The KS10 Pager

Rob Doyle
doyle (at) cox (dot) net
09 May 2016

During the design of the KS10 FPGA, the Pager has been a perpetual problem. This paper describes the operation of the Pager in detail and outlines my approach to finally resolving the lingering issues.

When the KS10 loads the Virtual Memory Address (VMA), the Pager simultaneously looks up the page translation address and status. This is implemented slightly differently in the KS10 FPGA because the Page Translation Memory employs synchronous block memory whereas the DEC KS10 uses asynchronous memory. That all works correctly.

The Pager essentially 'snoops' the address and memory control signals. When a memory access occurs, the Pager checks the page translation status: Page Valid, Exec/User mode match, and Page Writable (if a write cycle). When a Page Fault occurs, the Pager jams the address of the PAGE FAIL handler in the microcode into the micro-sequencer so that the microcode can handle the page failure. That works correctly also.

Simultaneous to this, the PAGE FAIL logic also generates a 'call' operation to the micro-sequencer which places the address of the faulting micro-instruction on the micro-sequencer stack. The microcode then handles the page fault. When the microcode is done with the page fault, the microcode executes a 'return' instruction which returns to the original micro-instruction and then re-executes it. Presumably the address translation has been updated and the memory operation will not page fail the second time. Note that returning to the original micro-instruction versus the next micro-instruction is very important to the operation of the PAGE FAIL handler. All of that works.

So what doesn't work?

The synchronization between the microcode and the memory cycle is not tightly controlled. The original KS10 memory was slow with respect to the micro-sequencer speed - so the KS10 tends to start memory operations (instruction pre-fetch, etc) as early as possible so as to minimize memory latency. The memory cycle overlaps other micro-instructions that occur during normal operation. Here's the problem: many of these microcode instructions cannot be re-executed without creating side-effects.

I keep re-discovering that the PAGE FAIL signal occurs on the wrong micro-instruction. This causes the KS10 to execute instructions incorrectly. I've seen this failure increment and then re-increment the program counter. I've seen the MOVSI instruction swap and then un-swap the operand. I don't quite understand how this is accomplished in the DEC KS10. Even if I did understand this, the KS10 FPGA backplane is quite a bit different than the DEC KS10 so it would not be exactly the same anyway.

This type of failure is insidious. The basic and advanced instruction diagnostics are executed with paging disabled. I thought they were all working - but - apparently they fail randomly when the paging is enabled. Worse yet, the failure usually only occurs on the first instruction of a new page. Proving that

'fixes' works properly under all conditions has been elusive. In fact, I've thought this was fixed several times already.

What to do?

First. I've hacked the KS10 Console to create some page table entries in KS10 memory, enable paging, load the various diagnostics into KS10 memory from an SD Card, and then execute the diagnostic. Now I can execute the Basic and Advanced Instruction Diagnostics with paging enabled. More importantly I can do this from the Verilog Simulator and see what is happening.

Second. I've hacked-up a special version of the Pager. Normally the Pager handles the page translation for a whole page. This specially hacked-up Pager simply remembers the entire Virtual Memory Address (VMA) associated with the page fill. When a memory access occurs, this Pager will provide the correct page translation when presented exactly the same VMA as the page fill but will generate a page failure for every other VMA. This guarantees that every new memory access will generate a page failure - which just hammers the paging hardware. It is slow but effective.

With the changes outlined above, I can start debugging the diagnostic programs again. This time I know where the problem will be.