The Problem(s) with the Minnow PDP-10 Microarchitecture
Rev 0.2

When I first examined the Minnow PDP-10 schematics, I knew that it would be a straight-forward task to implement this in a single small to moderate sized FPGA. I still believe that. I also thought that modern implementation techniques might yield a device that was significantly faster than the PDP-10s of the time. I'm not so sure about that.

Background:
The Minnow PDP-10 was originally implemented using AMD 29xx "4 Bit-slice" devices. The 36-bit ALU is implemented using 9x AM2903 "ALU Super Slices" which handles the mathematics, an AM2904 "Status and Carry/Shift Controller" which controls arithmetic shifts, logical shifts, rotates, and flags. The ALU registers are augmented by 9x AM29705 16x4 Bit "Dual Port RAMs" which increase the number of registers available to the ALU from 16 to 32. The micro-sequencer uses an AM2910 and the interrupt controller uses an AM2914.

Detailed descriptions of all of these devices are available and "cranking" the schematic into VHDL or Verilog is straightforward.

ALU Architecture:
The ALU implementation in FPGA is fairly slow. The main ALU path includes the ALU and a SHIFTER which results in many levels of logic in the ALU path. This could get a significant speed-up from pipelining but this would render the existing microcode unusable. On the good-side, as you would expect, the ALU implementation is extremely efficient: it handles addition, subtraction, multiplication, division, and floating point operations with very few resources and a lot of clock cycles.

Memory Architecture:
The memory architecture of the micro-machine is hierarchical. (We're not talking about PDP-10 system memory here). There are two levels of memory: the 32 word register file attached to the ALU, and an 1K word "RAM File" that is *glued* onto the bus.

The dual-ported registers attached to the ALU are a problem for high speed operation. Most modern RISC processors typically employ thee or four-ported (or more) registers which allow the ALU to perform the following types of operations (pipelined, or course) in a single clock cycle.

$$rD \leftarrow rA \text{ op } rB$$

In the AMD 29xx architecture, this requires at least two clock cycles.

To access the RAM File, one must first write the address to an address latch, perform read or write operation, and then move the data in to or out from the ALU. This all takes many micro-cycles to accomplish. Of course this RAM File is a workaround to the real problem: there aren't enough registers.

The "Y Bus":
You will note by examining the schematic that there is only one bus that moves data around the various parts of the micro-machine. In this implementation, for non-ALU operations, everything comes in the "Y Bus", goes through the ALU and is output on the "Y Bus". Even on ALU operations, the "Y Bus" is in use.

This single thread data path precludes any notion of parallelism.

Interrupt Architecture:
The Interrupt Architecture is schizophrenic. The hardware implementation includes an AM2914 Interrupt Controller - which capabilities are mostly unused. Although the AM2914 is capable of generating vectored interrupts, this capability is not used. Instead, the AMD2914 is used strictly as a maskable priority encoder which is polled on occasion by the microcode.

Strangely enough, the Minnow design includes 8 UARTS; and each UART generates 3 interrupts (RX, TX, Line Status Change) each. These have a separate 24-bit priority encoder. The UART "interrupt" status is read on a different IO port. Again the interrupt status is polled by the microcode.

The Interrupt Architecture presented by the micro-architecture a simulation of the PDP-10 architecture simulated in microcode.

Implementing a real interrupt using real hardware would speed the interrupt processing significantly.

Timer Architecture:
Like the Interrupt architecture, the Interval Timer and the Time Base Register are microcode simulations of the real hardware devices.

Again, actually implementing the timers in the hardware would remove a significant chunk of microcode that is executed as a part of every PDP-10 instruction.

Just to document the insanity of this: a timer flag from the hardware is asserted at a 27 KHz rate. Every instruction checks this flag, and if set executes about 15 instructions to count to 27. Every 27 times (1 KHz), 100 or so more instructions are execute to update the interval timer.

All this saves a few counters.

Pager:
Again, the Pager is mostly a microcode function. There is some RAM in the RAM File allocated for use as Page Tables but all the validity checks are a microcode function.

The Microinstruction
The choice of the microinstruction greatly impacts efficiency. The microinstruction that was chosen for Minnow has only a single 12-bit "immediate data" field. (Probably because the field can be also be loaded into the 12-bit uPC). Loading a 36-bit register with immediate data requires at least 3 microinstructions to get 36-bits of immediate data and yet more microinstructions to concatenate the pieces together. A 18-bit "immediate data" field would ease this burden greatly as half-words could be loaded and stored directly.

To work around this issue, the Minnow microcode wastes a large number of registers and RAM File storing masks and other constants which are loaded at startup and never changed.

The Missing Barrel Shifter and Masker:
When a machine is designed to parse the instructions of another machine it is always nice to have a Barrel Shifter and Masker so that subfields can be extracted from, and inserted into registers in a single clock cycle. This is missing from the Minnow design. The AM29xx ALU requires one clock cycle for every shift position.

General Architecture:
The micro-machine as implemented with AM29xx parts is relatively straight forward and reflects the thinking of the time. The micro-machine is a register-based architecture and uses an IO-like operation read PDP-10 memory, onto the "Y Bus", through the ALU, and into a register.

Of note, it is very generic with only a few PDP-10 specific appendages added. Most of the feature of the PDP-10 platform are simulated in microcode. This leads to yields a dense, efficient, but low performance implementation.

The KS-10 takes a different approach. The KS-10 uses the AM29xx components to implement the PDP-10 micro-architecture not simulate it in software.

Summary:
My comments are not intended to reflect negatively on the Minnow design as it existed. It is what it is. Times were different. The constraints today are different. Today gates are cheap.

Bob Doyle
doyle (at) cox (dot) net