Build your own VMEbus interface chip using open source

By Wade D. Peterson

The make versus buy decision

One of the first choices that an engineer makes when designing a new VMEbus board is whether to make or buy the interface chips. In most cases it makes sense to buy an off-the-shelf, proven design. In other cases the custom interface is the only way to go, especially for special requirements such as:

- Board service life longer than five years
- Low voltage interfacing (e.g., 3.3V)
- Wide temperature range
- Radiation hardening
- JTAG Boundary Scan
- Special packaging
- Low cost/high volume interfaces
- Integration of custom peripherals
- Improved security

Perhaps the biggest advantage of custom interfaces is board service life. Traditionally, VMEbus board makers have enjoyed many choices when it comes to interface chips. However, all chips eventually become obsolete, and when they do the board manufacturer must either redesign or discontinue the board. If the board is discontinued, then most manufacturers will offer a lifetime buy opportunity. However, lifetime buy options require a substantial capital investment on the part of system integrators, which they may or may not be willing to make. As we’ll see shortly, open VMEbus interfaces can alleviate this problem.

In many cases the board requires special environmental or operational characteristics. Low voltage operation, wide temperature ranges, and radiation hardening are three common requirements. For example, several Mars landers have used VMEbus boards with custom interfaces because they needed chips that consume little power, operate over wide temperature ranges, and tolerate high radiation levels.

One common complaint about off-the-shelf VMEbus interfaces has been the lack of JTAG Boundary Scan capability to test the interconnections between the chips on the board. JTAG Boundary Scan technology has become increasingly popular as chip packaging has moved to highly dense parts such as ball grid arrays and flip chips. JTAG simplifies the task of testing the board and is available on most FPGA devices.

The major drawbacks to custom interfaces are generally development time and cost. Custom chips can take weeks, months, or years to develop. They can cost anywhere from a few thousand dollars to hundreds of thousands of dollars. However, these drawbacks are changing with the advent of new technologies such as:

- Low cost FPGA and ASIC chips
- Better development software
- Open IP cores

The development cost of both Field Programmable Gate Array (FPGA) and Application Specific Integrated Circuits (ASICs) has fallen dramatically in the last decade. The main reasons for this are lower manufacturing costs, streamlined distributing, and much better software development tools.

Another area where costs are falling is in the Intellectual Property (IP) that is placed on each chip. Chip designs are very labor intensive and take a long time to develop. However, new design reuse strategies are changing this. At the cutting edge of this trend is open IP, where reusable chip designs are traded in the same way that software is traded in the GNU/Linux community. For example, the OpenCores organization at www.opencores.org uses WISHBONE to connect their cores together.

The OpenVME project

This article will describe two publicly available VMEbus interface cores. These can be downloaded for free and implemented on widely available FPGA and ASIC devices. The first is a VMEbus SLAVE interface called VMEcore. It can be used as a basic building block for any VMEbus interfacing project. The second is a VMEbus to PCI Bridge SoC. It uses VMEcore as a component in a much more sophisticated System-on-Chip.

The OpenVME project at www.openvme.org makes both these cores available. The OpenVME project is a consortium of industry veterans and newcomers who are volunteering their time to create a set of open VMEbus interfaces.

OpenVME was spearheaded by Ken Boyette of Critia Computer and myself and is based on the Apache Software Foundation at www.apache.org. That organization was formed to create and maintain GNU/Linux compatible software for the Apache Web server. Although its purpose is to foster open VMEbus interfaces,
needed because FPGA and ASIC chips lack enough drive current to external driver and receiver (buffer) chips. The buffer chips are the timing parameters also account for time delays through the internal logic elements and signal propagation delays through the internal logic. A process called synthesis converts the VHDL source code into machine instructions that the different brands of FPGA devices understand. The whole process is roughly analogous to how software, which is written in a high level language like C, can be compiled to run on a desktop PC or a Macintosh.

A very difficult problem

VMEbus interface cores are surprisingly rare in the industry. Such cores are rare because VME is asynchronous, meaning any central clock does not coordinate the backplane signals. This can be a very difficult problem when interfacing to programmable chips, which generally use synchronous design practices. To make matters worse, VMEbus interfaces usually require delay lines to guarantee their timing parameters. These lines are not available in most FPGA or ASIC chips, and so the timing parameters must be guaranteed in other ways.

In FPGA parts the timing parameters are entered into the manufacturer’s routing software. The timing parameters limit the propagation delays through the internal logic elements and signal traces and prevent race conditions from occurring in the chip. The VMEcore was designed so the parameters enforce the VMEbus timing specifications, too.

The timing parameters also account for time delays through the external driver and receiver (buffer) chips. The buffer chips are needed because FPGA and ASIC chips lack enough drive current to drive all the VMEbus backplane signals directly. The buffer chips are placed between the target device and the VMEbus connector.

The VMEcore interface solves these timing problems with some elegant strategies. The most important was the basic design of the WISHBONE back end interface. The WISHBONE SoC was originally designed as the back end of a VMEbus interface, allowing the two sides of the interface core to mesh together quite easily. In fact, WISHBONE can be thought of as a synchronous version of VMEbus.

Easy interface timing

One early VMEcore interface design goal was to make the interface easy to integrate. For one thing, this means that the system integrator shouldn’t have to fuss with too many timing constraints in the final design. The VMEcore interface only requires that three timing parameters be known:

- One clock input-to-clock setup time
- One clock flop-to-flop transition time
- One clock-to-output delay time

Figure 2 shows the timing constraints for all of the VMEbus input and output signals. There is also a bidirectional case that isn’t shown.

The input-to-clock setup time is the time it takes for a VMEbus backplane signal to propagate through an input buffer to the input of a flip-flop inside the target device.

The flop-to-flop transition time is the time it takes for a synchronous signal to propagate from the output of one flip-flop and be set up at the input of another. Stated another way, it’s the internal timing for the core’s Register Transfer Logic (RTL).

The clock-to-output delay time is the time it takes for a synchronous signal to exit a flip-flop and propagate through a VMEbus output buffer.

The core works with standard VMEbus buffer chips that are available in a variety of configurations. Integration with the buffer chips is even easier than with standard VMEbus interfaces because the core is delivered as open IP, meaning the SoC integrator can easily change the buffer control logic.

The synchronous RTL VMEcore clock is called [VCLK]. It is used as the sampling clock for the asynchronous VMEbus signals and also as the WISHBONE interface clock. The clock must be fast enough to sample the data strobes that, at their minimum low or high time, are 30ns long. That means that the minimum clock speed for the interface is 1/30ns, or 33.33 MHz.

The maximum clock speed is dependent on the data strobe skew of the backplane. The VMEbus specification limits the skew to 20ns, which results in a maximum frequency of 1/20ns, or 50 MHz. When the core is operated at the maximum speed, each read
or write operation requires an average maximum of 2.5 clock cycles, or 50ns, to complete. When the interface is connected to an equally fast MASTER it can operate at the maximum VMEbus data transfer rate.

The WISHBONE back end
The front end of the SLAVE interface is connected to the buffered VMEbus signals, and the back end is connected to a WISHBONE MASTER. WISHBONE is a completely synchronous bus that can be connected to other peripherals and memory devices. Data can be exchanged with other cores using point-to-point, shared bus, or crossbar switch interconnections.

The operation and features of WISHBONE were discussed in the last issue of VMEbus Systems and can be found online at www.vmebus-systems.com using an article search for WISHBONE. If you missed that article there is also a tutorial available at www.silicore.net/wishfaq.htm.

A VMEbus address decoder determines when the core responds to bus cycles. The core is delivered with a five-bit address comparator. It can be connected to a dipswitch, a software register, or the VME64 geographical address pins. The user can completely control how and where the address is decoded because the address decoder arrives as an open core.

The WISHBONE back end interface supports 8-, 16-, and 32-bit data transfers. When the VMEbus interface participates in a bus cycle it detects which VMEbus byte lanes contain data and automatically routes them to the correct location on WISHBONE. This makes the core completely transparent to the user.

VMEbus to PCI bridge SoC
The second core example is the VME64 to PCI bridge System-on-Chip. This core allows data transfer between a VMEbus slave interface and a PCI target interface. It is delivered as a VHDL soft core that is intended for use on the Xilinx Spartan 2 FPGA. However, with some modification it can be implemented on other brands of FPGA or ASIC devices. System integrators can also add other cores to the interface, making it extremely flexible and versatile. Figure 3 shows a functional diagram of the bridge.

Communication between the two sides of the bridge is made through a set of four control registers, seven semaphore registers, and nine shared memory buffers. Data is loaded into a buffer on one side of the bridge and unloaded from the other side. The system software can also use the semaphore registers to determine when and if data is available in the buffers.

VMEbus core developers used this topology to simplify the bridge and improve its performance. While it is theoretically possible to directly connect VME and PCI bus cycles with each other, it is rarely done because it’s so inefficient. The shared memory topology allows both sides of the bridge to run at, or close to, their maximum speeds.

The bridge uses the A24:D32:D16:D08(E0) VMEcore SLAVE interface described previously. In fact, the bridge SoC is an excellent example of how to integrate the VMEcore interface.

Each internal memory buffer has its own arbitration circuit to prevent simultaneous data transactions (from both sides of the bridge) from corrupting data as it passes through the bridge.

The buffer memories are split into eight memory areas of 1 Kbyte each. This feature can reduce memory access conflicts to improve the speed of the bridge.

Because this is a soft system, each of the buffer memories can be removed and replaced with a different WISHBONE compatible core. When the bridge is viewed in this way the central buffer memory area resembles a 16-slot microcomputer bus backplane.

The bridge SoC is provided as open source code under the GNU Lesser General Public License. This is the same license that is applied to the VMEcore interface described previously. That means that anybody can download, use, modify, and share the bridge without any royalties or other fees. However, the bridge SoC uses a Xilinx LogiCORE PCI core with a WISHBONE wrapper on the backside. This is a commercial core that must be purchased from Xilinx, although other high quality PCI interfaces (with WISHBONE back end buses) are available and could be substituted for the Xilinx PCI core.

The PCI target interface responds to the following bus cycles:
- Configuration read
- Configuration write
- Memory read
- Memory write
- Memory read multiple
- Memory read line

The PCI interface responds to 64 Kbytes of memory I/O space starting at the address programmed in the BAR0 register. All registers and buffer memory areas located on the PCI target interface have port sizes of 32 bits (DWORD). All have 16-bit (WORD) granularity, meaning that they can be accessed as 16- or 32-bit quantities. The bridge uses a 16/32-bit data interface, which means that it can read and write 16-bit WORD or 32-bit DWORD values. The interface does not support BYTE accesses. All PCI accesses to unused address areas are terminated with a PCI TARGET ABORT.

Special wrapper circuitry connected to the Xilinx LogiCORE PCI interface controls how the target interface responds to the PCI commands. During the initial phase of a PCI burst cycle, a binary counter (located inside the wrapper circuitry) latches the PCI starting address. The counter increments during every data transfer, thereby maintaining the current address of the data transfer. It is incremented after every cycle that accesses the high byte of a 32-bit DWORD transfer. That means that the address counter is incremented after every 32-bit transfer or after a high order 16-bit transfer.

The bridge supports PCI single and burst transactions, but the control registers for all the memories within the bridge do not support burst transactions. If a burst transaction is tried in a memory region that does not support burst transfers, then the bridge responds with a TARGET ABORT termination. The core is implemented as a PCI target, meaning that it cannot initiate bus transactions.

The PCI core returns a PCI Device ID and a PCI Revision ID that are unique to the VMEbus to PCI bridge core. These are returned from PCI configuration registers 0x00 and 0x08 respectively. The PCI Device ID always returns 0x030. The PCI Revision ID identifies the hardware revision level of the bridge.
The internal clock frequency of the bridge is 33 MHz. A single clock operates the VMEbus interface, the two internal WISHBONE buses, and the PCI core.

EEPROM interface
The VMEbus side of the bridge also includes an interface for programming Atmel AT17 series EEPROM devices. These are called *configurators* because they are specifically designed to configure (program) a variety of FPGA parts from several manufacturers. This allows the programming of other FPGA devices, located on the same VMEbus module, from the VMEbus interface. This powerful feature allows hardware upgrades to be supplied with the system software and saves a great deal of time and money during hardware enhancements.

The EEPROM interface named CEEPROM in the bridge documentation forms the standalone interface shown in Figure 4. Communication with the core is made through a 32-bit WISHBONE data bus. The core itself runs at about 400 KHz, much more slowly than the 33 MHz WISHBONE SoC bus. Data transactions with WISHBONE are synchronized with the slower, local clock. The core includes a clock divider to generate the slower EEPROM clock.
Conclusion

VMEbus compatible interfaces can be created on programmable devices such as FPGA and ASIC chips. This strategy has many advantages over standard parts such as long service life, integration of special functions and a wide choice of packages, environmental conditions, and voltage ranges. Using portable design practices, such as combining WISHBONE SoC and open IP use, can render the interface chip nearly immune to parts obsolescence.

Wade Peterson is president and CEO of Silicore Corporation, a consulting firm specializing in electronic design, sensors, and semiconductor cores. Wade is the principal author of the WISHBONE System-on-Chip architecture and has more than 20 years of design experience in industrial controls, sensors, software, and chips. Wade holds a BSE degree in Electrical Engineering from the University of Minnesota.

For further information, contact Wade at:

Silicore Corporation
6310 Butterworth Lane
Corcoran, MN 55340
Tel: 763-478-3567
Fax: 763-478-3568
Email: wadep@silicore.net
Website: www.silicore.net