

Designing Multiprocessor SOCs

Using Xilinx EDK tools and IP, you can scale your applications by designing SOCs with multiple processors.

by Vasanth Asokan
Staff Software Engineer
Xilinx, Inc.
vasanth.asokan@xilinx.com

Given the rapid growth of embedded processing requirements, system architects are turning toward multiprocessor designs to solve the twin problems of burgeoning complexity and inadequateness in single processor systems.

With their high logic density and high-performance hard blocks, recent generations of FPGAs have made powerful chip multiprocessing (CMP) solutions a reality. The challenge now lies in how you can rapidly explore and create designs in this solution space.

Xilinx® Embedded Development Kit (EDK) tools and IP provide the flexibility to create uniquely crafted, customized multiprocessing solutions on FPGA logic real estate that can meet both price and performance targets. In this article, I'll give a broad overview of multiprocessing concepts as they apply to Xilinx solutions based on PowerPC and MicroBlaze™ embedded processors.

Scenarios

Performance and functional partitioning are compelling reasons to design systems with multiple processors. In general, there are some commonly encountered scenarios where multiprocessing can help:

- Multiple independent functions. The design may have multiple, independent sets of processing tasks. An attractive way to address this challenge is to create independent processing modules dedicated to each processing task, assigning each processing module a unique processor and peripheral set.
- Control or data-plane offload. One common scenario is the presence of a distinct set of real-time (compute- or data-intensive) and non-real-time tasks that might cause a non-response in solutions based on a single processor to be non-responsive. In these cases, you can dedicate a slave processor to perform the real-time task in a timely manner. This leaves the master processor to perform other regular tasks and serve as an interface to the host system. The master processor also monitors and controls the slave processor. The slave processor may contain specialized functions or interfaces, allowing it to meet computation performance requirements. Some examples of this scenario include network offload, media processing, and security algorithms.
- Interface processing. On systems that act as a bridge for or switch between multiple interfaces, you can dedicate a slave processor to the processing of data at each interface, while one or more master processors perform higher level bridging and switching tasks.
- Stream processing. For handling stream-oriented computation, you can arrange

processors to act on the data stream in a pipeline fashion. Each peer stage in the multiprocessor pipeline acts on one portion of the computation before passing it to the next processor. This is an effective way to increase system throughput.

- Reliability and redundancy. You can replicate processing systems multiple times to provide reliability and redundancy.
- Symmetric processing. Traditional symmetric processing (SMP) is a useful solution with which you can scale up (by adding more processors) the performance of applications that do not possess clean partition boundaries. An SMP-capable OS layer manages parallel tasks and automatically schedules them across multiple processors. The SMP use model cannot be applied to Xilinx processors, however, because they lack cache coherency, a requirement for implementing SMP.

Apart from the SMP scenario, all other scenarios are feasible on Xilinx FPGAs with EDK tools. The unique capability of Xilinx processing solutions is the flexibility to customize each of the processing subsystems to the application requirements. For example, not all processors may need a cache or a floating-point unit. By assigning specific functions to specific processors, you can create a tailored solution that meets all design goals.

A Simple and Scalable System Architecture

As you can see, a number of use models are possible for multiple processors. There are also a large number of possibilities for the system architecture. Reconciling a use case to a clean, scalable topology and architecture can be daunting, so it helps to define a baseline architecture that fits most needs.

Figure 1 illustrates a dual-core architecture. This architecture presents a simple and scalable multiprocessor system definition. You can generate derivative topolo-

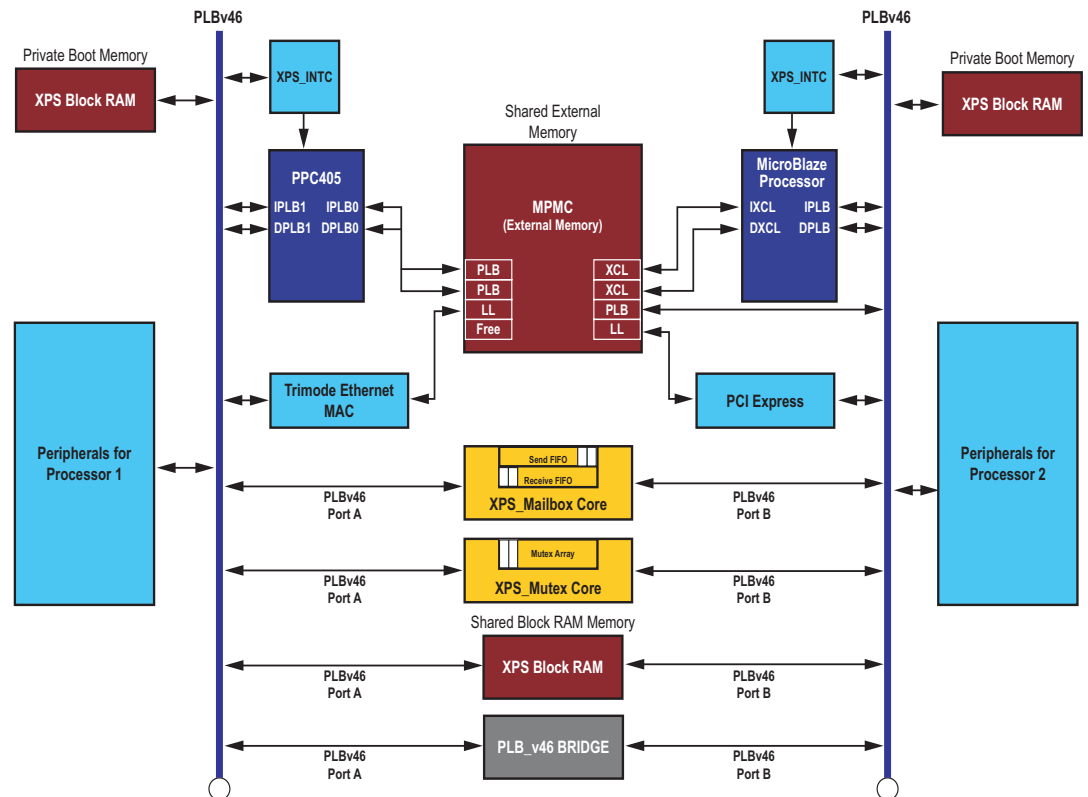


Figure 1 – Dual processor architecture

gies, starting from this definition, to meet design constraints or challenges. The key concepts of this architecture are as follows:

- The architecture is a simple extension of two completely independent single processing systems, formed by linking the systems together with communication components.
- The shared components are all multi- (or dual-) ported in nature. The multiported nature of these components allows each processor's system bus to be independent of the other, both in terms of static as well as dynamic load. By isolating each processing subsystem, you ensure that the system bus is not locked out for a processor or peripheral because of a transaction executed for another processor. All multiported peripherals arbitrate accesses on various ports internally.
- The key shared peripheral is the multiported memory controller (MPMC). MPMC provides access to external memory through different port interfaces. Multiple processors can connect to MPMC through independent ports. This topology allows both the PowerPC and MicroBlaze processors to simultaneously access external memory with minimal latency and high bandwidth. MPMC currently provides a maximum of eight ports, thus allowing as many as three to four processors to connect to a single external memory.
- The architecture also shows sharing of internal block RAM memory between processors. Sharing on-chip block RAM can be an extremely fast way to pass kilobyte-sized data between the processors. Accesses to block RAMs can also be deterministic – an important requirement in some applications.
- Apart from shared memory, there are two other cores – the XPS Mailbox and XPS Mutex – that provide simple forms of interprocessor communication. The XPS Mailbox core provides a low-latency, FIFO-style message-passing interface between the two processors synchro-

nously or asynchronously. It can be used for either directly passing messages or for passing pointers to messages stored in shared memory. You can use the XPS Mutex core for arbitrating accesses to shared resources (whether they are on-chip or off-chip) between software on the two processors. Together, these cores allow you to build cooperating software programs on each processor.

- Some systems might wish to share peripherals that are not multiported (like a UART or SPI or I²C). Such a situation requires providing a system bus bridge from the bus that does not connect to the peripheral to the bus that does connect to the peripheral. Figure 1 shows the use of a bus bridge to share a UART between the two processors.
- Figure 1 intentionally shows a PowerPC 405 processor as the first and a MicroBlaze processor as the second to illustrate certain specific characteristics of each processor. However, any one processor can be equivalently replaced by the other with very minimal adaptation. Thus, this architecture can be seamlessly transitioned between processors.

Although Figure 1 illustrates the recommended overall multiprocessing architecture, various constraints may require you to further refine the architecture. For instance, in a system in which logic area and resource usage are a key concern, all of the processors could be connected to the same system bus. Although this makes the system less deterministic and increases run-time load on the bus, it offers area savings by eliminating a new system bus as well as removing the need for multiple ports on IPs.

Other derivative architectures are also possible, such as having the high-performance processor on a separate system bus and multiple low-performance processors on a shared system bus. You can also create hierarchical topologies by connecting processing subsystems to each other through multiple levels of bridges. The various tools and IP provided in EDK allow you to further refine this base topology down to something that exactly fits your needs.

Other Considerations

You will typically apply a few other considerations to your multiprocessing architecture. For instance, you will need to define memory maps in a non-conflicting manner between the two processors. The automatic address generation tools in EDK simplify this task down to a push of a button.

You will also need to give some thought to your clocking and reset networks. You will have an option to clock all of the processors at the same rate or use different clocking domains for each processor. Similarly, you can define reset domains at various levels such as processor-only reset, processor subsystem reset, and system reset. The processors must also be independently connected to a debugger port, thus allowing separate debugging sessions for each processor.

Beyond the hardware considerations, you will also be designing your software systems such that they can work cooperatively. This includes using shared memory, message passing, and common synchronization concepts such as barriers and rendezvous so that the system behaves in a predictable and synchronized fashion. Commercial software stacks are also available that can provide higher level communication paradigms.

Conclusion

For a more details about the possibilities of multiprocessor systems, a longer white paper version of this article is available at: www.xilinx.com/support/documentation/white_papers/wp262.pdf. Also consider the reference designs described and provided in Xilinx application note XAPP996, “Dual Processor Reference Design Suite,” at www.xilinx.com/support/documentation/application_notes/xapp996.pdf.

Stay tuned for new features in future EDK tools, such as support for automated multiprocessor design creation and cooperative debugging. The Xilinx Virtex™-5 FXT platform, with the powerful PowerPC 440 processor, also opens up endless possibilities for creating ultra-high performance multiprocessor systems. ●●