WP141 (v1.0) April. 27, 2001

UART to PCMCIA Bridging for Bluetooth

Author: Antolin Agatep

Introduction

A Xilinx based fast UART to PC Card (PCMCIA) bridging solution is the ideal mechanism for integrating industry standard Bluetooth communications into legacy systems. Such a solution can be realized quickly, can leverage a wide variety of low cost Bluetooth components, and can be optimized to impose a minimal impact on the host system implementation and performance. The result is a cost effective solution, fast time-to-market, and preservation of host MIPS for the primary device application.

With Xilinx programmable logic on board your design will also benefit from unprecedented flexibility and re-programmability. FPGA and CPLD based systems can better deal with the inevitable unknowns you will encounter from the benchtop in your 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 can reap large rewards and greatly improve the return on your engineering investment.

Bluetooth HCI Bridging

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 2000 companies involved in the Bluetooth Special Interest Group it is now emerging in countless devices with shipments expected to approach 1 billion units per year by 2005.

The Bluetooth specification defines a uniform structure for devices to communicate with minimal user intervention. The technology can support peer-to-peer connections as well as wireless access to LANs, PSTN, mobile phone networks and the Internet. However, typical Bluetooth silicon solutions only provide a UART and USB host controller interface (HCI).

Figure 1 illustrates the typical 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

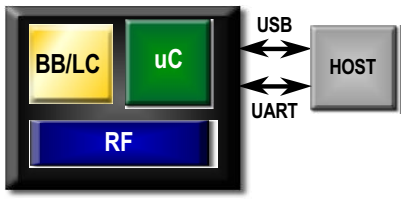

Figure 1: **Zero-Glue Bluetooth Interface**

In many cases, this approach may not be workable for any of the following reasons:

- The host system may not have sufficient USB or UART ports for the application. This can occur when the host has a limited number of ports, or when that application requires a large number of ports.
- The host system may not have serial ports that can support the data rate required for full Bluetooth performance. In order support maximum system level performance, Bluetooth ASSPs include UARTs that are capable of supporting transfer rates of up to 1.5 Mbps.

- Standard port interface ASSPs are not tailored for Bluetooth protocol handling, and as a result they may consume significant processing resources when operating at Bluetooth rates. And, interrupt processing overhead can become a drain on host processing resources.

In any of these situations, a Xilinx Spartan-II FPGA can be used to implement an alternative interface. This paper focuses on how to efficiently bridge from UART to PCMCIA to Bluetooth enable a legacy system with a UART serial peripheral (which often exhibit the constraints listed above).

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 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

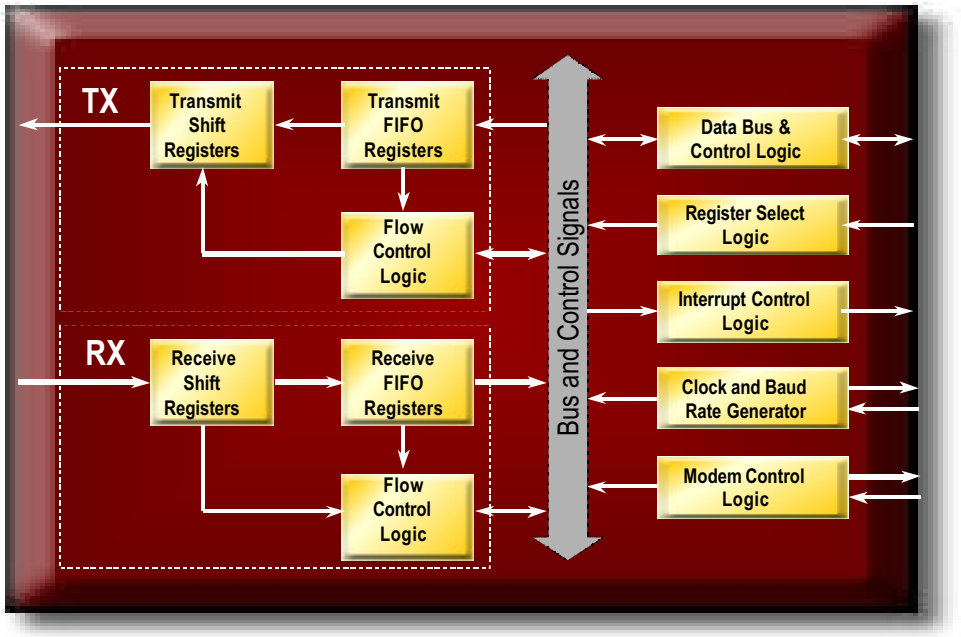

Figure 2: **Standard UART Functional Blocks**

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.

PC Card (PCMCIA) Introduction

In the early 1990s, the rapid growth of mobile computing drove the development of smaller, lighter, and more portable tools for information processing. One of the most exciting of these innovations was PC.

Card Technology

The power and versatility of PC Cards quickly made them standard equipment in mobile computers. The rapid development and worldwide adoption of PC Card technology has been due in large part to the standards efforts of the Personal Computer Memory Card International Association (PCMCIA).

The first release of the standard defined the 68-pin interface and the Type I and Type II PC Card form factors. The initial release of the PCMCIA Standard specified the electrical and physical requirements for memory cards only. It defined the Metaformat or Card Information Structure (CIS) that is critical to interoperability and plug-and-play for PC Cards. There was no concept of input/output (I/O) cards in the first release of the PC Card Standard.

The second release of the standard defined an I/O interface for the same 68-pin interface as was used for the PCMCIA memory cards in the first release of the Standard. Release 2.0 also added various clarifications to the first release, support for dual-voltage memory cards, and sections dealing with card environmental requirements and test methods. Release 2.01 added the PC Card ATA specification, the Type III card type, and the Auto-Indexing Mass Storage (AIMS) specification geared toward digital images was also added. It also included the initial version of the Card Services Specification. Release 2.1 further enhanced the Card and Socket Services Specification and made improvements to the Card Information Structure.

The latest release of the PC Card Standard added information to improve compatibility and added support for features such as 3.3V operation, DMA support, and 32-bit CardBus bus mastering.

Table 1: **PC Card Physical Characteristics**

Physical Interface	68 Pins
Back End I/O Connectors	Proprietary(1)
Length	85.6 mm
Width	54.0 mm
Thickness	Type I = 3.3 mm Type II = 5.0 mm Type III = 10.5 mm
Operating Temperature	0 to 55°C
Storage Temperature	–20 to 65°C
Minimum Insertions	Office Env. 10,000 Harsh Env. 5,000

Notes:

1. Two standardized connectors are available as part of the optional PCMCIA Specific Extensions Specifications

Metaformat (CIS) Specification

The goal of the Metaformat Specification is to allow PC Cards to handle numerous, somewhat incompatible data-recording formats and data organizations. Metaformat is also known as Card Information Structure (CIS). The Metaformat is a hierarchy of layers. Below the Metaformat is the physical layer, the electrical and physical interface characteristics of PC Cards. The Metaformat layers are Basic Compatibility, Data Recording, Data Organization, and System-Specific.

- **The Basic Compatibility Layer** specifies a minimal level of card-data organization. Tuples at this level provide fundamental information about the PC Cards including supported configurations, manufacturer, and individual device characteristics such as size, speed, and programming information. An exampleof a tuple from the Basic Compatibility

Layer is the Function ID Tuple, shown here.

Table 2: **CISTPL_FUNCID: Function Identification Tuple**

Code	Name	Code	Name
0	Multi-Function	7	AIMS
1	Memory	8	SCSI
2	Serial Port	9	Security
3	Parallel Port	A - FD	Reserved
4	Fixed Disk	FE	Vendor Specific
5	Video Adapter	FF	Do Not Use
6	Network Adapter	-	-

- **The Data Recording Layer** includes tuples that describe partitioning information and provide card initialization information.
- **The Data Organization Layer** currently includes a single tuple, CISTPL_ORG, which specifies the partition organization (for example, the file system) in use in a partition described by Data Recording Format Layer tuple(s).
- **The System-Specific Layer** includes the special purpose tuple, CISTPL_SPCL, and the range of vendor-unique tuple codes. The special purpose tuple provides a mechanism for documenting the format and interpretation of special tuple usage within the PC Card Standard.

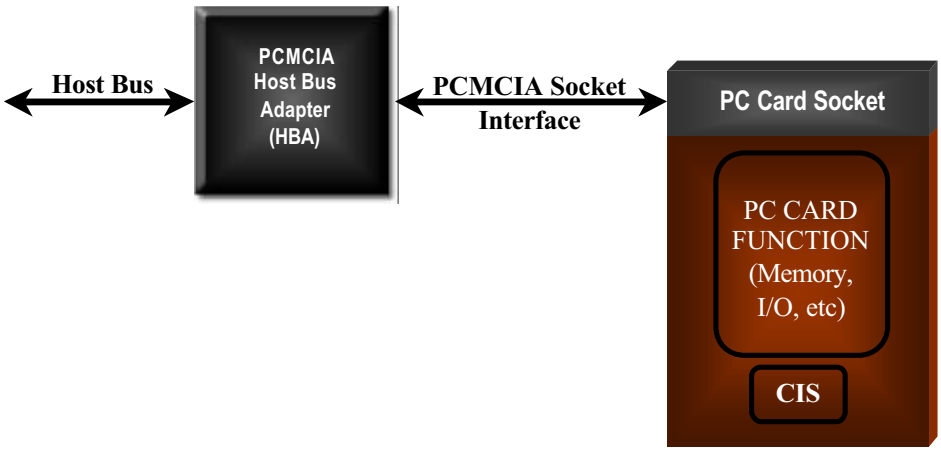

Figure 3: **A Typical 16-bit PC Card Subsystem**

UART to PC Card (PCMCIA) HCI Bridging

Figure 4 provides an overview of the HCI bridge. The Spartan-II device takes in Bluetooth frames from the UART at one end and communicates this to the PC Card (PCMCIA) interface in the other end. And, in the opposite direction, the Spartan-II device takes in information from the PC Card (PCMCIA) bus and sends commands and or data to the UART connected to a Bluetooth module.

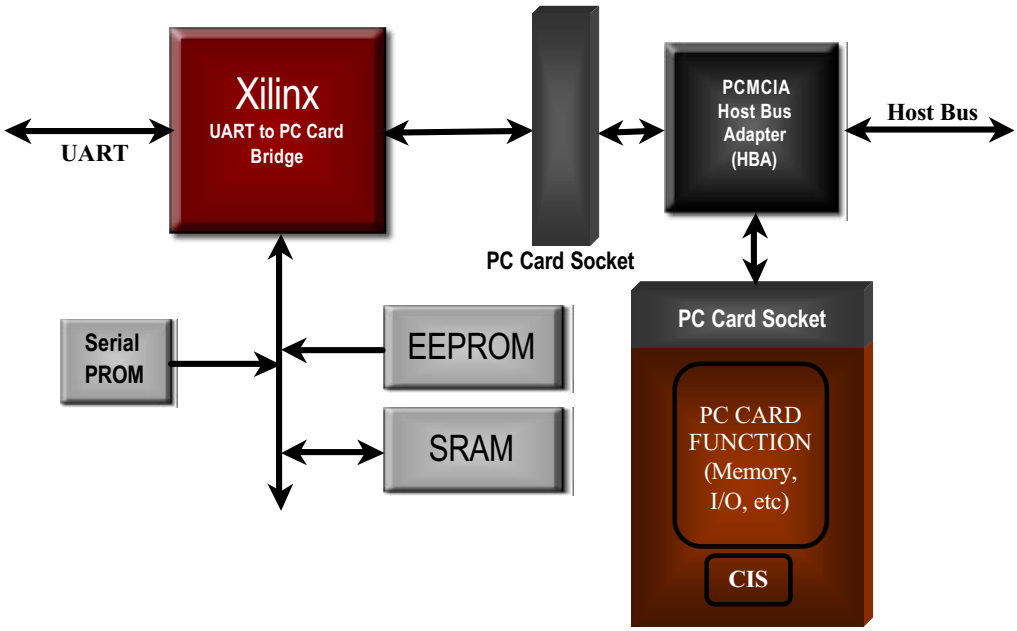

Figure 4: **Block Diagram of UART to PC Card (PCMCIA) Bridge**

Several of the immediate areas that are accelerated by using Spartan-II devices are FIFOs and shift registers. The FIFOs could be reconfigured in depth and width to accommodate much faster data rates as well as wider words and custom word widths. The shift registers can also be made wider to enable them to deal with different word widths.

Figure 5 shows the functional components of the standard PCMCIA to UART bridge while Figure 6 shows an identical block diagram of the bridge with the DMA and Bluetooth HCI frame transfer state machine logic. This approach improves system level performance by reducing the overhead of servicing interrupts for transmission and reception. 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 is accomplished by having application specific logic that decodes the frame size information in the HCI header and configures the DMA logic appropriately. This logic also checks to ensure that proper frame level synchronization is being maintained.

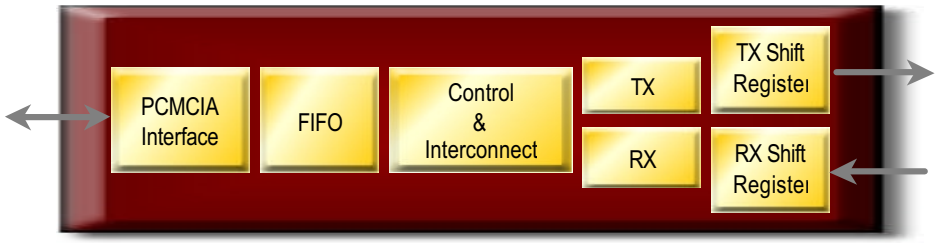

Figure 5: **Functional Blocks of Standard PCMCIA to UART Bridging Using Spartan-II**

The net result is the burden of interrupt handling is considerably reduced for the host processor. As a result more processor performance is available for other value added functions. In

addition, with the wide range of standard interface and memory controller IP available, the Spartan device can be used to implement other core logic functions as well.

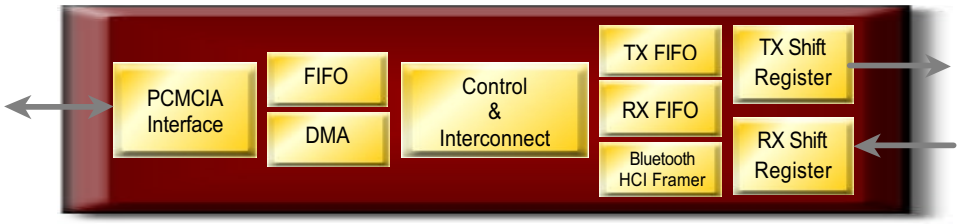

Figure 6: **Functional Blocks of High Speed PCMCIA to UART Bridge**

Sustainability

Complementing the previously described solution are the following technologies:

- **IRL** — enables the remote upgrade and servicing of designs incorporating Xilinx FPGAs,
- **Select I/O** — allows the system designer to effectively target all existing industry I/O standards from PCI to AGP to LVDS to etc, and
- **Reconfigurability** — Xilinx FPGAs are truly, dynamically and partially reconfigurable, enabling the products they're designed in to have much longer time in market.

Derivatives

Extending the design of the Figure 6 bridge to support multiple UART interfaces, one would get the system shown below by Figure 7. This extended architecture will allow several simultaneous Bluetooth HCI bridging which in turn effectively increases the bandwidth and frame transactions per unit time.

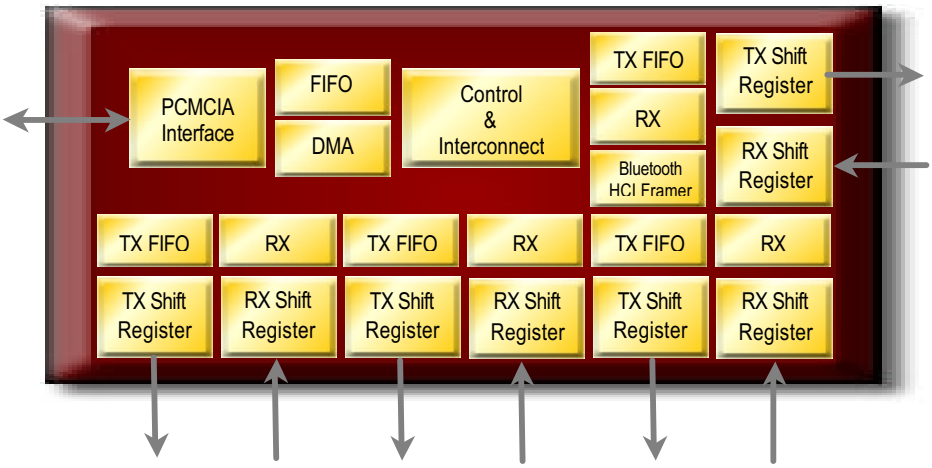

Figure 7: **Functional Blocks of PC Card (PCMCIA) to Multiple UARTs Bridge**

Another advantage that is not apparent with the Figure 7 system is that additional fast UARTs can be created to address communications with multiple serial devices.

Altogether, Figure 7 can be extended again to meet the 33 MHz, 32-bit, Hot-Pluggable, PCI compliant CardBus architecture for applications that require much higher system performance.

Xilinx System Strengths

To summarize, the following immediate benefits can be achieved by designing with Xilinx Spartan-II FPGAs:

- Deeper FIFOs for support of higher speeds in transmission and reception
- Wider FIFOs to accommodate various bus and data widths
- Faster Transmit and Receive Shift Registers
- Wider Transmit and Receive Shift Registers
- DMA to allow more efficient bus traffic
- Multiple transmit and receive blocks for independent additional fast serial channels
- All of the above implemented in a fully reconfigurable Spartan-II fabric
- Spartan-II FPGAs are all PCI compliant devices that provide a path to the Hot-Pluggable, high-performance, 32-bit, 33 MHz CardBus standard

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 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!

Conclusion

An investment in engineering is the same as any financial investment. Every time a company decides to develop a product, it is taking risks that the effort will pay off with big rewards. Often times the risks are overly simplified and don't become apparent until later when modifications are impossible or prohibitively expensive.

The present markets for emerging technologies are accelerating towards convergence, which exacerbates the situation. More and more products compete to get to market first with the newest features and capabilities and there just isn't enough time to properly debug design flaws and/or perhaps add features that weren't considered but that suddenly become crucial to a product's success. Xilinx Spartan-II FPGAs time and again have proven themselves to be the ultimate system building blocks that enable future proofing of emerging and vertical market applications which directly translate to profits and market leadership for those who employ them.

References

1. *Bluetooth HCI Bridging.* White Paper, January 2001, Kent Dahlgren
2. *UART to 1394,* White Paper, March 2001, Saeid Mousavi
3. 200 MHz UART design by Ken Chapman, **http://support.xilinx.com/xapp/xapp223.pdf**
4. *Specification of the Bluetooth System*, Core, version 1.0B December 1999, Bluetooth SIG
5. *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
04/27/01	1.0	Initial Xilinx release.