WP145 (v1.0) May 8, 2001

UART to Powerline Bridging for Bluetooth

Author: Saeid Mousavi

Introduction

Distribution of video, audio, and PC data has employed different interconnection technologies in networking. Depending on the bandwidth, quality of the content, cost, performance, regulations, and geographical locations different technologies can be used. Ethernet, IEEE 1394, Bluetooth, USB, HomePNA, HomePlug, HomeRF, HiperLAN2, and Wireless LAN are among the enabling technologies to network homes and/or small offices. Each one of these technologies has its own pros and cons and is suitable for specific application(s).

Bluetooth is one of the fastest growing wireless technologies in the market. Intended to replace the cables connecting almost every electronic device, it promises great convenience. With over 2,000 companies involved in the Bluetooth Special Interest Group, Bluetooth is now emerging in countless devices with shipments expected to approach 1 billion units per year by 2005.Bluetooth is intended to replace the cable connecting portable and/or fixed electronic devices.

In networking, technology bridges are being used for protocol translations. The existence of different home networking protocols and standards makes FPGA devices (Spartan™-II) an ideal candidate for bridging functionality. Bridging between UART and powerline-based technologies can connect Bluetooth enabled devices to a powerline network.

Powerline home networking technology allows consumers to use pre-existing electrical wiring system to connect home appliances to each other and to the Internet. Home networks that utilize high-speed power line technology can control anything that plugs into an outlet. The existing global powerline infrastructure makes it a cost effective and readily available medium for home networking. New powerline products are targeting data rates of 12 Mbps with solutions addressing data attenuation, security, and noise issues.

With Xilinx programmable logic, on-chip designs will also benefit from Bluetooth's unprecedented flexibility and reprogrammability. FPGA and CPLD based systems can better deal with the inevitable unknowns designers will encounter from the benchtop in the lab to deployed products in the field. Further, programmable solutions can be quickly and efficiently leveraged into derivative configurations to rapidly address new market opportunities. When fully exploited, these benefits will reap large rewards and greatly improve the return on your 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

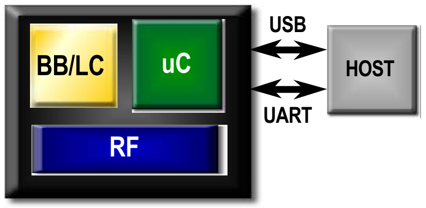

Figure 1: **Zero-Glue Bluetooth Interface**

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 9,600 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 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

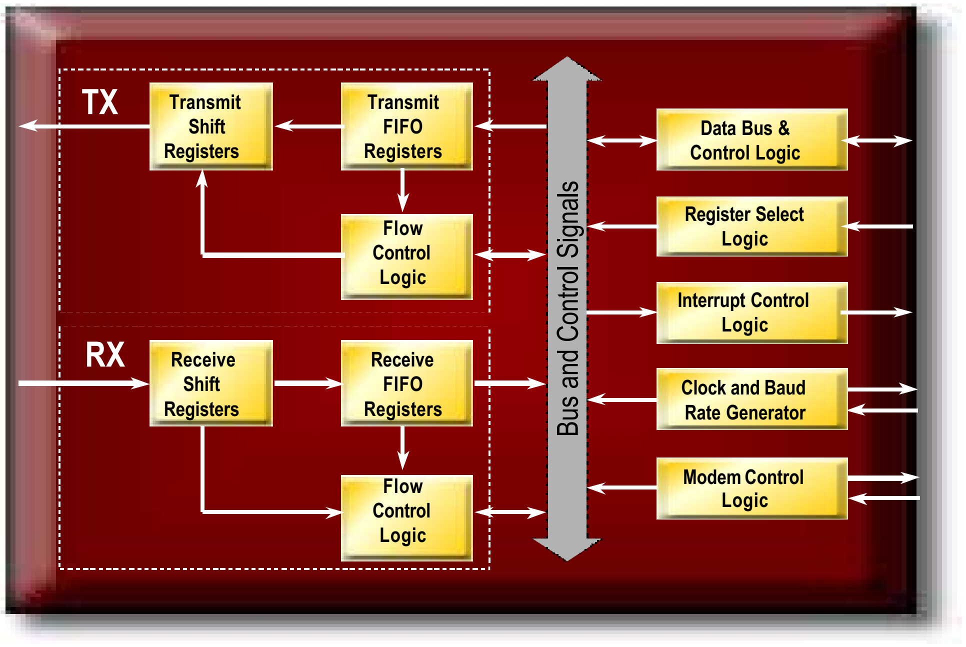

Figure 2: **Standard UART Functional Blocks**

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 chips 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.

Powerline Transceiver

A powerline transceiver allows devices to communicate using ordinary powerlines as the transmission media. Transmitter and Receiver are the main functional blocks of a powerline transceiver.

Transmitter

The processor assembles data to be transmitted in the buffer RAM and commands the transmission to begin. The transmitter hardware handles all tasks of transmission until they are completed.

Each byte of data is converted from parallel to serial form and encoded into differential phase shift keying (PSK). Two types of modulation are usually supported for powerline-based modulation; binary phase-shift keying (BPSK), or quadrature phase-shift keying (QPSK).

The transmitter is responsible for sending the preamble, encrypting the, and generating a 16-bit CRC. The CRC is appended to and sent at the end of the packet. The transmitter interrupts the processor with a transmit packet interrupt after packets are sent.

Receiver

The receiver data are driven to the powerline transceiver as digital input signals, one for each carrier used on the powerline. The processor is not involved with the actual reception of packets beyond enabling them to begin. A valid packet consists of a valid preamble and a destination address to which this node responds. Each byte of data is decrypted as it is received. The packet’s data is also checked against its CRC. Each data byte is stored in the internal buffer RAM for later access by the Processor.

After a valid packet is received, a received packet interrupt is generated. After a successful reception, the hardware stops hunting for new receive packets. The receiver gives status about each channel that is guaranteed valid after a received packet interrupt is generated. The status stays valid until the receiver is again enabled to receive packets. Status shows that address match occurred, which errors occurred in the reception of the packet, and at what symbol time the packet was received. In the past the use of programmable logic was considered an expensive solution. However, times have changed!

UART-to-Powerline Bridge

Different networking and interconnection technologies must co-exist with each due to different performance requirements, distribution methods, and cost considerations. A technology bridge successfully connects these disparate technologies together. Re-programmability features of an FPGA provide excellent flexibility and reliability for bridging and translating these technologies.

The Spartan™-II family of FPGAs provide a flexible and programmable architecture of Configurable Logic Blocks (CLBs) surrounded by a perimeter of programmable Input/Output Blocks (IOBs). There are four Delay-Locked Loops (DLLs) for clock management (can be used for customized UART Baud Rate Generator). Spartan-II has two columns of block RAM between the CLBs and the IOB columns. A powerful hierarchy of versatile routing channels interconnects these functional elements.

The Spartan-II family FPGAs are customized by loading configuration data into internal static memory cells. Unlimited reprogramming cycles are possible with this approach. Stored values in these cells determine logic functions and interconnections implemented in the FPGA.

Configuration data can be read from an external serial PROM (master serial mode), or written into the FPGA in slave serial, slave parallel, or boundary scan modes. These unique features enable designers to develop programmable Application Specific Standard Products (ASSP) in a very short time cycle.

The Spartan-II family devices provide system clock rates up to 200 MHz and internal performance as high as 333 MHz. The Spartan-II family FPGAs offer the most cost-effective solution while maintaining leading edge performance. In addition to the conventional benefits of high-volume programmable logic solutions, Spartan-II FPGAs also offer on-chip synchronous single-port and dual-port RAM (block and distributed form), DLL clock drivers, programmable set and reset on all flip-flops, fast carry logic, and many other features.

A Buffer Manager, along with an adjustable Transmit/Receive Shift Registers, and Transmit/Receive FIFO Registers of a UART can act as an actual bridge between UART and Powerlines packages. This buffer manager should have a RAM bandwidth of 100 Mbps. It provides storage for Powerlines and UART data, automatically storing them and passing them to the appropriate destinations, without any intervention from the processor.

The Spartan-II FPGAs dedicated on-chip blocks of 4096-bit dual-port synchronous RAM are ideal for use in FIFO applications. This feature, along with other features, is ideal for building a UART-to-Powerlines bridge. The FIFOs can be reconfigured in depth and width to accommodate much faster speeds in transmission and reception data rates. This gives designer a great amount of flexibility to customize their UART-to-Powerlines bridge based on different applications and system performance requirements.

Other advantages of using Spartan-II in UART-to-Powerlines Bridge include:

- Faster Transmit and Receive Shift Registers
- Wider Transmit and Receive Shift Registers
- DMA to allow more efficient bus traffic
- Removable internal baud rate generator
- Customizable CPU interface
- Multiple transmit and receive blocks for independent additional fast serial channels

Spartan II FPGA has up to 200K gates which make the multiple implementations of UART possible. DMA channels can also be implemented to increase the performance and functionality of the bridge. Figure 3 illustrates the implementation of Powerlines MAC/PHY and UART in Spartan-II.

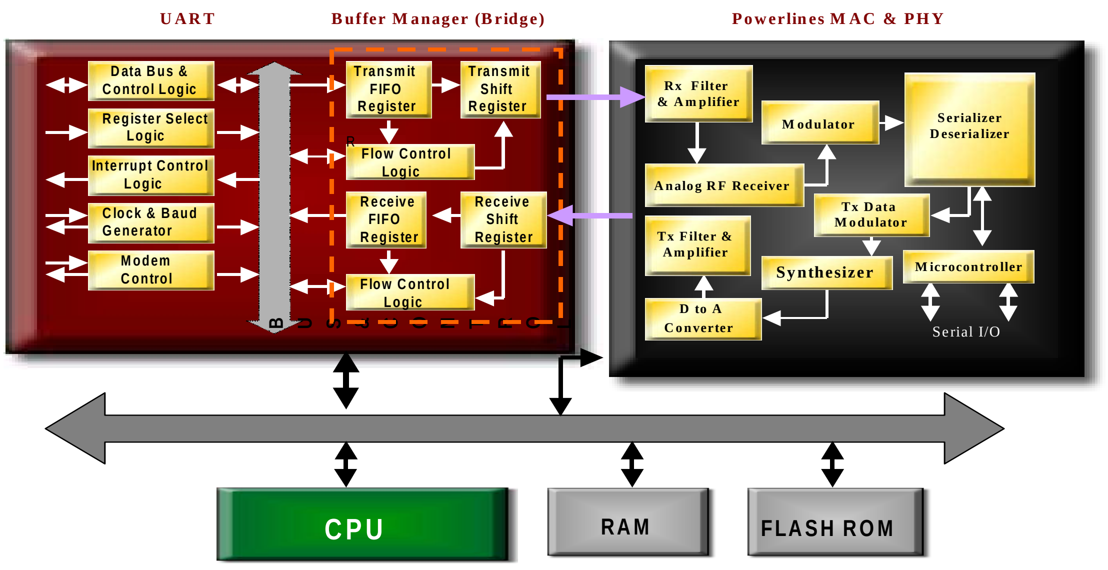

Figure 3: **UART-to-Powerline Bridge**

FPGA Advantages

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 rproduce 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. UART to PCI White Paper, March 2001, Mamoon Hamid

4. UART to 1394 White Paper, March 2001, Saeid Mousavi
5. 200 MHz UART design by Ken Chapman, http://support.xilinx.com/xapp/xapp223.pdf
6. Specification of the Bluetooth System, Core, version 1.0B December 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.