WP142 (v1.0) May 8, 2001

UART to PCI Bridging for Bluetooth Applications

Author: Mamoon Hamid

Introduction

A Xilinx UART (Universal Asynchronous Receiver and Transmitter) to PCI (Peripheral Component Interconnect bus) bridging solution is ideal to integrate the emerging Bluetooth communications standard into legacy systems. It leverages a wide variety of low-cost Bluetooth components, and can be quickly implemented and optimized to ensure minimal impact on host system performance. The result is a fast-to-market, cost effective solution that preserves host MIPS for the primary device application.

With on-chip programmable logic, Xilinx designs benefit from unprecedented flexibility and reprogrammability FPGA- and CPLD-based systems are best for handling the inevitable unknowns encountered during a design flow from the benchtop phase through to product deployment in the field. Further, Xilinx programmable solutions can be quickly and efficiently leveraged into derivative configurations to address new market opportunities. In short, the benefits of Xilinx programmable logic provide unprecedented flexibility and an increased return on engineering investment.

Bluetooth to HCI Bridging

Bluetooth is one of the fastest growing wireless technologies in the marketplace. By offering an alternative to cabled systems that currently connect most electronic devices, it promises a new level of customer convenience and demand. More than 2,000 companies currently participate in the Bluetooth Special Interest Group, and Bluetooth product shipments are expected to approach one billion units per year by the year 2005.

The Bluetooth specification defines a device communications interface that requires minimal user intervention. It supports wireless peer-to-peer connections as well as wireless access to LANs, PSTNs, mobile phone networks, and the Internet. At present, however, most standard Bluetooth silicon solutions simply provide a UART and USB host controller interface (HCI).

Figure 1 illustrates a standard interconnection between a host system and a Bluetooth module.

The traditional Bluetooth module consists of the following three components:

- RF - Radio frequency component
- BB - Baseband processor
- µC - Microcontroller

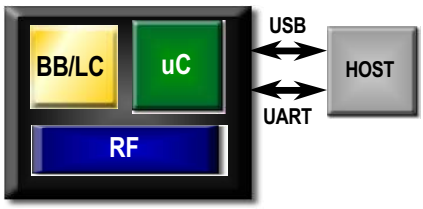

Figure 1: **Zero-Glue Bluetooth Interface**

In many cases, the traditional approach may not be practicable, for instance:

- The host system may not have an available USB or UART port
- The host system may not have a UART port with adequate performance
- The host system is MIPS limited and implementing these port interfaces in the typical manner imposes excessive processing overhead
- The target application is intended to integrate additional capabilities not supported in the standard components

In any of these situations, a Xilinx Spartan FPGA can be used to implement an alternative interface. In this paper, we focus on how you can efficiently bridge from the UART to PCI to Bluetooth enable a legacy system with a PCI bus (which often exhibit the constraints listed above).

While Spartan FPGAs can implement either a USB or UART port, the simpler transport protocol of the UART results in a more cost-effective solution with better system level performance. Since a UART operates in a single mode with eight bits of data, no parity, and one stop bit, the FPGA implementation can be very simple and operate at very high speed. In addition, unlike USB or RS232, the UART transport layer does not need external level shifters or transceivers when implemented in an FPGA. Finally, reduced implementation complexity and better system level performance can be achieved by tailoring the solution specifically to the characteristics of the target components.

UART

A Universal Asynchronous Receiver and Transmitter (UART) is used for communication with serial input/output devices. Serial communication is needed either for devices such as modems and telephone lines, which are inherently serial, or when the cabling cost has to be reduced at the expense of operating speed (e.g., a twisted pair in laboratory instrumentation).

Typically, the UART is connected between a central processor and a serial device. To the processor, the UART appears as an 8-bit parallel port, which can be written to or read from. To the serial device, the UART presents two data wires, one for input and one for output, which serially communicate 8-bit data. The rate of data communication depends on the peripheral device. Some devices operate at a single clock speed (e.g., old modem at 9600 baud) and generate an internal clock. Other devices operate at multiple clock rates and get their clock input from the UART.

A UART is used for building serial communication devices such as modems and serial ports. It has a transmitter section and a receiver section. The transmitter converts the (8-bit) bytes into a serial stream of data bits as they are prepared for transmission. The receiver takes the incoming stream of bits and groups them into 8-bit chunks so they can be reconstructed as bytes.

The UART also monitors input control lines and has the ability to change the state of output lines depending on the program code running at the time. UARTs can be wired as either Data Terminal Equipment (DTE) or Data Communication Equipment (DCE). They are controlled by a clock usually running at different speeds. A buffer is also used to temporarily hold incoming data. This buffer varies by design and is usually very small, ranging anywhere from 1 byte to 128 bytes.

For example, a 16-byte FIFO may not sound like much, but it allows up to 16 characters to be received before the host has to service the interrupt. This increases the maximum bps rate the host can process reliably from 9600 to 153,000 bps, if it has a 1 ms interrupt dead time. A 32-byte FIFO increases the maximum rate to over 300,000 bps.

The two different types of RAM that can be implemented on a Xilinx device are Distributed SelectRAM and Block SelectRAM. Distributed SelectRAM is implemented in LUTs (look-up tables) and is suitable for shallow memory structures and small FIFOs. Block SelectRAM are dedicated 4K-bit RAM blocks, in which depths and widths are parameterizable. These are suitable for applications that require larger blocks of memory. The memory resources available on even the smallest Spartan-II are more than sufficient to build highly efficient fast UARTs.

As illustrated in Figure 2, a typical UART architecture consists of the following blocks:

- Transmit
- Receive
- Control Logic
- Baud Rate Generator
- Internal Control
- Modem Control

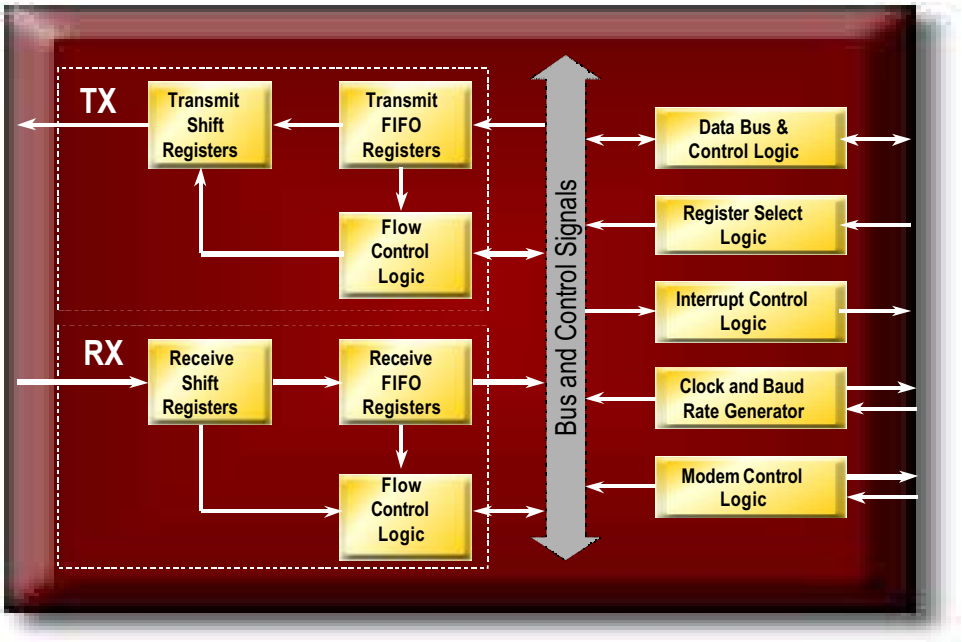

Figure 2: **Standard UART Functional Blocks**

Transmit Block

The Transmit block includes the Transmitter Holding Register (THR), Transmitter Shift Register (TSR), Transmitter FIFO Register (TFR),and Transmitter Flow Control Logic. Whenever the CPU writes parallel data into the Transmitter Holding Register or the Transmitter FIFO, it is immediately transferred to the Transmitter Shift Register. Data is shifted out serially from the TSR.

Receive Block

The Receive block includes the Receiver Buffer Register (RBR), Receiver Shift Register (RSR), Receiver FIFO, and Receiver Flow Control Logic. Whenever the Receiver Shift Register has received a complete character, it is immediately transferred to the Receiver Buffer Register (in character mode) or to the receiver FIFO (in FIFO mode). The CPU reads parallel data either from the RBR or from the RFR depending upon the mode of operation.

Control Logic Block

The Control Logic block consists of two sub-blocks — **Select** and **Control** logic blocks. They decode the address lines and chip select lines and generate enables to various UART internal registers. The Register Control block includes two control registers — the line control register and the FIFO control register. The bits set in these two control registers control the operation of the UART.

Internal Registers

A typical UART device provides internal registers for monitoring and control. These registers function as data holding registers (THR/RHR), interrupt status and control registers (IER/ISR), FIFO control registers (FCR), line status and control registers (LSR/LCR), modem status and control registers (MSR/MCR), programmable data rate (clock) control registers (DLL/DLM), and as a user assessable scratchpad register (SPR).

- **FIFO Control Register (FCR)** is used to enable the FIFOs, clear the FIFOs, set the transmit/receive FIFO trigger levels, and select the DMA mode.
- **Interrupt Status Register (ISR) —** A UART device should provide six levels prioritized interrupts to minimize external software interaction. The Interrupt Status Register (ISR)

provides the user with six interrupt status bits. Performing a read cycle on the ISR will provide the user with the highest pending interrupt level to be serviced. No other interrupts are acknowledged until the pending interrupt is serviced. Whenever the interrupt status register is read, the interrupt status is cleared.

- **Line Control Register (LCR)** is used to specify the asynchronous data communication format. The word length, the number of stop bits, and the parity are selected by writing the appropriate bits in this register.
- **Line Status Register (LSR)** provides the status of data transfers between the UART device and the CPU
- **Modem Control Register (MCR)** controls the interface with the modem or a peripheral device.
- **Modem Status Register (MSR)** provides the current state of the control interface signals from the modem or other peripheral device that the UART device is connected to.

Modem Control Block

The Modem Control block includes the modem control register and the modem status register. It monitors changes in the modem input signals and sets corresponding bits in the modem status register. The CPU sets bits in the modem control register whenever it wishes to transmit data to another terminal or handshake to the modem input signals.

Interrupt Control Block

The Interrupt Control block includes the interrupt enable register (which masks the interrupts from the receiver ready, transmitter empty, line status and modem status registers) and the interrupt identification register. The function of this block is to generate an interrupt whenever:

- Data is received
- Error in the received character
- Character timeout occurs
- Transmitter holding register becomes empty
- Change in the modem input signals.

Programmable Baud Rate Generator

A single baud rate generator is provided for the transmitter and the receiver, allowing independent TX/RX channel control. The programmable Baud Rate Generator is capable of accepting an input clock up to 24 MHz, as required for supporting a 1.5 Mbps data rate.

The generator divides the input 16X clock by any divisor from 1 to 16. A UART device divides the basic crystal or external clock by 16. Further division of this 16X clock provides two table rates to support low and high data rate applications using the same system design. Customized (using an FPGA) baud rates can be achieved by selecting the proper divisor values for the MSB and LSB sections of baud rate generator. Programming the Baud Rate Generator Registers DLM (MSB) and DLL (LSB) gives the user the capability for selecting the desired final baud rate.

PCI Interface

The PCI (Peripheral Component Interconnect) Local Bus is one of the most successful standards in history, serving as the main general-purpose bus in virtually every desktop computer and an ever growing number of embedded systems throughout the world. A great deal of this success can be attributed to PCI's original forward-thinking design. The flexibility of PCI has allowed it keep pace with tremendous increases in CPU performance and data capacity, while making it suitable for laptops, desktops, servers, and embedded applications.

PCI's standardized components and silicon have also enabled huge economies of scale that make PCI products easy and inexpensive to develop. As a result, PCI has become the universal connection standard for all types of low-cost subsystems and peripherals, and has

given PCs an affordable graphics capability that was unachievable with previous bus technologies.

PCI has become one of the most popular bus standards, not only for personal computers, but also for industrial computers, communication switches, routers, and instrumentation. It solves a wide range of compatibility problems and performance limitations that were encountered with the older ISA and VME standards.

However, PCI is a significant design challenge; the stringent electrical, functional, and timing specifications are difficult to meet in any technology-and the standard continues to evolve to meet the dynamic needs of the industry.

One needs a flexible PCI solution that will meet both current and future requirements, while guaranteeing full PCI compliance with no limitations on performance or functionality. Xilinx released its first PCI product in January, 1996 and since have been proven in over 1000 customer designs, demonstrating its flexibility and a cost-effective solution for a fully compliant, high-performance PCI system.

The Xilinx LogiCORE PCI32 and LogiCORE PCI64 are 5V and 3.3V fully compliant 32-bit and 64-bit PCI interfaces for up to 66 MHz designs. Together, with Spartan-II, they provide a flexible PCI solution, that is field upgradeable and able to meet both current and future requirements for the lowest possible cost. This allows replacement of many ASSPs, such as a UART to PCI Bridge. A typical PCI design requires additional FPGAs or CPLDs for value added logic devices, all of this can be integrated with the PCI interface in a single Xilinx Spartan-II FPGA. Figure 3 illustrates a 32-bit PCI interface as Xilinx provides it. To the left hand side are the standard PCI bus signals and to the right are the user interface signals needed to interface to User Application, which in this case is any variation of a UART.

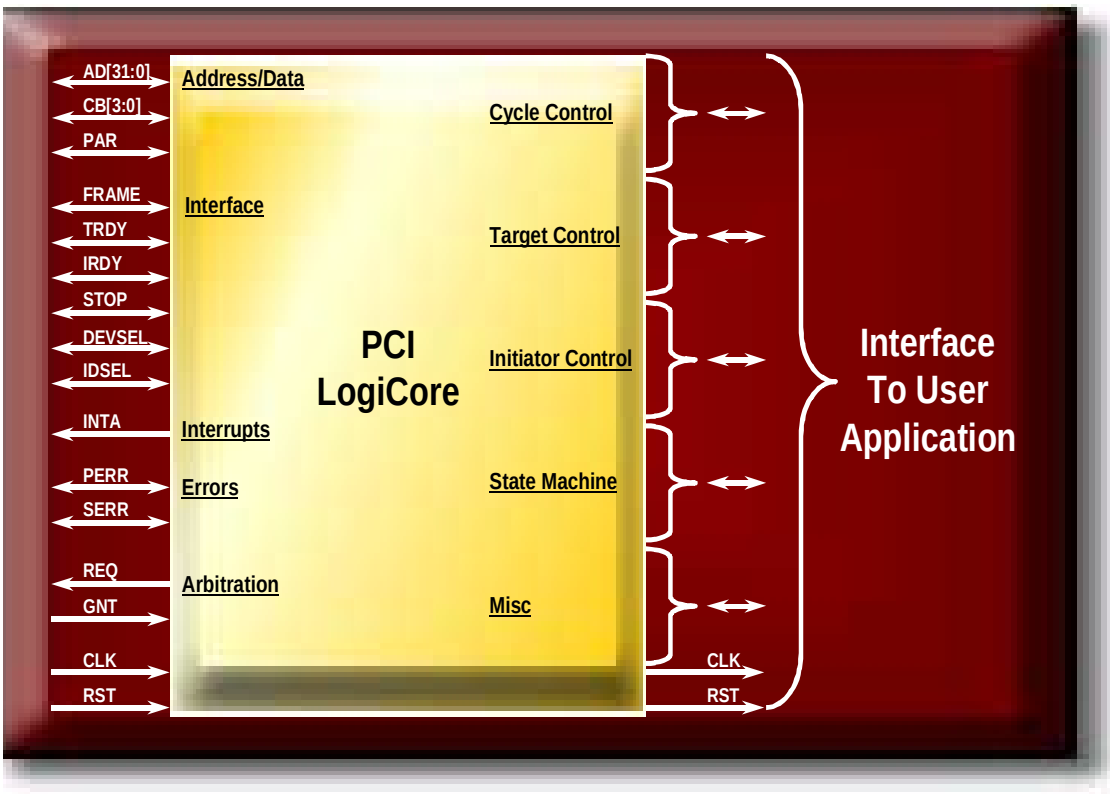

Figure 3: **The PCI LogiCore Block**

The LogiCORE PCI32 Master and Slave Interface is partitioned into five major blocks and a user application as described below. Figure 4 shows the detailed block diagram of the PCI solution.

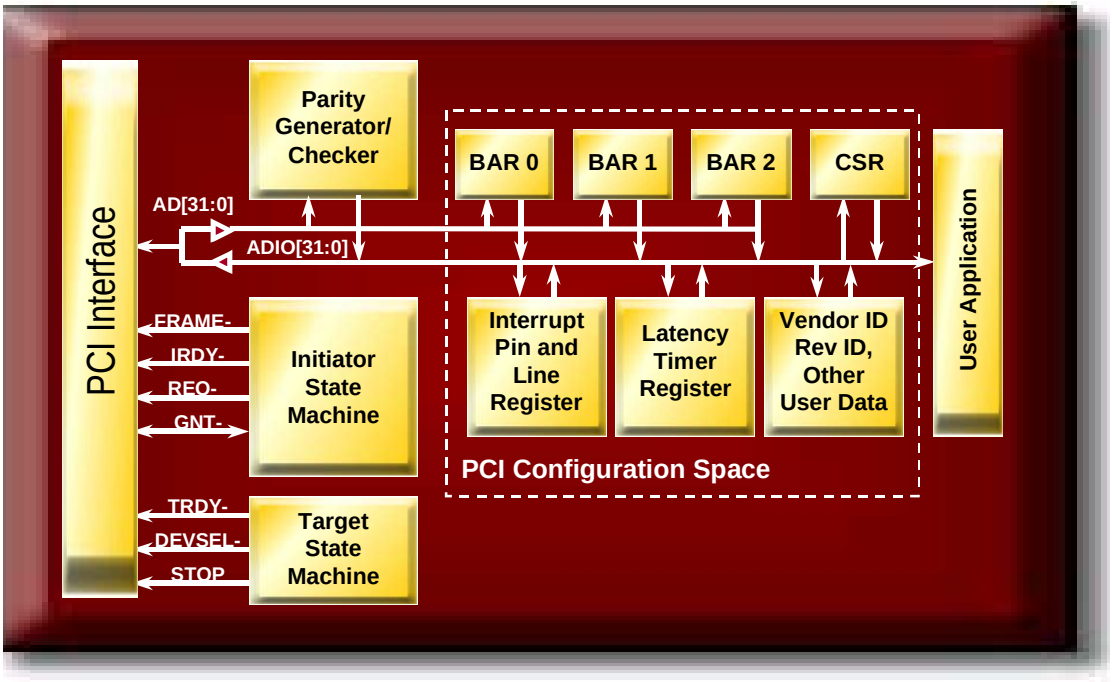

Figure 4: **Breakdown of the PCI LogiCore**

PCI Configuration Space

This block provides the first 64 Bytes of Configuration Space Header (CSH) to support software-driven "Plug-and Play" initialization and configuration. This includes information for Command, Status, and three Base Address Registers (BARs).

Each BAR sets the base address for the interface and allows the system software to determine the addressable range required by the interface. Every BAR designated as a memory space can be made to represent a 32-bit space. Using a combination of Configurable Logic Block (CLB) flip-flops for the read/write registers and CLB look-up tables for the read-only registers results in optimized logic mapping and placement.

PCI I/O Interface Block

The I/O interface block handles the physical connection to the PCI bus including all signaling, input and output synchronization, output three-state controls, and all request-grant handshaking for bus mastering.

Parity Generator/Checker

This block generates/checks even parity across the AD bus, the CBE lines, PAR and the PAR signal. It also reports data parity errors via PERR- and address parity errors via SERR-.

Target State Machine

This block controls the PCI interface for Target functions. The states implemented are a subset of equations defined in "Appendix B" of the PCI Local Bus Specification. The controller is a high-performance state machine using one-hot (state-per-bit) encoding for maximum performance. State-per-bit encoding of the next-state logic functions facilitates a high performance implementation in the Xilinx FPGA architecture.

Initiator State Machine

This block controls the PCI interface for Initiator functions. The states implemented are a subset of equations defined in “Appendix B” of the PCI Local Bus Specification. The Initiator Control Logic also uses state-per-bit encoding for maximum performance.

UART to PCI Bridge

The immediate areas accelerated by using Spartan-II FPGAs are the FIFO Buffers. These FIFOs can be reconfigured in depth and width to accommodate much faster data rates as well as wider words and custom word widths. Spartan-II BlockRAM is ideal for use for deep memory structures whereas distributed SelectRAM is suitable for shallower memory blocks and small FIFOs. Figure 5 shows the functional components of a single-channel UART to PCI Bridge.

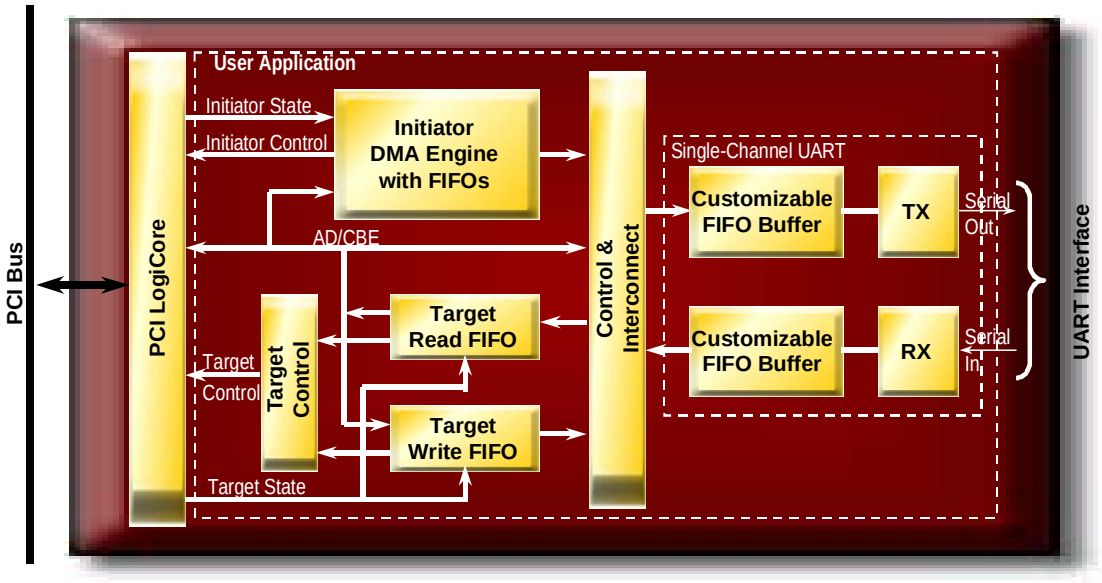

Figure 5: **Single-Channel UART to PCI Bridge**

Shown also is the interface to the PCI LogiCore block. This block on one side provides the standard PCI bus signals and on the back side provides the interface signals necessary to implement a useful user application, which in this case happens to be DMAs and FIFOs to control the data flow to the UART.

The Initiator “DMA” (Direct Memory Access) engine accepts requests from the local side to perform block transfers. It then issues a transaction across the PCI bus to satisfy the local side’s request in as many PCI transactions as required. Similarly, a representation of the interface to the PCI LogiCore and the Target Read and Write FIFOs are shown. For more details on the implementation of Initiator and Target Design, refer to the PCI Implementation Guide (see references).

The critical portion of this design is the control of the data flow from the PCI bus, which can reach optimal throughput of 132 MB/s on 32-bit/33 MHz architecture. Since a single channel UART only performs at a fraction (1%) of this bandwidth, the Control and Interconnect needs to be carefully designed to interface to the UART RX and TX FIFOs to control the data flow.

A variation of *figure 5* is the block diagram shown in Figure 7. To increase the supported Bluetooth connection density it may be desirable to instantiate multiple UARTs each with a separate Bluetooth module. This example illustrates a quad-channel UART variation, each with

a 16 byte FIFO, which could easily support FOUR simultaneous 721 Kbps maximum data rate Bluetooth connections.

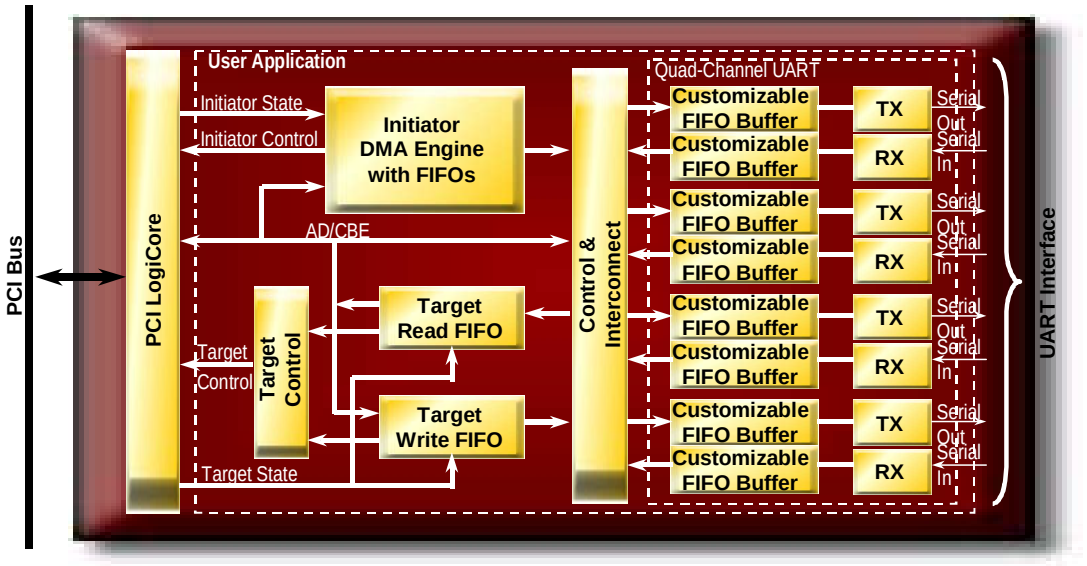

Figure 6: **Quad-Channel UART to PCI Bridge**

Figure 7 illustrates another derivative of Figure 5, in this case with intelligent DMA functionality. With the integration of a DMA engine and Bluetooth HCI frame transfer state machine logic the host system overhead for servicing interrupts can be significantly reduced. Unlike traditional UARTs where an interrupt is generated every time the small on-chip FIFOs are filled or emptied, this design generates an interrupt only when a complete HCI frame has been transmitted or received. This can be accomplished with application specific logic that decodes the frame size information in the HCI header and configures and manages the DMA appropriately. This logic can also ensure that proper frame level synchronization is being maintained. The net result is considerably reduced overhead for the host processor preserving more MIPS for its primary value-add applications.

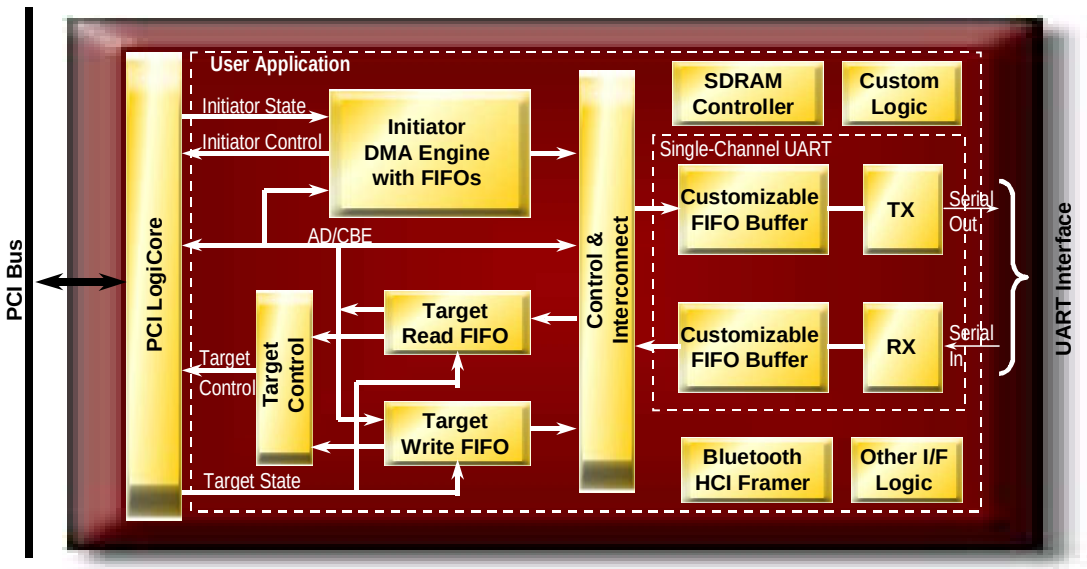

Figure 7: **Single-Channel UART to PCI Bridge with Added Functionality**

With the wide range of standard interface and memory controller IP available for Spartan FPGAs similar derivative designs can be constructed to implement almost any other core logic

function as well. For instance, *Figure 7* also illustrates how you can integrate additional value adding features such as an SDRAM controller and custom logic. Such versatility builds staying power and punch into your engineering investment, positioning you to quickly and efficiently evolve your design to address emerging market opportunities before your competition.

The Benefits of Programmable Logic

As we have seen, programmable logic provides an excellent platform for integrating Bluetooth technology into embedded systems. Let's highlight the key benefits that they bring:

Time-to-Market

Xilinx programmable logic provides several advantages that reduces time-to-market. First, a broad range of IP support from Xilinx and a ever growing number of third party IP partners provide quick access to key design building blocks. Second, as benchtop programmable solutions, they allow the system designer to achieve a functional hardware platform more rapidly than any alternative. And third, programmable devices are standard parts that are easy to reproduce quickly in limited-to-high volumes to capture a position in strategic accounts before the competition.

Rapid Software Development

Software development is one of the biggest issues in integrating Bluetooth technology. And, obviously, since programmable logic can achieve functional hardware sooner it creates an advantage in this area. However, this advantage can be even greater when you consider the flexibility that programmability brings to the equation. For instance, it is often desirable to use existing or third party drivers and firmware. With a programmable solution, you have full control over the behavior of the interface ensuring a workable approach. This can be particularly valuable when third party code is involved because it may not be well documented or understood and modifications can raise support and maintenance concerns.

Time-in-Market

Product development by its nature is not an exact science. Bugs and incompatibilities are simply a reality that engineering must deal with. Here, especially Xilinx programmable devices can provide a valuable advantage, as our solutions are inherently **reprogrammable**. Thus, patches for known problems can be put into production as soon as they are validated on the existing hardware revision and can also be deployed to installed systems. This allows you to keep your existing design shipping and greatly reduces the risk of obsolete part inventories and expensive field replacement programs.

Rapid Design Derivation

A system design is a corporate asset and in today's world of hyper competition and compressed development cycles, these assets must be flexible. Standards evolve, customers request new features, and experience reveals new business opportunities that can be exploited. Thoughtful designs that incorporate programmable logic are inherently more scalable and are superior platforms for rapid and efficient product derivation. Thus, well exploited programmable logic can make your future product roadmap a strategic competitive advantage.

System Level Cost Reduction

In the past, the use of programmable logic was considered an expensive solution. However, times have changed because Moore's Law has worked to the advantage of programmable solutions. Today, $10 will buy 100,000 system gates in volume, off the shelf, and ready to go. And, as these devices usually replace other functions in your design as well, they can often enable real system level cost reductions. Programmable logic has replaced the small cost reduction ASICs of yesterday and brings many other advantages to your system too!

References

1. *Bluetooth HCI Bridging*, White Paper, January 2001, Kent Dahlgren
2. *UART to PCMCIA*, White Paper, March 2001, Antolin Agatep
3. U*ART to 1394*, White Paper, March 2001, Saeid Mousavi
4. *200 MHz UART Design* by Ken Chapman, **http://support.xilinx.com/xapp/xapp223.pdf**
5. PCI SIG, pcisig.org
6. The PCI Design Guide, **http://www.xilinx.com/products/logicore/pci/docs/design_guide_30.pdf**
7. *Specification of the Bluetooth System*, Core, version 1.0B December 1999, Bluetooth SIG
8. *Bluetooth PC Card Transport Layer, White Paper*, version 1.0 August 1999, Bluetooth SIG

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
05/08/01	1.0	Initial Xilinx release.