

Pushing PicoBlaze – Part1

PicoBlaze, the brainchild of Xilinx's Ken Chapman, are a family of tiny microprocessors or more accurately Programmable State Machines. They complement larger scale FPGA based microprocessors such as MicroBlaze and PowerPC. As such these tiny microprocessors fill a gap in the twilight zone between the worlds of hardware and software.

Today we are going to deal with the FPGA based variants of this family KCPSM, KCPSM2 and KCPSM3 and how to get a little bit more out of these processors.

All variants of PicoBlaze have a restricted address space with FPGA variants offering between 256 and 1024 address locations. This is the principal limitation of PicoBlaze but is also one of its strengths. Restricting addressing space is the main reason why these processors are so small.

Paging memory is one the oldest techniques known to processor designers and software engineers allowing extension of addressing range or code space in microprocessor systems. To make use of this technique usually needs the software to write to a page register. This page register is usually connected to the upper address lines of memory device thereby changing the code that the microprocessor "sees" within its addressing range. This paging switch usually assumes that software execution will subsequently jump to the area of paged memory or that a mirror location of the current execution exists. Mirror locations are simply address locations that have the same or complimentary execution code such that when paging switch occurs there are no noticeable effects in code execution. The downside of all paging schemes is that something or someone has to keep track of page mapping locations for sub-routines. In processor families like X86, tools were developed to handle page switching. As yet there are not any software tools for PicoBlaze that will support this kind of feature. Consequently the software designer must maintain his own subroutine page mappings and be prepared to hard code page offset values into their code.

So how did we fit a paging scheme to a PicoBlaze? PicoBlaze differs from many microprocessors that use paging schemes in that it operates within a FPGA fabric. Usually it uses an internal BlockRam (used as BlockRom) to provide to code memory space and not an external memory that could be page switched. A set of BlockRams, or sections of one BlockRam, could be switched in and out in a similar fashion to a discrete external memory. The main drawback to doing this is that BlockRams are valuable FPGA resource to burn unnecessarily. So what else can we do? As it happens BlockRams are inherently dual port memories. This feature allows the contents of a BlockRam to be changed whilst PicoBlaze runs from the memory. While PicoBlaze is connected to one port of the BlockRam, the other port can be driven by a loader which can modify memory locations either byte by byte, or otherwise using port widths in the range of 1 to 36 bits.

We now have a slightly different arrangement to that of the tradition paging mechanism. There are limits as to how fast the complete memory contents can be replaced but conversely a small section of memory contents might be all that needs to be replaced. It is even possible that only a small part of the sub-routine might be loaded before

execution of the sub-routine starts. Starting before completion of loading needs care in use and depends on knowing what has to be replaced first.

This idea can be extended further. Replacing the single sub-routine within memory, or even on a ping-pong basis between two working sub-routine memory sections now opens up a much larger application space. The software structure is now starting to look like a permanent kernel, with sub-routines being loaded in and out by the external loading mechanism, or dare I say memory manager. We are now starting to approximate what programmers, operating at higher levels, do as standard in C and C++ programs i.e. using a kernel calling sub-routines.

So what are the limits of this technique? The first limitation is the maximum size of code. Let's use half the addressing range as a rule of thumb for maximum length. So you have between 128 and 512 address locations. If the subroutine ends up bigger then you will need to break it down into a number of smaller chunks. The next limitation might be the number of sub-routines, and their order, that will be stored as part of the kernel. However we can offer the kernel some assistance in this case. Using an external sequencer we can supply subroutine order and type. More complex than including the order in the kernel but the code capability of PicoBlaze has just risen significantly. These parameters can be passed to PicoBlaze via its I/O interface registers.

The next problem is how to control the loading of a sub-routine. A good way to do this is to use a simple semaphore consisting of a single output bit indicating service request and a single return bit indicating routine loaded and ready to continue. In conjunction with these signals one or two bytes of output I/O can be used to indicate which subroutine to load and, if implemented, one or two bytes of input I/O when an external sequencer is used to indicate the next subroutine.

Having put in place this loading control you need to consider what will do the loading. You can use a dedicated state machine to do this. If this is too inflexible for your needs why not use another PicoBlaze, with a DMA engine bolt-on, to do the task. If your design uses multiple PicoBlaze processors you can share this loading resource to make efficient use of your FPGA resource.

A further variation of these mechanisms to consider is something used here at Enterpoint. We use an external flash memory in conjunction with a FPGA embedded control engine to expand the range of PicoBlaze designs. Our development example uses a Broaddown2 development board to provide FPGA resource and location for a flash memory. Broaddown2 has a host Spartan3 and a series of expansion headers. These headers are standard 0.1 inch socket headers and accommodate standard pitch packaged devices. We have fitted a flash memory into these headers for purpose of our mechanism development.

In summary, PicoBlaze is an opportunity for innovation. Don't be tram lined into thinking that it is just another microprocessor or microcontroller. The ability to extend the reach of PicoBlaze using add-on features extends its reach beyond that of many of its competitors.

John Adair

© 2005 Enterpoint Ltd. - Xilinx Xperts Partner