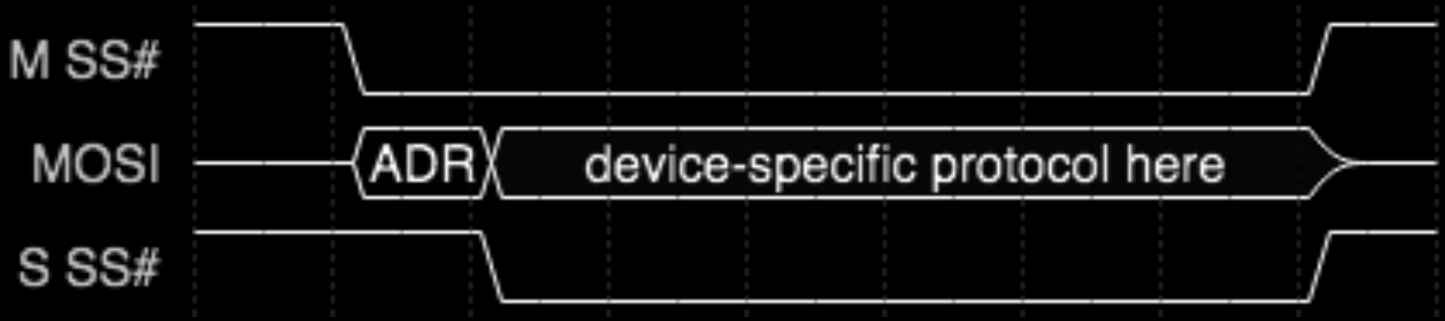
Why RapidIO

To support two or more I/O devices on a single channel, IBM opted to complect their channel I/O concept with “block multiplexors” and “byte multiplexors”.

Why RapidIO

SPI link expanders globally affects protocol all devices talk on that link.

Drivers must be aware this is happening!



Why RapidIO

Channel I/O still not a good fit for autonomous peripherals.

E.g., no way to tell device A to deposit received traffic *directly* into device B’s channel without going through the host’s channel hardware first.

Why RapidIO

Packet-switched I/O fabric.

Proven protocol I don’t have to invent myself.

Typically spec’d for links 1 Gbps or faster.

Proper superset of channel I/O.

Why RapidIO

RapidIO is a set of:

- open,
- easily available,
- *easily understood*

standards.

http://www.rapidio.org/rapidio-specifications/

RapidIO Summary

Supports bus semantics as simple as the 6502.

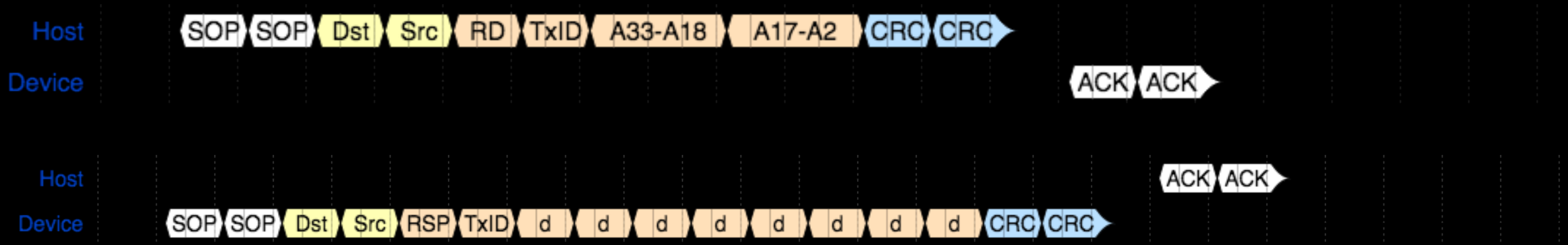
Supports NUMA and ccNUMA architectures.

Supports general message passing.

First implemented to replace PowerPC’s FSB.

RapidIO Summary

Typical overhead: 24 bytes including ACKs.



RapidIO Summary

Peer-to-peer, **not** master/slave.

34-/50-/66-bit address space per device.

256, 65536, or 4.2B devices per fabric.

Up to 256 byte payloads in powers of 2.

Intended for hardware implementation.

PatientIO

RapidIO physical layer bindings for SPI

- 50Kbps to 100Mbps, x1 SDR.
- 6.4Gbps+ via 100MHz, bonded, x4 DDR.

PatientIO

Pin many RapidIO variables.

- 256 devices per fabric.
- 34-bit address space per device.
- All devices have vendor ID $FFFF.
- UUID-based service discovery.
- SPI-specific CARs and CSRs

PatientIO

"Vendor ID" needed to make peripherals.

$9500/year to RapidIO TA.

WAY too expensive for the homebrewer!

PatientIO

Inefficient device driver pairing.

(model ID * vendor ID) -> driver.

(Just look at the horror that is Linux PCI support.)

PatientIO

PatientIO uses COM-inspired query-interface approach to service discovery.

No vendor ID necessary.

(Though, not mutually exclusive.)

PatientIO

```
extern UUID PIOIID_STD_UART;
...
if(queryInterface(deviceID, &PIOIID_STD_UART, &ptr)) {
   writeString(deviceID, ptr, “+++”);
   sleep(SECONDS(2));
   writeString(deviceID, ptr, “ATH\r\n”);
} else {
   printf(“Device %d doesn’t offer a UART.”, deviceID);
}
```

PatientIO

```
CREATE PIOIID_STD_UART
  $1111111122222222 , $3333333344444444 ,
: err ( devID -- )
  ." Device " . ." doesn't offer a UART." CR ;
: slam ( devID ptr -- )
   2>R S" +++" 2R@ writeStr 2000 MS
   S\" ATH\r\n" 2R> writeStr ;
: hangup ( devID -- )
  DUP PIOIID_UART query IF slam ELSE err THEN ;
```

PatientIO

If a device reports it supports PIOIID_x, then:

- It implements *at least* the required registers specified by “x”.
- It implements *at least* the required control bits as specified by “x”.
- Registers *must be* placed relative to each other as “x” specifies.

PatientIO

If a device reports it supports PIOIID_x, then:

- It implements *at least* the required interrupts that “x” specifies.
- Supports *at least* the required mailboxes and messaging protocols that “x” specifies.
- Behavioral semantics *must* be *fully* backward compatible with “x”.

PART IV

Keyboard and Mouse

Keyboard and Mouse

Today, Keyboard Interface Adapter

- Handles PS/2 communications for one port.
- Input only. No output capability exists.
- Memory-mapped.

Keyboard and Mouse

KIA characteristics include:

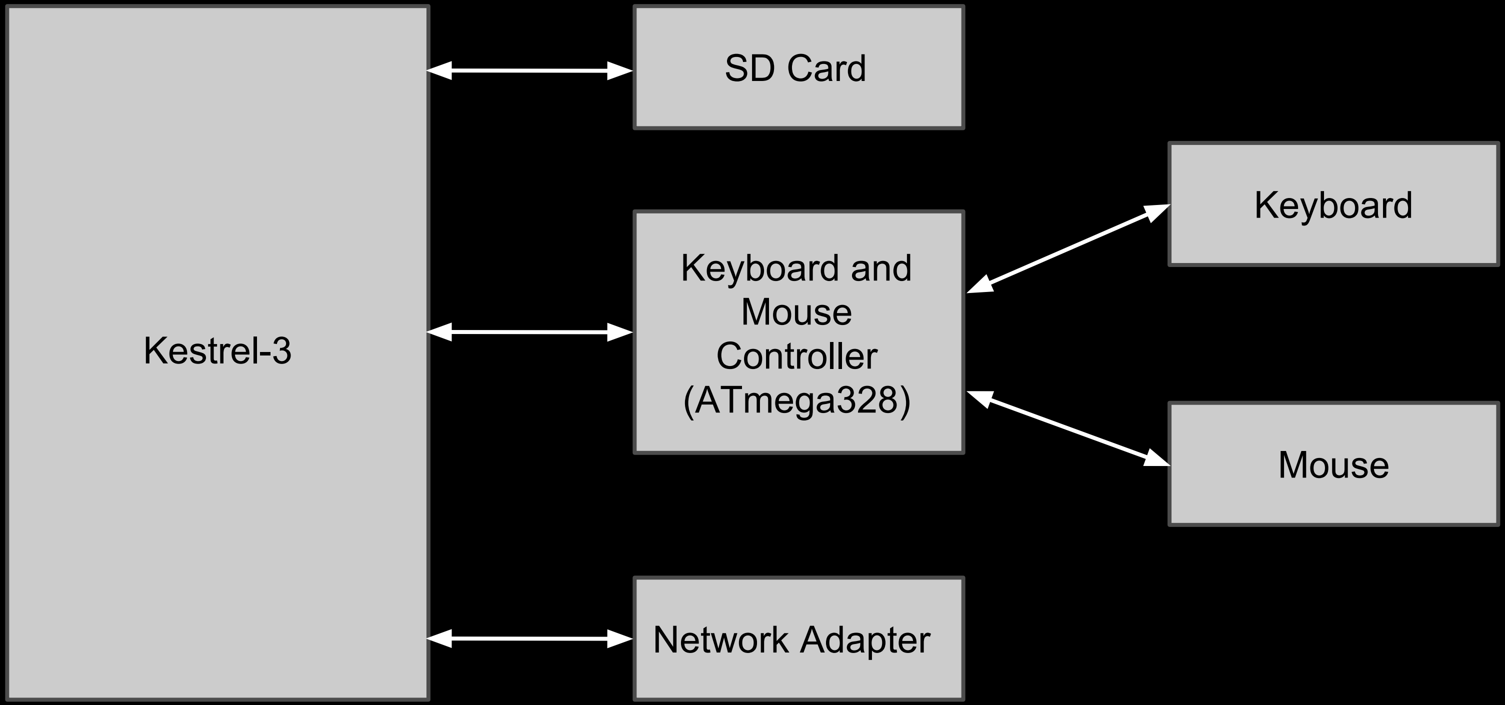
- 16-byte queue, but no interrupts.
- CPU must poll periodically or risk data loss.
- First introduced in the Kestrel-2.

Keyboard and Mouse

Digilent Nexys2 FPGA board only has one PS/2 port.

I must use one Pmod port for mouse anyway, I might as well use it for keyboard too.

Keyboard and Mouse



Keyboard and Mouse

Two PS/2 ports (one keyboard, one mouse)

Event notification and LED control through RapidIO message passing.

Keyboard and Mouse

Isolate system firmware from keyboard hardware through generic scancodes.

Between generic scancodes and message passing semantics, keyboard driver software should work with any kind of keyboard.

Keyboard and Mouse

Anticipated keyboard message (8 bytes)

K1	K2	K3	K4	K5	K6	K7	K8

Keyboard and Mouse

Anticipated mouse message (8 bytes)

Buttons	00	00	00	dX (high)	dX (low)	dY (high)	dY (low)

PART V

Network Adapter

Network Adapter

Couples to 100Mbps Ethernet.

Line rate limited to 50Mbps due to RAM bandwidth.

Built on AVnet FPGA board.

Network Adapter

To send: Nexys2 board assembles entire frames to transmit in AVnet’s RAM first, then tells network controller to send it.

To Recv: AVnet sends doorbell to Nexys2, which then reads only the frames it wants from AVnet RAM.

Network Adapter

Use NREAD, NWRITE, SWRITE packets to access network adapter’s memory.

Use MESSAGE packets to tell network adapter what to do with them.

PART VI

Native PatientIO Bridge

PatientIO Bridge

Everything discussed to this point builds on top of SPI Channel I/O.

SPI is slow enough for this to work.

What about the future though?

PatientIO Bridge

With a 4-wide, x4 DDR SPI channel running at 100MHz, setting up the channel program and kicking it off takes more time than sending a packet.

PatientIO Bridge

Remember: RapidIO intended to replace PowerPC front-side bus.

Entire protocol meant to be codified in hardware. What I’ve done with PatientIO is a hack!

PatientIO Bridge

Bridge acts as a surrogate peripheral in local address space.

Translates local, parallel bus transactions into serializable RapidIO packets.

PatientIO Bridge

Base Address Registers (BARs) define the start and size of a RapidIO region in local address space.

They also tell the bridge which device the I/O window corresponds to.

PatientIO Bridge

A bridge also serves as a local memory bus master for incoming traffic, too.

Translates in-bound transactions into local memory accesses. No need to interrupt CPU to handle remote DMA requests.

PatientIO Bridge

Significant hardware investment (on par with a CPU cache controller).

However, it’d eliminate the majority of PatientIO driver software in firmware. Only fabric auto-config and device registry would remain.

Summary

Channel I/O is foundation for 1st-gen PatientIO.

Consistent device model, driver architecture.

Scales from kilobits to gigabits per second.

Switches offer zero-overhead port expansion.

Easy software implementation for µCs.

2nd-gen hardware bridges for rubber-burning performance.

Summary

Channel I/O to drive SD card.

Keyboard and mouse rely only on messaging.

Network adapter relies on both messaging and RDMA.

Two device driver stacks: SD/MMC and PatientIO.

NIC, keyboard, and mouse drivers just thin veneers.

Questions?

EXTRA SLIDES

Software Expectations

Expected CPU workloads:

- Bit-block transfers from VNC.
- Packet and event routing.
- Keyboard and mouse updates.
- Block storage I/O.

Channel I/O (Digression)

I have to admit, I’m jealous of IBM.

IBM’s channel I/O is so elegant because they control channel hardware, instruction set, *and* controller hardware, so it all plays together nicely. I don’t have that luxury. :(

PatientIO

Interface-based approach does not require an expensive, central authority to maintain a vendor ID registry.

You can make a peripheral today without my knowledge or consent.