

# Making the Most of MOST Control Messaging

This case study presents the design of Xilinx LogiCORE MOST NIC control message processing.

by Stacey Secatch  
Staff Design Engineer  
Xilinx, Inc.  
[stacey.secatch@xilinx.com](mailto:stacey.secatch@xilinx.com)

At only 2,600 slices, the Xilinx® MOST (Media Oriented Systems Transport) network interface controller (NIC) occupies roughly half of a Spartan™-3E 500 device, which leaves plenty of room for the micro-processor and other peripherals. This full-featured embedded core provides easy access to data and network control as a slave or master node.

In this article, I'll present a control message processing design case study and show how the MOST NIC is able to autonomously handle messages without any host software intervention. With careful planning, the minimal-resource PicoBlaze™ configurable state machine can handle control messages correctly and efficiently.

## Control Messages on a MOST Network

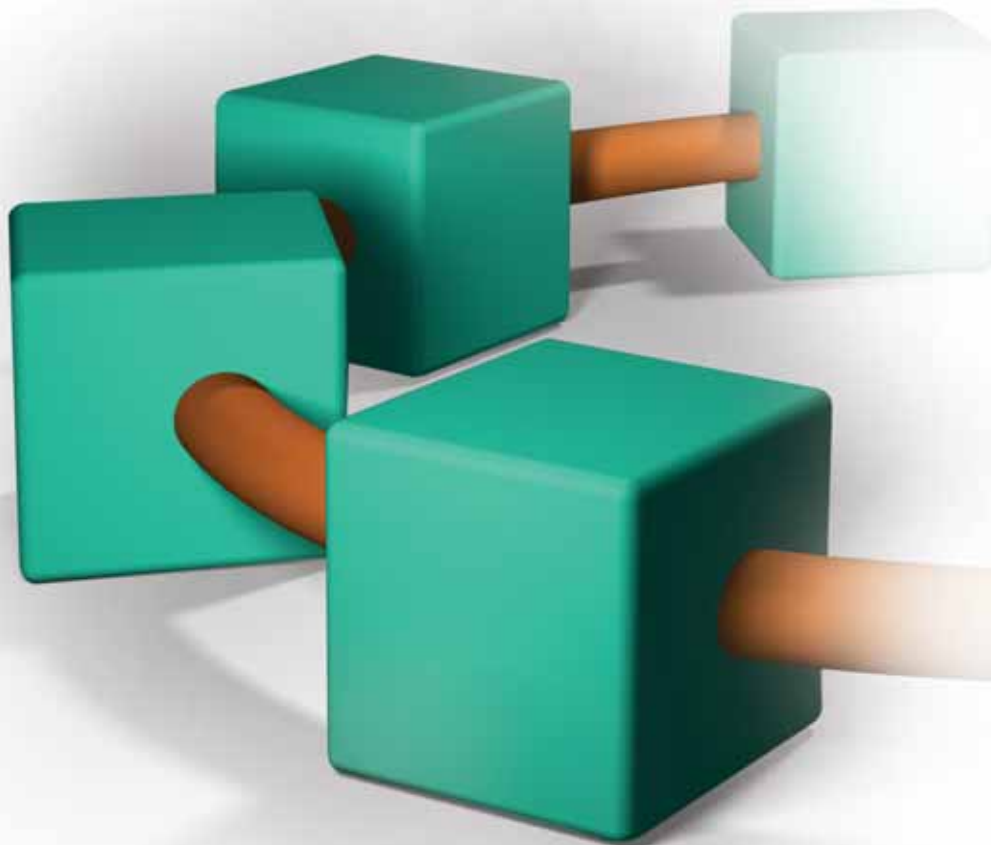
MOST is an automotive infotainment-oriented network standard with a ring topology. One of its strengths is that it supports multiple simultaneous streams of media intended for several nodes in the ring. However, like any network, more than just data is sent between nodes; control communication is also required. A simple MOST ring network is shown in Figure 1, with one network master and several slaves. Six different control message types are available. Let's focus on the three that are most commonly used during operation of a MOST network.

### Normal Messages

Normal messages are distinct from other types in that they are intended for high-level application use in the host processor. All other messages are used for ring operation and debugging. For example, as part of the boot sequence of the ring, the master could send a normal message to determine the identification of slave B. Normal messages may be combined in the application layer when there is more information than one message can hold.

### Channel Allocation and Deallocation

Allocation messages are sent to the master node to gain access to the portion of the frame set aside for media data. The requestor indicates how many byte-wide channels are desired and the responder (master) confirms both the status of the request (granted or denied) and which channels have been granted. Deallocation messages are similar to allocations, where the requestor indicates which channels to deallocate and the master responds with deallocation granted.



**Choosing the Hardware/Software Partition**

Embedded design involves choosing the best boundary for the hardware/software interface in terms of functionality and resources. Choosing to implement too much in hardware is costly because of the additional resources required to implement the design. Choosing to implement too much in software can result in an overburdened microprocessor and stalled logic, possibly even resulting in a non-functional design if the processor drops events.

For the MOST LogiCORE™ solution, we found a clear boundary for dividing up control message processing.

**Delay Time Requirements**

Control messages are broken across multiple frames. As messages travel around the ring, specific handshaking is required between the communicating nodes, with a turnaround time of roughly one of those frames (about 23 μs). Message-response generation usually requires calculations. For example, allocation responses depend on determining whether enough free channels are available.

Because this service requirement is so tight, it makes sense to place control message handling within the core as hardware, leaving the external processor available to run the network stack and media-oriented user application.

**Items to Leave to Software**

Although nearly all control message types can be handled completely by the LogiCORE system, normal control messages must be passed on to the application. Therefore, the MOST NIC LogiCORE solution provides interrupts to signal that a normal message has been received and is available for processing. And because all messages are initiated by the application, there are configuration registers for the application to indicate that a message is available for sending and interrupts to alert the application that the message sequence is complete.

**Selecting the Implementation**

The next step after deciding to process control messages in hardware was to choose the most efficient design possible. We could

have decided to implement this in traditional fashion as a custom state machine. Preliminary estimates placed the size of the state machine at more than 1,000 slices, plus a block RAM to contain all of the message buffers. Note that control messages only use 2 bytes out of a 64-byte frame, yet processing just these two bytes would have accounted for more than a third of the core.

Instead, we chose to incorporate the PicoBlaze configurable state machine as the backbone for control channel processing and minimize core resources. In our implementation, this includes two block RAMs to hold the instruction memory, approximately 100 slices for the PicoBlaze component, and 150 slices for peripherals, as well as the original block RAM for message buffering.

Choosing a PicoBlaze controller offers you similar opportunities for resource minimization. You will probably also see the same benefits that we did, namely a shortened design cycle and easier code maintenance thanks to the simplicity of coding message processing procedurally.

**Organizing Data Storage**

Determining what kind of memory to use for variables when designing an RTL state machine is rather straightforward. In general, you use RAM for large buffers and stick everything else in a register. When using a PicoBlaze controller, you need to do a bit more up-front planning to minimize resources while still providing access to all of your data.

**State Machine Variables**

The 64 “free” scratch pad registers are ideal for storing variables that you will never use outside of the PicoBlaze controller. For example, the accumulators to store intermediate checksum results for MOST control messages are only needed by the state machine.

**Larger Buffers**

Block RAM is an ideal location for buffering. Although the PicoBlaze controller only provides direct access to 256 locations, it is probably more efficient for you to use one block RAM than to instantiate distributed

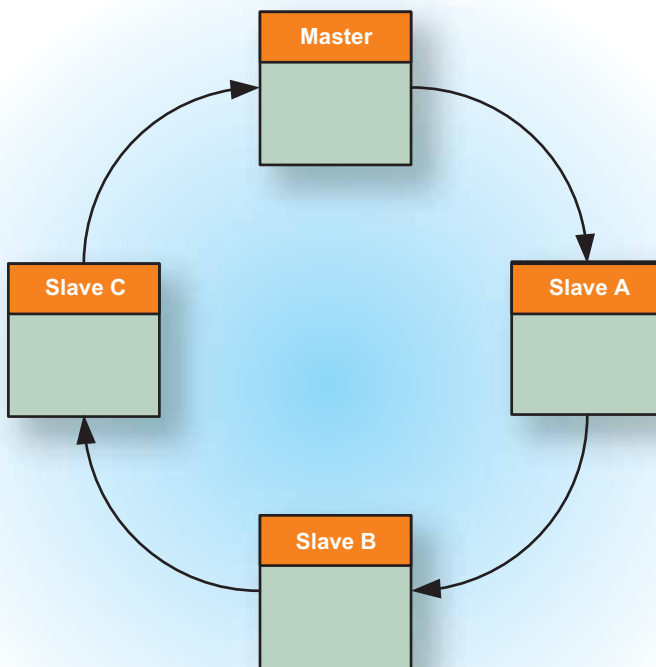


Figure 1 – A sample MOST ring

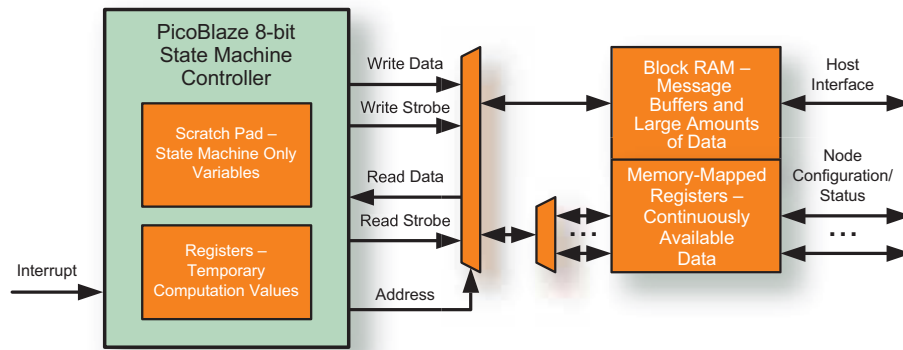


Figure 2 – Memory organization for Xilinx MOST NIC control message processing

memory for all of your buffers. And because a PicoBlaze configurable state machine executes sequentially, you will never need to access more than one buffer at a time, which makes grouping together all of your memory easy. As a bonus, Xilinx block RAM automatically converts data widths, seamlessly allowing 32-bit access from the host side and 8-bit access from the PicoBlaze controller side.

**Shared Variables**

Some configuration and status variables will need to be continually available. You can map these variables into registers in the PicoBlaze controller external memory space for your state machine to access, just like the memory in the MOST LogiCORE control processing module (Figure 2). In the LogiCORE system, the PicoBlaze controller external memory port is mapped to a shared block RAM, with the exception that the end of the memory range is memory-mapped registers.

As an example, the currently programmed addresses of the MOST node are mapped into a read-only “register” (a mux input) to determine if a received message is meant for this node. Status information about the current occupancy of the buffers is placed into a write-only register for access by the external host.

**Ordering Your Processing**

Complex state machines are generally built from smaller interlocking state machines that operate in parallel. PicoBlaze configurable state machines execute sequentially, providing advantages in terms of conflict

avoidance and race conditions. You will need to plan carefully, however, to ensure that your event scheduling is correct and that that no queued events get dropped. Here are some guidelines that might help in designing your PicoBlaze application:

- Prioritize your event processing. If you have events that are not important, choose to execute those last. For example, the value of buffer occupancy counts is changed by a MOST host buffer read. However, this update may be deferred for several frames. Because responding to a message must complete in one frame, the core services message responds first.
- Preserve event ordering. If your processing requires that one event complete before another, you will want to execute the code for the first event, even if the second event may have a tighter timing requirement. For example, control bytes must be sent out before the bytes in the current frame

are even received. Therefore, for a given frame, transmit events are always processed before receive events.

- Consider using a busy-wait loop. If you have a very long event that cannot be processed in the time frame required by other events, you might consider placing the long event in the main processing wait loop.

For example, preparing a MOST control message for transmit can take multiple frames to complete. We could have set up multiple levels of interrupt servicing to ensure that we could still process frame data, but this type of task fits very well into a busy-wait loop, which automatically gives it the lowest priority.

If you do choose to process data in the main loop, some additional care is needed to ensure that you do not corrupt your main loop when servicing interrupts. At the start of interrupt servicing, you should copy all registers that might get overwritten, and restore those registers as interrupt servicing completes. There are also PicoBlaze commands for disabling interrupts when executing critical portions of your code.

**Conclusion**

The Xilinx MOST NIC has been shown to efficiently handle control message processing. It is ideal for automotive infotainment systems, as it provides similar efficiency in streaming data. With upcoming device support for the Spartan-3A DSP family, this will allow even more efficient multimedia processing for your design.

**TAKE THE NEXT STEP** (Digital Edition: [www.xcellpublications.com/subscribe/](http://www.xcellpublications.com/subscribe/))

- Get more information about embedded design using the Xilinx MOST NIC LogiCORE solution in the MOST NIC lounge.
- The MOST NIC User Guide, available after generating a LogiCORE implementation, provides more information about all MOST control message formats for the MOST link layer.
- Download the PicoBlaze microcontroller reference design, which includes supporting files such as the assembler and extensive documentation such as the User Guide, with many coding examples and documentation for the complete instruction set.