Selected Lessons Learned Building the KCP53000

Samuel A. Falvo II <kc5tja@arrl.net>

I do not have a name! I have a number!

I am a free core!

Use numbers, not names.

- KCP53000 went through several names during its development.
  - Original name long forgotten.
  - Polaris (trademarked by AMD for a family of GPUs)
  - Now, KCP53000.
- We want what Intel, AMD, et. al. doesn't want.
- Be proactive in avoiding legal issues.

Avoid Premature Pipelining.

Get it working first!

Avoid pipelining at first.

Build slow, sequenced, working implementation first.

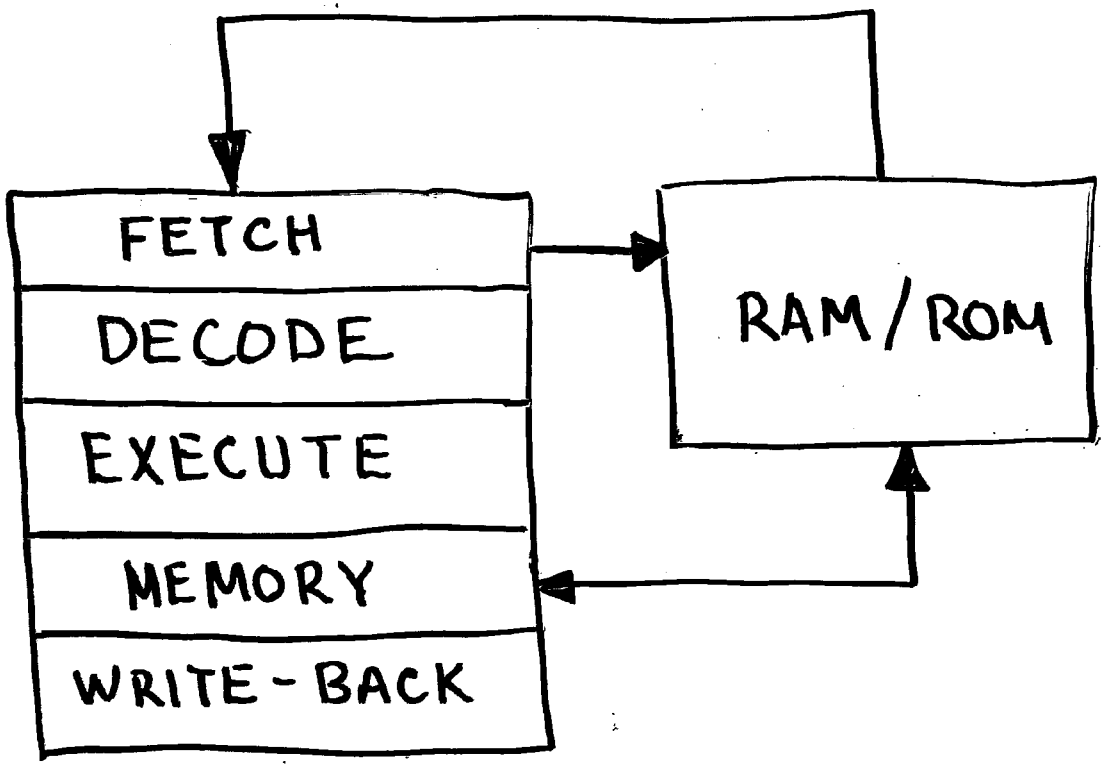
- First four attempts at building pipelined CPU failed.
- Hockey-stick complexity; can't keep it all in my head at once.
- Premature evil is the root of all optimization.

Avoid pipelining at first.

- Fifth attempt finally successful.
  - Two stage pipeline.
  - PLA-based sequencer and decode logic, just like the 6502.
  - Instruction fetch and execute *currently* take turns.
  - Easy to alter PLA to allow overlapping fetch and execution later.

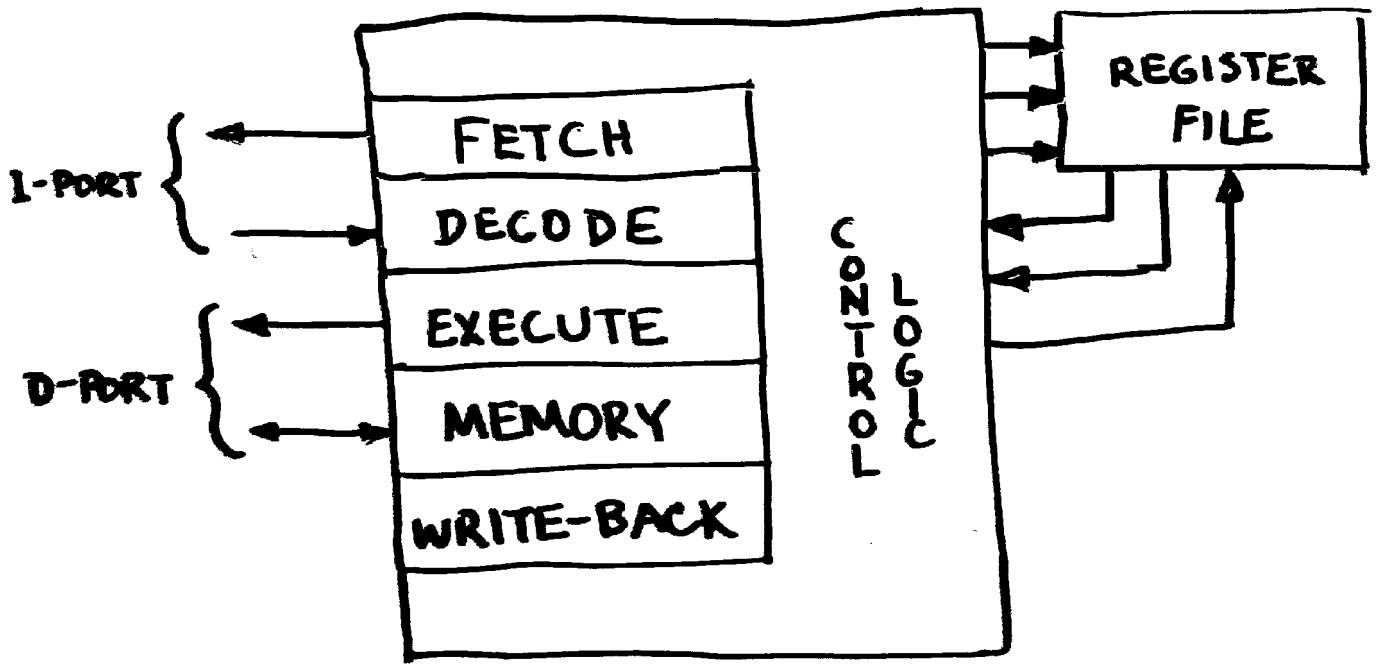
Pipeline design documents are misleading.

- They're not *wrong*; but they're not *right* either.
- Only data-flow shown; control-plane obscured.



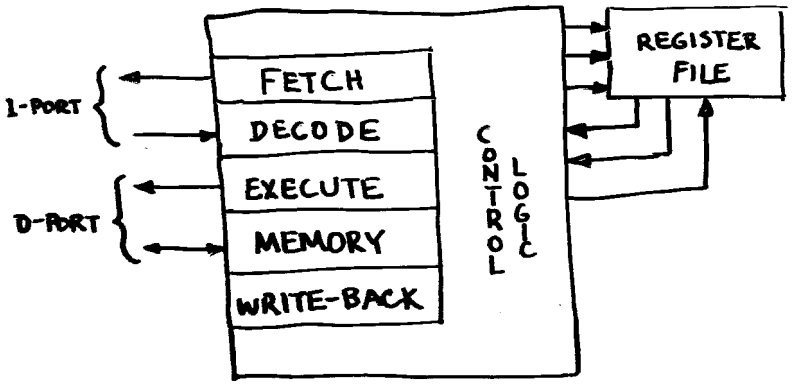
Pipeline design documents are misleading.

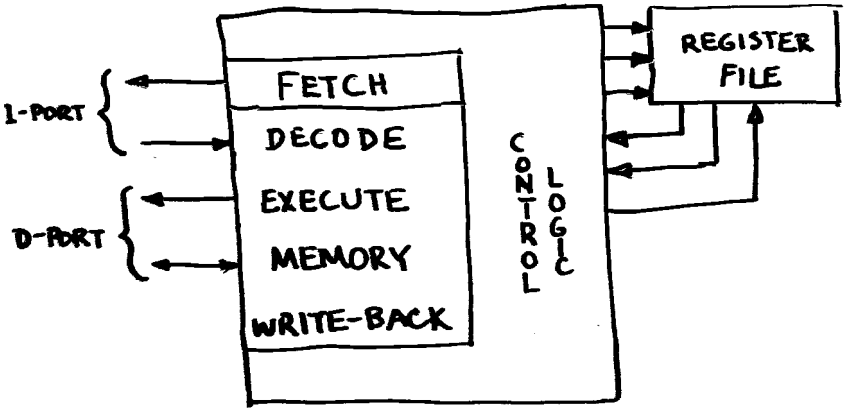
- Pipeline stages are *really* independent functioning units, under centralized control.



Pipeline design documents are misleading.

- Separation of concerns makes refactoring easier later on.
- Compare ideal vs. current KCP53000 pipeline.





Pipeline design documents are misleading.

Opportunities for Optimization

Optimization	Minimum	Maximum
Overlap instruction fetch and execution	60%	100%
Perform both source register fetches at once	25%	33%
Overlap instruction fetch and register write-back	30%	50%

These compound; given same 25MHz clock, design that once ran 6 MIPS now can run 16 MIPS to 24 MIPS.

Count Furcula!

64 bits! Ah … ah … ah!

Furcula Bus

- Fork of Wishbone B3 to better meet CPU's needs.
- Justified data transfers; **no byte lanes.**
- Full address bus exposed; no low address erasure.
- SIZ_O[1:0] exposed to indicate transfer size.
- SIGNED_O exposed to indicate sign-extension.
- Otherwise, identical to Wishbone B3.

In retrospect, exposing SIGNED_O was a bit of a mistake, but thankfully not a terribly big one.

Wishbone vs. Furcula Comparison

Wishbone byte-lanes

	Byte 0	Byte 1	Byte 2	Byte 3	Hword 0	Hword 1	Word
DAT_IO[7:0]	X				X		X
DAT_IO[15:8]		X			X		X
DAT_IO[23:16]			X			X	X
DAT_IO[31:24]				X		X	X

Wishbone vs. Furcula Comparison

Furcula Justification

	Byte 0	Byte 1	Byte 2	Byte 3	Hword 0	Hword 1	Word
DAT_IO[7:0]	X	X	X	X	X	X	X
DAT_IO[15:8]					X	X	X
DAT_IO[23:16]							X
DAT_IO[31:24]							X

Why Furcula?

- Allows direct CPU-to-external-circuitry connection.
- Intel CPUs pre-decode size and lower address bits to break data into individual byte lanes.
- Buses out-live CPUs.
- Supporting legacy buses requires resynthesizing information from byte lane enables.
- Thus, bus bridge circuitry must "undo" what CPU-internal circuitry already did.

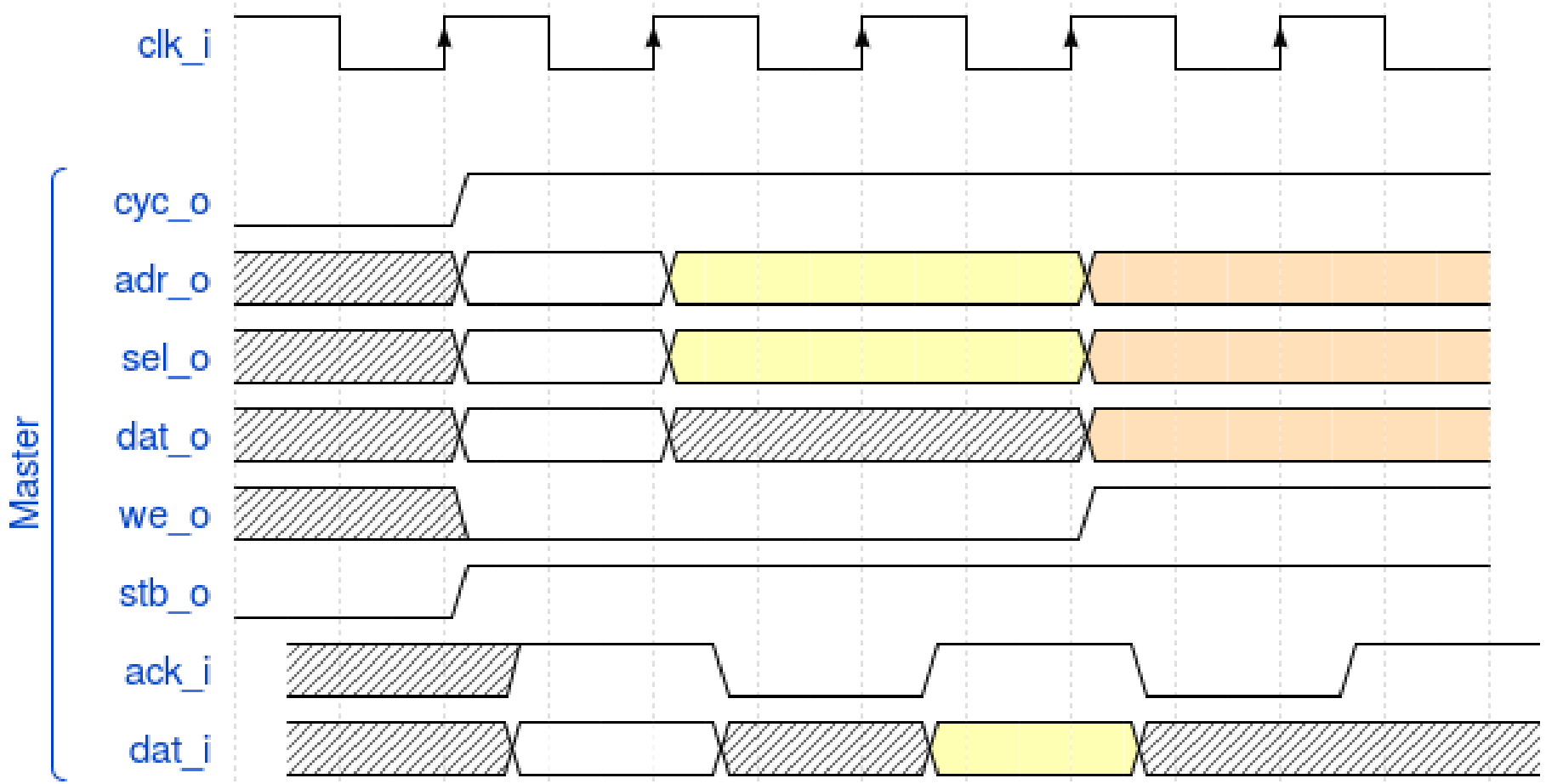
Why Furcula?

- Therefore, just let external circuitry handle alignment and sign-extension exactly *once*.
- Trivial to bridge to Wishbone B3.
  - Just needs some data bus multiplexors to route data to/from byte lanes.
  - Handful of gates to decode byte lane select signals.
  - Sign-extension logic.

Shoulda, Woulda, Coulda.

I should have chosen
Wishbone B4 over Wishbone B3.

Wishbone B3



Wishbone B3

- Closer in scope to PCI than it is to AXI4.
- ACK_I **must:**
  - assert in same cycle to achieve single-cycle performance. (<= 35ns for 25MHz)
  - negate after transaction completes.

Wishbone B3

- To get 100% efficiency on bus, you **must**:
  - Use fully asynchronous slave logic that responds in **35ns or less** (assuming 25MHz bus).
  - Ensure at least one address bit changes between cycles, so slave can detect end of one cycle and start of another.
  - Alternatively, explicitly declare adjacent transactions to the same address undefined, and *assume* software does the right thing.

Wishbone B3

Why It Fails for Kestrel-3:

- Bus bridges and adapters add propegation delays to address decoding logic.
- Brings max. clock speed from around 100MHz to 29MHz on Xilinx parts. Ouch!
- Demonstrably incompatible with PSRAM chip timings on Nexys-2 board.

Kestrel-3 takes around ***33ns*** from address valid to ACK_I. (Remember, 35ns was our limit!)

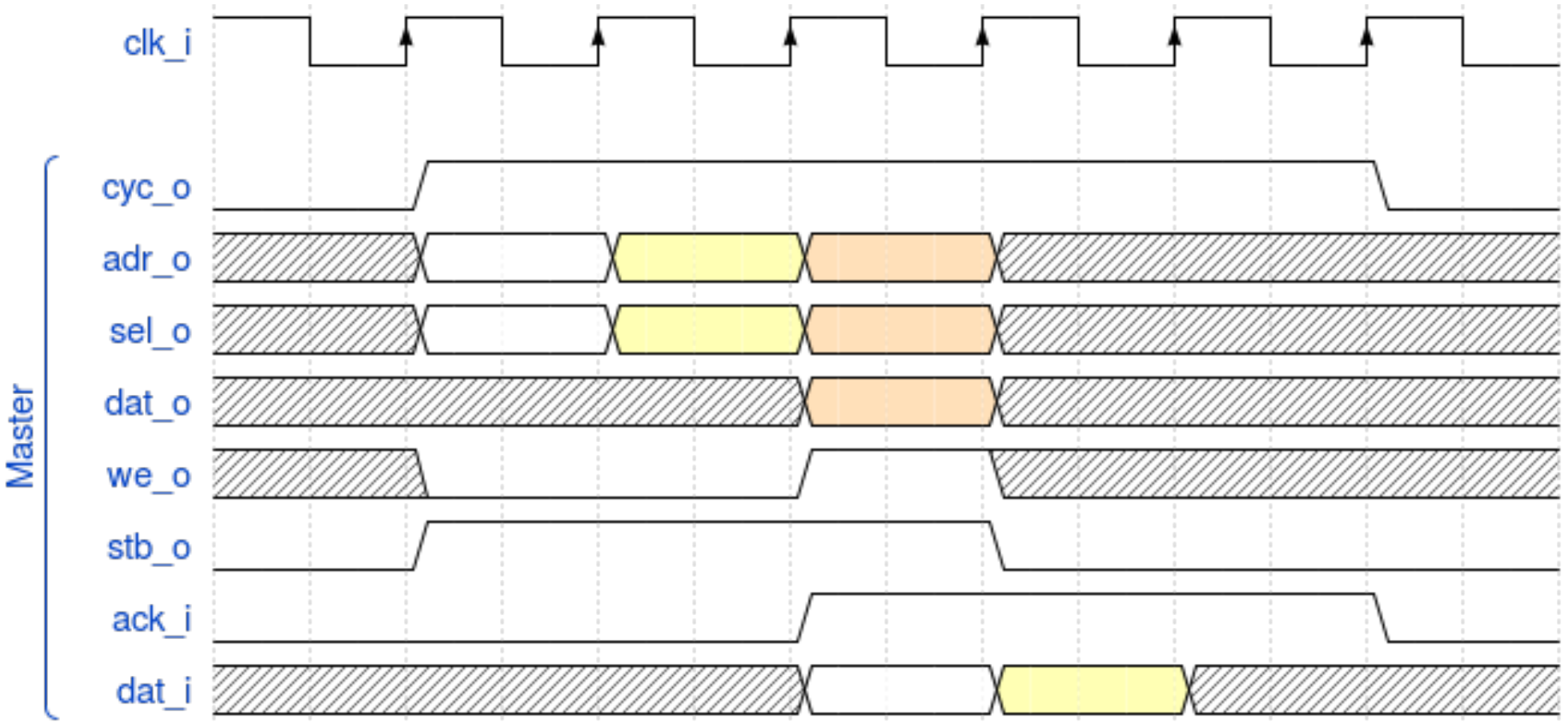
Wishbone B3

Why It Fails for Kestrel-3:

- Improving performance requires introducing multiple clock domains, one for CPU, one for Wishbone bus, one for RAM.
- Requires FIFOs and hard to debug async interlocks to work successfully.
- Works comfortably at somewhere below 20MHz clock speeds for something the size of Kestrel-3.

For anything faster, you start to want something that works more closely *and more locally* with FPGA resources.

Wishbone B4 Pipelined



Wishbone B4 Pipelined

or: How I learned to stop worrying and love the clock!

- CYC_O asserted as long as *at least one cycle remains unterminated.* Can still be used as a bus request.
- STB_O now is the prime determinant in whether or not a bus cycle is valid or not.
- ACK_I no longer scoped by STB_O. It may arrive ***at any time after*** a corresponding STB_O (in order).
- STALL_I (not shown) used to throttle master if required.

Wishbone B4 Pipelined

Address decode latency not cured by increasing clock or introducing pipelining. So why bother?

- Wishbone 3 works great at lower speeds because you have lots of time to generate ACK from address and STB.
- As clock period approaches propegation delay between LUTs, glitches increasingly manifest.
- DFFs work like synchronizers. Every LUT has a corresponding DFF; adding them costs nothing.

Wishbone B4 Pipelined

Address decode latency not cured by increasing clock or introducing pipelining. So why bother?

- Pipelining supports **overlapping bus transactions** without altering address decode latency.
- **Easier to synchronize across different clock domains**.
- **Easier to bridge across different protocols.** (Wishbone to RapidIO, for example.)

Thanks, Everyone! Q&A

Email	Samuel A. Falvo II <kc5tja@arrl.net>
Web	https://kestrelcomputer.github.io/kestrel
GitHub	https://github.com/kestrelcomputer