Composable System Resources as an Architecture for Networked Systems

by

Sandeep Chatterjee

S.M. (E.E.C.S.) Massachusetts Institute of Technology (June 1997)
B.S. (E.E.C.S.) University of California at Berkeley (June 1995)

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January 2001

Author ....
Department of Electrical Engineering and Computer Science
January 30, 2001

Certified By ....
Srinivas Devadas
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted By ....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Composable System Resources as an Architecture for Networked Systems

by

Sandeep Chatterjee

Submitted To The Department Of Electrical Engineering And Computer Science
On January 30, 2001, In Partial Fulfillment Of The
Requirements For The Degree Of
Doctor Of Philosophy In Electrical Engineering And Computer Science

Abstract

Network devices promise to provide a variety of user interfaces through which users can interact with network applications. The design of these devices stand in stark contrast to the design of personal computers in which new software content is accommodated by increased processor performance. Network device design, on the other hand, must take into consideration a variety of metrics including interactive performance, power consumption, battery life, transaction security, physical size and weight, and cost. Designing a general-purpose platform that caters to all of these metrics for all applications and devices is impractical. For an application mix, a processor architecture and platform can be designed that is optimized for a selected set of metrics, such as power consumption and battery life. Each of these optimized processor architectures and platforms will no doubt be applicable to a variety of devices.

This suggests a modular system architecture for network devices that segments the computational resources from the device UI. Computational resources can be selected for a device UI that are optimized with respect to application mixes as well as to user preferences and metrics. Segmenting out the device UI reduces the complexity of device UIs, simplifying development and lowering costs. At the same time, with little electrical circuitry resident on device UIs, the selected platform can more fully optimize the entire device.

In this thesis, I describe an architecture for network devices that is based on using pluggable system resource modules that can be composed together to create a close-to-optimal platform for a particular application mix and device. Frequently used applications execute efficiently, while infrequently used applications execute less efficiently. Metrics for calculating efficiencies and selected application domains and mixes are specified by individuals as opposed to one-size-fits-all metrics specified by manufacturers. I show that such a composable system architecture is effective in optimizing system performance with respect to user preferences and application requirements, while the modularity of the architecture introduces little overhead. I also explore opportunities that arise from segmenting devices into UI and computational resource components, and show that an automated design environment can be created that greatly simplifies custom device design, reducing time-to-market and lowering costs.

Acknowledgements

Many people have contributed over the years to this thesis and to my MIT career.

I would like to begin by thanking my doctoral advisor, Srinivas Devadas, for supporting me, providing the resources I needed to develop the ideas in my thesis, and most importantly, for being a terrific friend. I also would like to thank David Tennenhouse, John Guttag, and Steve Ward for their help and support throughout my Ph.D. work.

My colleagues in the Computer-Aided Automation Group have made coming to work every morning more enjoyable. In particular, Farzan Fallah and Prabhat Jain, with whom I have shared an office for over two years, have been invaluable sounding boards for technical and sometimes not so technical ideas. Dan Engels, and his endless supply of jokes, has made our work environment more pleasant and entertaining. George Hadjiyiannis has spent many hours helping me to flush out my ideas and to make my work more concrete.

Finally, I wish to thank my family without whom I would not be where I am today. My brother has helped me and guided me in the right direction at every juncture in my life. My parents, grandmother, and sister-in-law have been a solid wall of support throughout my Ph.D. and throughout my life. And, finally, I am indebted to my beautiful four-month-old niece who has motivated me to quickly complete my studies and move back to California.

Contents

Figures

Tables

Composable System Resources as an Architecture for Networked Systems

Sandeep Chatterjee

Chapter 1
Introduction

Network devices promise to provide a variety of user interfaces through which users can interact with network applications. The design of these devices stand in stark contrast to the design of personal computers in which new software content is accommodated by increased processor performance. Network device design, on the other hand, must take into consideration a variety of metrics including interactive performance, power consumption, battery life, transaction security, physical size and weight, and cost. Designing a general-purpose platform that caters to all of these metrics for all applications and devices is impractical. For an application mix, a processor architecture and platform can be designed that is optimized for a selected set of metrics, such as power consumption and battery life. Each of these optimized processor architectures and platforms will no doubt be applicable to a variety of devices.

This suggests a modular system architecture for network devices that segments the computational resources from the device UI. Computational resources can be selected for a device UI that is optimized with respect to application mixes as well as to user preferences and metrics. Segmenting out the device UI reduces the complexity of device UIs, simplifying development and lowering costs. At the same time, with little electrical circuitry resident on device UIs, the selected platform can more fully optimize the entire device.

In this thesis, I describe an architecture for network devices that is based on using pluggable system resource modules that can be composed together to create a close-to-optimal platform for a particular application mix and device. Frequently used applications are executed efficiently, while infrequently used applications execute less efficiently. Metrics for calculating efficiencies and selected application domains and mixes are specified by individuals as opposed to one-size-fits-all metrics specified by manufacturers. I show that such a composable system architecture is effective in optimizing system performance with respect to user preferences and application requirements, while the modularity of the architecture introduces little overhead. I also explore opportunities that arise from segmenting devices into UI and computational resource components, and show that an automated design environment can be created that greatly simplifies custom device design, reducing time-to-market and lowering costs.

1.1 Optimal Device-to-Resource Mapping

The architecture of personal computers (PCs) is designed such that all applications are executed adequately well. Hardware accelerators are used for those applications that require additional performance, and it is common to find graphics or audio accelerators in today's PCs. The generality of the PC architecture has been supported by increases in processor clock frequency, memory transaction time, and peripheral bus speeds, but at the expense of increased power consumption, physical size, weight, and cost.

As computing becomes pervasive and starts to impact our daily lives – at home, at work, at school, and everywhere in between – a new computing device is necessary. Ideally, this will be a single device with a user interface that supports intuitive user interaction in all environments and with a computational platform that executes all applications efficiently. This device would function as a cell phone for voice communications, a gaming system for playing video games, a baby monitor for observing the status of an infant, and an instrument for trading securities. Not only would this ideal device have such a robust UI, but its computational resources would also execute applications in these disparate domains efficiently with respect to a set of metrics, including power consumption, battery life, security, interactive performance, and cost.

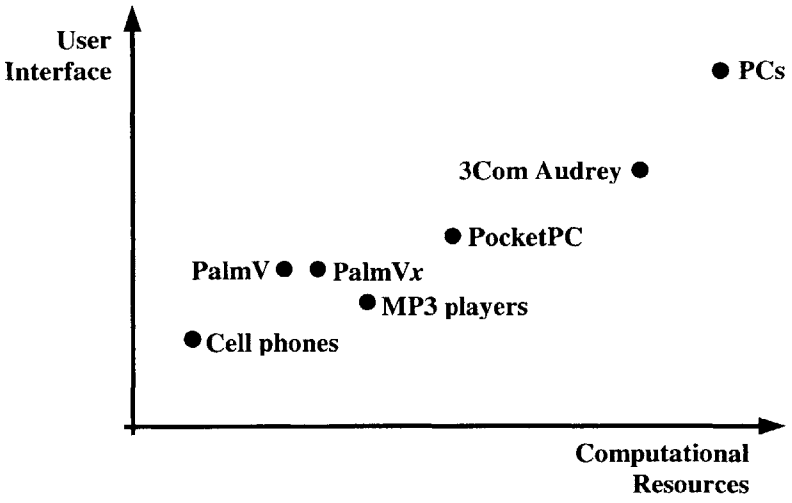


Figure 1: The spectrum of commonly available devices and their relationship with one another with respect to the user interface they implement and their underlying computational resources.

Clearly, such a widely applicable UI cannot be constructed. Nor can a single computational resource platform (e.g., processor architecture, memory type and configuration, and peripherals) be developed that executes all current and future un-envisioned applications sufficiently well so as to cater to individual preferences. Accordingly, the one-size-fits-all UI of the PC is starting to be augmented by a variety of devices, each of which provides an UI that is specific to a particular function or domain. Figure 1 depicts a spectrum of devices that are available today, and their relationship with one another with respect to the UI they implement and the underlying computational resources they use. The user interface axis represents a set of discrete combinations of modalities and user interfaces, while the computational resources axis represents a set of discrete combinations of computational resources, such as processor architectures, application- and domain-specific processors, memory, storage, networks, and peripherals.

PCs comprise for the most part a display, keyboard, and mouse UI while providing an immense amount of computational power. The 3Com Audrey [32] is an appliance for the kitchen or family room, and provides a reasonable-sized display, custom knobs for easily interacting with family-oriented applications, and less powerful computational platform. The Palm V implements

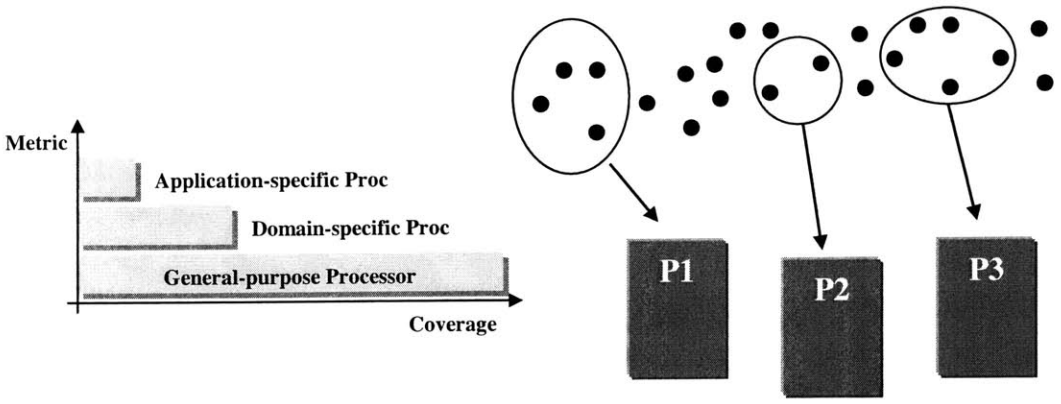


Figure 2: Optimal and close-to-optimal mappings of application mixes to computational resource platforms.

a small stylus-based touchscreen for mobile data entry and time management, and provides limited computational resources. The Palm V*x* implements the same UI as that of the Palm V, but provides more memory [39].

For a given device, one need only use a computational resource platform that efficiently executes applications that are applicable to that particular device UI. Device UIs such as that of the Palm V are sufficiently broad that they can support applications from many distinct domains, including data entry, time management, games, sales force automation, and security. Some of these application domains can be executed efficiently on a single computational resource platform, such as one based on a general-purpose microprocessor. Others, such as graphics-rich games and security applications, can be executed much more efficiently on platforms optimized for such applications (see Figure 2). For example, software-based encryption for universal secure network communications can place an undue burden on a general-purpose processor and increase energy consumption that can be resolved by the use of a hardware accelerator or co-processor [67].

The emergence of new applications and new domains for a particular device UI will sometimes also require users to upgrade or change the device's computational resources. As part of adopting new applications, user preferences may also change. For example, consider an application that enables mobile devices to access enterprise data securely. The need to encrypt

and decrypt large numbers of data packets is oftentimes at odds with long battery life [67]. On the other hand, a new computational resource platform that is optimized for security and energy consumption will enable the user to securely access enterprise data while maintaining reasonable battery life. New software strategies for energy management may be based on hardware mechanisms not yet implemented, and require a computational resource upgrade [52]. All together new hardware platforms that efficiently execute applications for a device UI while catering to individual preferences can also be used [26-28, 50, 51, 68, 74].

Computational resource modules also simplify device software development. Resource modules control and communicate with the device UI, and existing applications and operating system environments associated with the modules can be immediately used, without configuration or porting. Eliminating or at least reducing the software development effort fundamentally affects device cost and time-to-market [75].

Essentially, a single general-purpose computational resource platform cannot be developed such that it is optimal with respect to a set of metrics for all device UIs and applications. Given an application mix for a device UI, computational resources optimize to each individual's liking different parameters such as interactive performance, power consumption, battery life, security, physical size and weight, amount of memory and storage, and cost.

1.2 System Architecture

The mapping of application mixes and user preferences to distinct computational resources have important implications for device design and their underlying architecture. If there are N possible device UIs and R possible computational resource platforms, there are $N \times R$ total devices that must be offered to fully cater to all potential users. The development of so many distinct devices, many of which will have low volumes, result in design and manufacturing inefficiencies and increased costs. Since these devices will be the enabler for the delivery of network-based content and services, reducing unit cost is paramount. Secondly, if a new device (device UI or

computational resources) is necessary before new content or services can be deployed, time-to-market is also important.

An alternative approach to an embedded device design is based on a modular architecture. Recognizing that the same computational resource platform can optimize the performance of application mixes of a number of device UIs, $N + R$ pluggable modules can be created that are *composed* together to realize custom devices that cater to individual needs and preferences. With such a modular solution, changes in user preferences can be accommodated by simply upgrading or replacing either the UI or the computational resources. Important preferences, such as long battery life, can be maintained and even optimized over time as new technologies are introduced. Finally, different computational resources or UIs that more appropriately accommodate new "killer app" content and services (or usage patterns) can be immediately deployed.

This is in contrast to status quo embedded systems that assume a static environment in which the hardware resource specifications of a device remain fixed. Constantly changing network content and associated changes in user preferences negate this assumption, and motivates a modular system architecture. Ideally, the division between the device UI and the computational resources is very close to the UI. This allows the UI, which is a low volume, custom design, to be easily developed and inexpensively manufactured.

In this thesis, I present a unique, strategic, and complete framework for the design, development, and distribution of network devices. This framework is comprised of the novel Composable Systems Resource (CSR) architecture, which proposes a segmentation between the user interface of a device and its underlying computational resources. The CSR architecture pushes this segmentation close to the device-specific UI components, whereby simplifying device UI development and lowering unit costs. Moreover, by exposing low-level UI hardware to the pluggable computational resources, changes in platform technology and network content can most flexibly utilize the UI. I also present an implementation of the CSR architecture that is based on off-the-shelf components, including popular PCMCIA PC Cards and commodity I/O circuits, and show the framework to be a cost-effective one in building highly customized devices and applications that are optimized with respect to user needs and preferences for

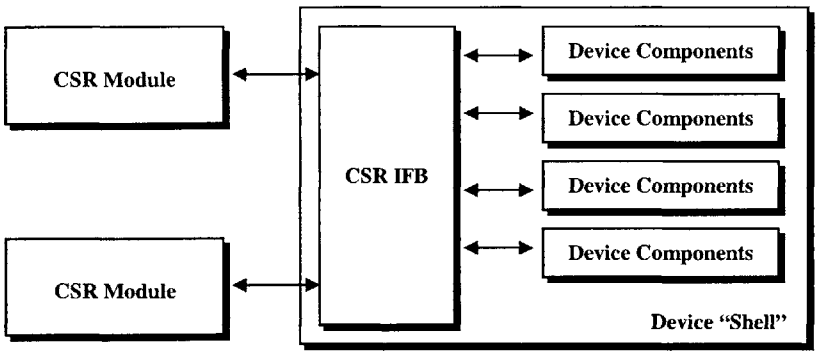


Figure 3: The three blocks of the Composable System Resources architecture are: 1) CSR Modules, 2) the device "shell" comprising various controllable components of the device, e.g., motors, digital-to-analog converters and audio speakers, and 3) a CSR Interconnect Fabric Backplane (IFB) that provides electrical connectivity between the device shell and CSR modules.

parameters such as interactive performance, power consumption and battery life, physical size and weight, ability to cache network content, security, and overall cost.

The following sub-sections further motivate and describe the CSR architecture, its associated design flow, and fulfillment model.

1.3 Composable System Resources

Today, standard battery cells, e.g., AA, C, or D, provide electrical current that powers many of our everyday devices—everything from two-hundred dollar boom boxes and portable televisions to twenty dollar calculators and even free flashlights. These standard battery cells relieve product manufacturers from having to design power systems from the ground up and enable consumers to maintain their devices by simply swapping in fresh batteries. These benefits come with lower unit costs from the standard cells' enormous volumes and manufacturing economies-of-scale.

The Composable System Resource (CSR) architecture is modeled after standard battery cells. Instead of providing electrical power in a standard packaging, CSRs provide computational, network, and peripheral resources as simple building blocks that can be *composed* together by manufacturers and consumers alike to realize a desired systems architecture. The CSR architecture consists of three fundamental components: 1) application-independent CSR computational or peripheral resource modules, 2) a device "shell" comprising the specific I/O components and plastics of a custom device, and 3) a CSR Interconnect Fabric Backplane (IFB) that provides electrical connectivity between CSR modules and the device "shell", and handles miscellaneous resource management issues. This entire system architecture is shown in Figure 3 [12-17].

The modularity of the CSR architecture addresses the fundamental issues surrounding network devices. First, a device UI or "shell" can be designed and manufactured without having to specify the computational resources apriori. Immediately before distribution, the appropriate CSR modules can be selected based on "killer app" content available at that time. This just-in-time kitting [40] between the device "shell" and the computational resources of the device insulate device manufacturers from fluctuations in content development and market adoption rates. Secondly, if changes in "killer app" content happen after a device has been introduced to consumers, manufacturers can immediately introduce next-generation devices based on existing "shells" and more resource-rich CSR modules. Similarly, consumers can also easily change or augment the capabilities of their "shells" by simply swapping in additional CSR modules or by replacing existing modules.

In this thesis, I present one implementation of the CSR architecture based on popular PCMCIA PC Cards [3, 49, 55, 60]. In this implementation, CSR peripheral modules are infact off-the-shelf PC Cards, while CSR computer modules are packaged as PC Cards and provide the electrical interfaces necessary to communicate with the PC Card peripherals as well as with components of the device "shell". This implementation takes advantage of the market penetration of PC Card peripherals, and, at the same time, leverages the low costs and availability of PC Card packaging and connectors.

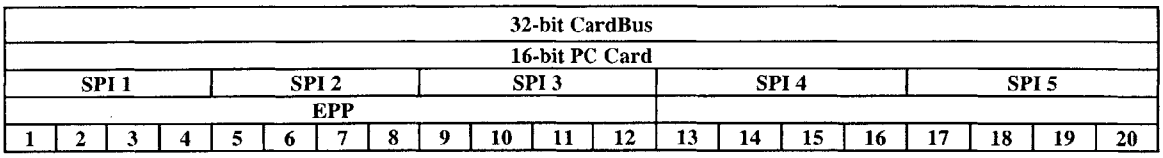


Figure 4: Multiple interfaces supported by the CSR architecture. Non-contiguous horizontal segments from each row can be coordinated together to form a CSR module's overall electrical interface.

The following sub-sections further describe the PC Card-based CSR implementation.

1.3.1 Configurable Universal Interface

The CSR architecture is centered on a configurable universal interface. As a CSR module is connected to an IFB socket of a device "shell", the IFB programs the module with information that allows the module to configure its I/O interface so that it can control and communicate with the components of the device "shell". The universality of the interface allows CSR modules to be general-purpose modules that can be used across application domains and market segments.

Figure 4 shows a set of standards-based interfaces that are supported by the current implementation of the CSR architecture. The current implementation supports both the 32-bit CardBus [63] and the original 16-bit PCMCIA PC Card [3] interface. It additionally supports a variety of common narrow datapath (hereinafter, narrowpath) interfaces, including simple bit-toggling (shown in Figure 4 as simply numbers), Serial Peripheral Interface (SPI), Microwire, RS-232, 8-bit parallel, and Enhanced Parallel Port (EPP). Narrowpath interfaces are typically found on commodity off-the-shelf components as they limit the pin count of a component, and thereby its overall size and cost. Multiple non-overlapping narrowpath interfaces (as shown in Figure 4) may be activated in parallel, and provide a simple architecture for designing a device "shell".

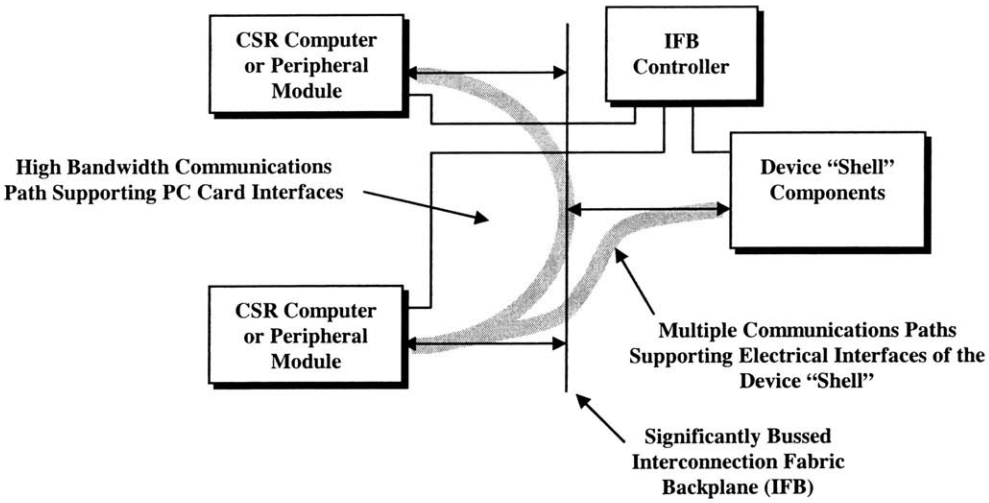


Figure 5: Multiple bus communication protocols over a single bussed environment.

1.3.2 Significantly Bussed Interconnect

In the PC Card-based CSR implementation, CSR computer modules communicate with other CSR computer modules, PC Card peripherals, and with the components of the device "shell". Typically, device "shells" are made from commodity components based on standards-based narrowpath interfaces. In order to reduce IFB size, cost, and power consumption, the current implementation uses a bussed architecture that interconnects CSR modules and narrowpath-based device "shell" components.

Figure 5 depicts a significantly-bussed IFB architecture capable of supporting two CSR modules. Each CSR module can communicate with each other as well as with device "shell" components over a significantly bussed interconnect. The IFB Controller handles resource management and prevents bus contention. By limiting the number of signals that must traverse through the IFB Controller, we reduce IFB Controller pin-count, package size, and die size,

thereby reducing overall system size and costs and making CSRs appropriate for a wider array of devices.

The direct connection between CSR modules and device "shell" components and the variety of electrical interfaces and protocols used thereby require time-multiplexing different interfaces over the shared bus. In particular, CSR computer modules time-multiplex the appropriate set of electrical interfaces over its connector so as to communicate with the fixed interfaces of either another CSR module or device "shell", with the IFB Controller preventing bus contention and providing target disambiguation.

The PC Card-based CSR architecture with a significantly bussed IFB is a simple and cost-effective system, which obviates the need for expensive and area consuming buffer memory on the IFB. As clock speeds and overall performance of PC Cards as well as of commodity analog and digital I/O chips (e.g., analog-to-digital converters, digital-to-analog converters) increase, the bandwidth limitations of time-multiplexing communications will diminish. Nonetheless, this thesis demonstrates that for many common applications, the herein described PC Card-based CSR architecture is sufficient and adequate.

1.3.3 Timer-Based Data Buffers

Timer-based data engines within CSR computer modules allow directly interfacing commodity analog and digital I/O components onto IFBs. Most commodity I/O components simply provide the I/O functionality, and assume that a dedicated controller exists to handle jitter and to stream data into and out from them. PC Card-based CSR computer modules subsume the role of dedicated controllers by providing configurable timer-based data buffers together with configurable protocol engines.

Figure 6 illustrates the timer-based data buffer unit used within PC Card-based CSR computer modules. Software can program each unit to output to or sample data from an external component at a preset time interval, and delay packets can be written to the data buffer itself to override the preset time interval. Timer-based data buffers together with configurable standard

electrical interfaces simplify device "shell" development. Off-the-shelf commodity components may be placed onto an IFB, without having to design any glue-logic, controller or timer circuit.

1.3.4 Symmetric System Composition

Most systems can be segmented into computational resources and peripheral (including networking and I/O) resources. System symmetry refers to the interchangeability of computational and peripheral resources within the overall system without any adverse performance degradation. Today, most systems are not symmetric. Personal computer motherboards or CompactPCI [36, 61] industrial backplanes with peripheral plug-in slots cannot support a computational resource plug-in card as well as it can in the primary Slot 1 processor slot. Differences in electrical interfaces, form factors, and overall use between peripheral resources and computational resources do not support system symmetry. Non-symmetric systems result in design flexibility limitations, confusion on the part of consumers trying to upgrade their systems, and inefficient inventory management and assembly by fulfillment centers.

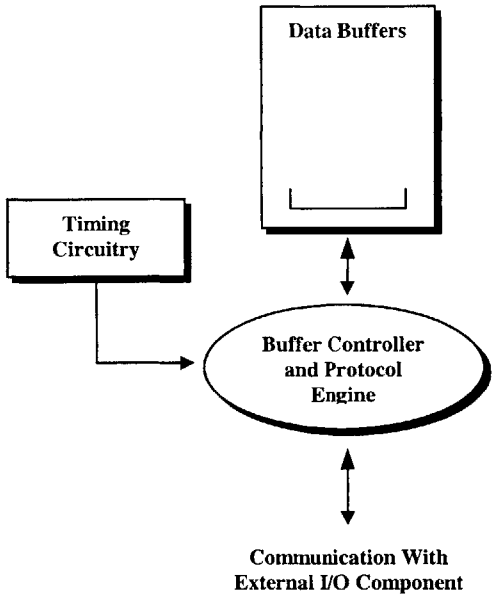


Figure 6: CSR Computer Modules include timer-based data buffers that output or sample data from external I/O components. The timer-based data buffers support the use of simple commodity I/O components without dedicated controllers or glue-logic.

The CSR architecture is fully symmetrical – CSR computer modules or CSR peripheral modules may be connected to any IFB sockets equally well and infact, two CSR computer modules can be connected together to form a multi-computer. In order to support fully symmetrical composition of CSR modules, the IFB controller acts as the central resource manager.

Having described the salient features of the CSR architecture, the next section describes a simple design flow inspired by the architecture.

1.4 CSR-Based Device Design Flow

A simple device design flow enables Web portals and any company whose core competence is not in developing devices to build *branded, custom, and content-specific* devices themselves. Such a design flow removes the dependence and uncertainties associated with contracting a third-

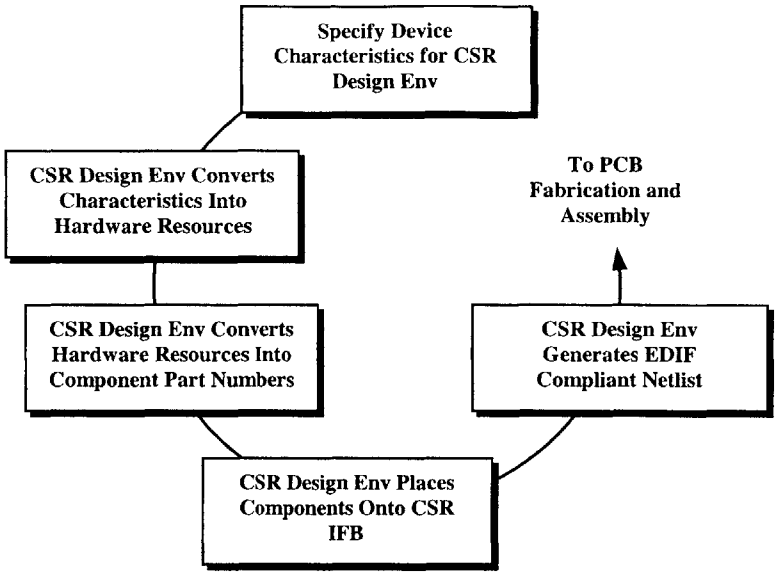


Figure 7: CSR-based device design flow and development steps.

party device manufacturer, as well as lowers unit costs and affords the company to choose from a variety of business models that make sense for the device.

If the device design flow is sufficiently simple then designers themselves can fully develop a device, without having to involve a set of engineers. Removing the need to communicate creative ideas and concepts from designers to engineers reduces overall time-to-market and system errors borne out of miscommunication. In the Post-PC Internet, as the characteristics of content-specific devices are used to enhance and differentiate content and increasingly more of our analog devices become digital, a simple design flow empowering creative designers themselves to fully develop devices will be necessary.

The CSR architecture is uniquely positioned to fill this need. The CSR architecture is infact a greatest-common-denominator architecture in which the system resources common to most devices are CSR modules. Only the custom look-and-feel and I/O interactivity of the device must be designed and developed; the appropriate CSR modules, which form the basis for the device platform, are simply purchased.

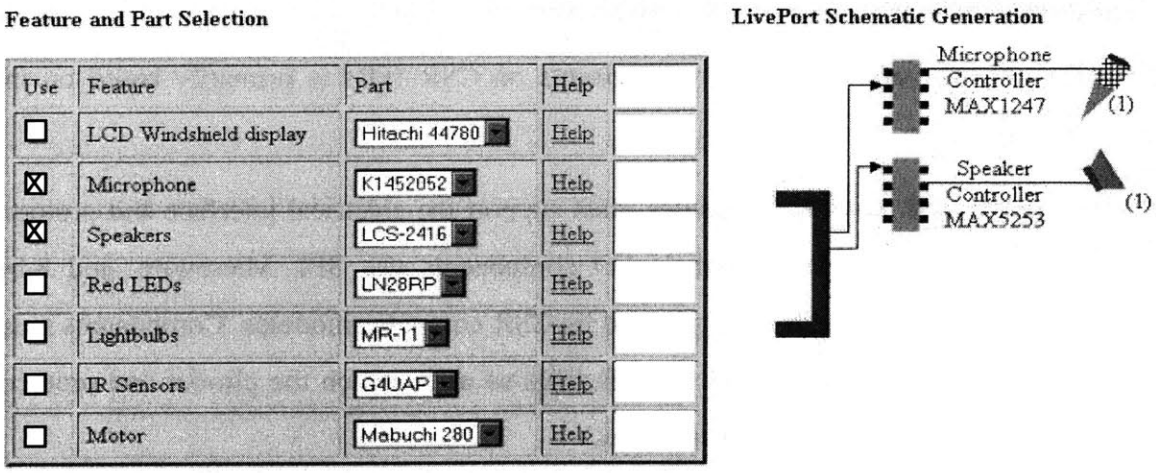


Figure 8: Device Builder Characteristic Selection

1.4.1 Characteristic-based System Design

The CSR architecture structures and greatly simplifies device design and development, while providing sufficient flexibility to support a wide assortment of devices. This structured design flow can be automated to a large degree and further simplified.

The first step in CSR-based systems design is to specify the general characteristics of the device. As shown in Figure 8, this may simply involve checking off a set of desired characteristics from an enumerated list. The automation environment basically comprises a set of heuristics for realizing various device characteristics. For example, if a device is to have the ability to output audio, it must have a digital-to-analog converter to convert digital audio representations into analog signal levels that can be output through an audio speaker.

Once a desired set of device characteristics have been transformed into a set of simple I/O resources, the next step is to choose the actual off-the-shelf component part numbers and to appropriately place them on the CSR IFB.

1.4.2 Component Selection, Placement, and Quality-of-Service

Selection and placement of off-the-shelf components on CSR IFBs is primarily based on the following properties:

- Interface support: CSR computer modules must support the electrical interface and protocol of each component. Many off-the-shelf I/O components use SPI, Microwire, and 8-bit parallel interfaces, all of which are supported by CSR computer modules. Components must be placed onto appropriate datapaths of CSR IFBs so as to match the pin-out and protocol engine positioning of CSR computer modules.
- Quality-of-service: The system must guarantee that peak bandwidth requirements between CSR modules as well as between CSR computer modules and device "shell" components can be met over the time-multiplexed bussed CSR IFB environment. Specified heuristics for low-quality audio, CD-quality audio, CSR module aggregate average bandwidth (e.g., 16-bit PCMCIA 1Mbit/sec Ethernet network PC Card), combined with component datasheet information (e.g., packet size, payload, clock frequency) allow design-time quality-of-service guarantees.
- Miscellaneous properties, including load capacitance, unit cost, availability and lead-time, size, power consumption of the component.

1.4.3 Netlist Generation and Software Configuration

Once components matching the desired characteristics have been selected, bandwidth and quality-of-service have been guaranteed, a system netlist can be generated. A netlist provides a textual representation and specification of pin-to-pin connections between all components. A netlist in the popular EDIF [22] format combined with layout information (commonly available for commodity components) can be automatically routed for printed circuit board fabrication and component population.

Information about the components of the device "shell" is also stored on the IFB. This allows fully generic CSR modules to be connected to a custom device "shell" and have the ability to

communicate with and control each of the "shell's" components. This information may be stored in ROM as a software readable data or it may simply be a short identifier that can be dereferenced (possibly over a network) to access the actual information.

Essentially, CSR modules together with the CSR-based design tool support the rapid development of complete working devices by simply selecting desired device characteristics.

1.5 CSR-Based Manufacturing and Fulfillment Model

Just as content-based devices provide new challenges in system architecture design, they too provide challenges in device manufacturing and order fulfillment. Abrupt changes in "killer app" content or market adoption cycles can leave devices unusable or unsaleable. The CSR architecture inspires a unique manufacturing and fulfillment model that is both strategic and economical, especially for smaller portals without large capital resources.

The universal interface between CSR modules and actual device "shells" support just-in-time kitting of the general CSR module to the specific "shell". This just-in-time kitting affords two fundamental benefits in manufacturing and fulfillment:

- Reduced inventory risks. As companies are reluctant or unable to allocate large amounts of capital for device inventory, the separation of device "shells" from CSR modules offer an unique distribution model that minimizes inventory space and dedicated capital. As device "shells" are specific to a particular company, they must be custom manufactured and expensed by the company. CSR modules, which are general and customer unspecific can be mass manufactured, and assembled into device "shells" only upon order fulfillment. In this fulfillment model, only upon fulfillment, must the company allocate capital for and absorb the costs of the entire device – "shell" and CSR modules.
- Immediate Deployment of Content. New content and services for existing device "shells" may be immediately deployed by simply changing or augmenting CSR modules such that the appropriate hardware resources are available. This frees content developers from being limited by the established base of device platforms (e.g., already deployed devices with CSR

modules), and instead empowers them to create compelling applications knowing that changes in hardware resources can be immediately accommodated.

- Recovery Of CSR Modules On Unsold Device Inventory. As new "killer app" content usurps the place of old "killer app" content and leaves unsaleable device "shell" inventory, CSR modules may be recovered from these devices and used in other devices, salvaging the platform's cost.

CSRs bring all the benefits and convenience of batteries to network devices. Essentially, CSRs can be thought of as the *batteries of the 21st century*, offering three degrees-of-freedom, in choosing (a) the computing environment, (b) the peripheral environment, and (c) the software environment.

1.6 Summary Of Contributions Of This Thesis

This thesis makes contributions in the areas of computer system design, device development paradigms and cycles, and man-to-machine interfaces. These are outlined in the following sections.

1.6.1 Computer System Design

Devices for accessing and interacting with network content will play an increasingly large part of our everyday lives – at home, at school, at work, and on the road. These devices present a set of requirements in design and development, manufacturing and distribution, and maintenance that are fundamentally different from today's consumer electronic devices and personal computers. Understanding these requirements will lead to recommendations for future systems design, as well as a set of metrics by which to judge network content device architectures.

The CSR architecture represents a reversal in today's computing paradigm: instead of connecting a device to a computer, the CSR architecture proposes to connect a computer (module) to a device. This modular architecture recognizes the issue of volatility of system

resources in network devices, and proposes the notion of *composability* of processor, network, and peripheral resources at both the consumer and manufacturer level. Similar to standard cell batteries, composable systems allow manufacturers to easily design and develop devices by simply following a few composition rules, and allow consumers to maintain and upgrade their devices as desired.

CSR-based composability provides a simple and cost-effective means to optimize devices along various dimensions of user preferences. Unlike traditional PCs that aim to provide maximum runtime performance given only the constraint of cost, network devices encompass a larger variety of constraints and trade-offs, including power consumption and battery life, interactive runtime performance, physical size and weight, amount of storage available to cache network content, security, and cost. The modularity of the CSR architecture allows individuals to configure their devices and optimize them to most fully meet user preferences instead of settling for one-size-fits-all devices as determined by manufacturers.

The CSR architecture also introduces the notion of system symmetry, which places peripheral and computational resources on equal footing. Any CSR module – computational resource or peripheral resource – can occupy any CSR IFB socket equally well without degradation in system performance. Symmetrical systems are important in providing manufacturers and consumers flexibility to address changes in resource requirements for "killer app" content and new preferences.

In order to better understand the requirements for network content devices as well as to understand the feasibility of the CSR architecture, this thesis presents an implementation and complete specification of the CSR architecture. An implementation of the CSR architecture that is based on popular PCMCIA PC Cards and off-the-shelf commodity I/O components is presented. Support for PCMCIA PC Cards as CSR peripheral modules allow instant availability of a broad spectrum of affordable peripheral and network modules, while support for off-the-shelf commodity I/O components allow fast and inexpensive development of device "shells". Analytical and experimental analyses demonstrate that the PC Card-based CSR architecture to be a cost-effective one in building optimized devices while incurring small overhead.

1.6.2 Application Development

In the CSR architecture, pluggable computational resource modules are used to optimize the performance of devices along a set of parameters so as to cater to individual preferences. This means that a single device (or device UI) can be powered by a variety of different computational resources. The implications for application development are enormous. Applications, which are currently written for a particular platform, such as WindowsNT or Linux, can no longer assume the existence of such a standard target environment. Instead, applications will be written for device UIs, and optimized by selecting CSR modules.

1.6.3 Device Development Cycles and Paradigms

Network devices represent a fundamental shift in the go-to-market model for consumer devices. Whereas PCs and consumer electronics have traditionally been retail distributed, Internet devices will typically be distributed through service providers, bundled together with their content or service offerings. In this scheme, the device is merely an enabler to the delivery of the real product – the service provider's content or service. Accordingly, device development must be fast, simple, inexpensive, and must easily cater to changes in "killer app" content. A device design flow that empowers creative designers, with minimal technical ability, to fully develop custom devices, rapidly, accurately and cost-effectively is necessary.

In order to more fully understand the requirements of such a design flow, the various aspects of device design has been analyzed and quantified. Given these requirements, the Device Builder Environment based on the CSR architecture has been specified.

1.6.4 Man-to-Machine Interfaces

The primary technical determinant of a product's success is oftentimes not by the number of functions it performs but by the simplicity and grace of its user interface. Much work in user interface design has focused on software user interfaces such as graphical user interfaces or organizational schemas, with much less work in the area of physical design and tactile interfaces.

However, highly personalized and graceful software user interfaces represented through a non-intuitive and difficult-to-use physical devices may lead to negative acceptance. The man-to-machine and user interface aspects of devices are not the main focus of this work, but the end-to-end device design flow and system architecture presented herein will enable more research in physical user interfaces. A set of CSR-based network devices that allow simple and intuitive interaction with network content has been built. The development of these devices not only shows that different user interfaces are oftentimes more appropriate for interacting with different classes of content and services, but also demonstrates the simplicity with which CSR-based devices can be designed and optimized.

1.7 Organization Of This Thesis

This chapter began with an introduction to devices based on network content and services. The dynamic nature of network content and services motivated the need for a similarly dynamic systems architecture that need not be fully specified prior to device manufacturing and distribution. The chapter went on introduce the Composable Systems Resource architecture for network devices, and described how the fundamental characteristics of the CSR architecture make it appropriate for use in network devices. The chapter also discussed the design flow made possible by the structure of the CSR architecture, and how an associated design automation environment reduces design errors and time-to-market. Finally, the chapter described a just-in-time fulfillment model based on the CSR architecture that is both strategic and economic in its importance to small and large companies.

Work related to the issues addressed by and underlying the Composable System Resources architecture is presented in Chapter 2. This related work is segmented into three areas. The first examines various systems paradigms for network devices. The second area examines modular hardware systems in general. The third area examines prior work on design automation tools.

Chapter 3 describes the CSR architecture, and, in particular, fully specifies the PC Card-based implementation. The chapter also provides background information on the PCMCIA Standard and the two primary PC Card interfaces.

Chapter 4 evaluates the PC Card implementation of the CSR architecture along a variety of metrics, including communication I/O performance, power consumption, and cost.

Chapter 5 describes the CSR-based design flow and specifies the CSR Device Builder automation environment.

Chapter 6 concludes this thesis and proposes directions for future enhancements and developments.

Chapter 2
Related Work

This chapter presents previous work related to the CSR architecture as well as previous work related to the CSR-based device design flow. The first section discusses related work on the computing paradigms and systems architectures of network devices. Much of the work in this area has focused on PC-based systems versus processor and network architectures for distributed devices. The next section discusses related work on modular consumer and industrial systems. For the most part, this work has addressed issues surrounding electrical interface design and bus architectures. The last section discusses related work on electronic design automation (EDA) tools that simplify systems development. Much of the work in this area has focused on creating abstractions for chip design and verification.

2.1 Related Work On Computing Paradigms

2.1.1 Centralized Computing Paradigm

The centralized computing paradigm is based on a single powerful central computer that provides computational resources to a set of peripheral devices connected to it. Today, most home

personal computers (PCs) have a pair of speakers, a printer, a digital camera, a monitor, a keyboard, and a joystick connected to it, each of which has time-multiplexed access to the PC's resources. Newer devices such as network phones and MP3 audio players use wireless communications between the PC and each device to enhance mobility.

A central PC that services multiple devices enables prorating the cost of the PC over all of the devices. Moreover, the standard interfaces commonly found on PCs, including serial and parallel ports [8, 9, 46], Firewire [2, 47], and Universal Serial Bus (USB) [4, 7], simplify and expedite device development. The structured software environment of PCs together with the well-defined electrical interfaces of the PC ports make complete device development unnecessary; device functionality must only be built around the hardware and software PC port specifications.

However, as transistor sizes continue to shrink and silicon chips become cheaper, the cost advantage of PC-connected peripherals diminishes. At the same time, the geographical limitations of tethered or wireless PC-connected devices oftentimes limit their mobility, and sometimes their overall utility.

Multiple devices, each with their own software applications and device drivers, supported by the hardware resources of a single PC, make application-level and systems-level software more difficult to develop, debug, and maintain.

The technology driver limiting the appeal of a centralized computing paradigm is power consumption. The power consumed per bit transmitted or received over a wireless network is asymptotically approaching its theoretical minimum, while the power consumed per Millions of Instructions Per Second (MIPS) of a silicon microprocessor is continuing its rapid decrease with improvements in technology (as predicted by Moore's Law). Accordingly, power savings can be achieved by minimizing network usage through data compression and encoding schemes that are then uncompressed and decoded by a processing environment local to the device.

The CSR approach is an eclectic one combining the abstraction and modularity benefits of PC-based devices, without incurring many of the disadvantages of centralized computing. Essentially, the modularity and separation between PC ports and PC-connected devices is similar

to that between CSR modules and CSR-based device "shells". As a CSR module is local to each device, the power consumption and software development concerns are minimized.

2.1.2 Distributed Computing Paradigm

Distributed computing systems are devices with local computational resources that are networked together. Sun Microsystem's Jini and Microsoft's Millennium are software environments for supporting distributed computing [38, 43].

Jini builds upon Java's platform independence and delivers a set of lookup and discovery protocols that enable heterogeneous devices to leverage one another's services. Jini uses the notion of "leasing" resources from other networked devices so as to leverage their services. Consider a digital camera that has its own local computational and storage environment for taking and maintaining digital images. Once in close proximity to a printer, the digital camera can "discover" the printer's print service and use it to produce hard copies of the images. By leasing the print service and the hardware resources of the printer, the camera need not use its own computational resources for that particular service [21].

Jini provides a software environment for discovering network-based services and for supporting communications between heterogeneous devices. It does not directly address the issues surrounding local hardware resource deficiencies arising from constantly changing network content. Within Jini's service model, applications may be divided into various proportions of remote services and local "stubs" so that local hardware resources are fully utilized. Oftentimes, application division is tedious, and the optimal proportion of local stub size to remote service size is difficult to maintain given a dynamically changing processor load (from changing content, applications, and services).

[37] describes a system architecture and implementation of a distributed environment based on nearest-neighbor communications. Each node of the environment is a low cost, resource-limited processor that only communicates with its closest neighbors who may or may not be the target node. If the desired information or service is not available at that node, the node

communicates the message with *its* neighbors, thereby propagating the message with little transmission power and with limited airwave (bandwidth) pollution. This architecture is effective for connecting simple devices such as remote controls and small appliances, but may not provide a sufficiently robust platform for rich real-time content and services.

[70, 72, 73] showcase the issues underlying and technology developments required for realizing the ubiquitous computing framework. In [71], Weiser describes the differences between today's computers and ubiquitous computing as follows:

> *Suppose you want to lift a heavy object. You can call in your strong assistant to lift it for you, or you can be yourself made effortlessly, unconsciously, stronger and just lift it.*

Personal computers (PCs) and personal digital assistants (PDAs) allow access to virtually all content and services. Similar to the strong assistant called in to lift the heavy object, PCs and PDAs oftentimes lead to non-intuitive and disruptive interaction with classes of network-based content and services. The CSR architecture allows the rapid and cost-effective definition and construction of a variety of devices in all shapes and sizes, through which we can effortlessly and unconsciously interact with content and services throughout our everyday lives.

2.2 Related Work On Modular Hardware Systems

2.2.1 PCMCIA PC Cards

[3, 49, 55, 60, 63] detail the specifications for PC Cards, which are credit-card-sized integrated circuit cards that allow upgrading and augmenting the peripheral capabilities of mobile computing environments. The Personal Computer Memory Card International Association (PCMCIA) defines and maintains the standard and promotes interoperability between computer manufacturers and PC Card manufacturers. Today, many peripherals can be found packaged as PC Cards, including network interface cards, joysticks, volatile and non-volatile memories, digital cameras, and wireless communication devices. The convenience and economic benefits of PC Cards have led to their widespread adoption with not only laptop computers supporting the standard, but also some television set-top boxes, DVD players, and digital cameras providing PC

Card sockets. Essentially, PC Cards address the need to upgrade and augment the peripheral resources, but not the computational resources, of a computing environment.

[35] describes a "peripheral" processor PC Card. Peripheral processors allow augmenting the resources of a primary processor, and rely on the primary processor system to configure it and to maintain proper operation. As new software functionality, content, and services become an increasingly important part of network devices, the need to upgrade both the computational and peripheral resources of a device will be necessary. Power consumption concerns and cost issues will require that primary processors and not additional peripheral processors be upgradeable and be made modular. The CSR architecture described herein presents such a modular primary processor architecture.

2.2.2 CompactPCI

CompactPCI is an industry standard bus and edge-card form factor popular with various types of industrial peripherals and computer modules. The CompactPCI architecture leverages the Peripheral Component Interconnect (PCI) specification [64] and adapts it for a more robust mechanical environment [61].

CompactPCI is a bus standard, and not a complete system specification for consumer or enterprise network devices. For instance, CompactPCI is not a symmetric bus – computer modules must be connected to a predetermined location on the bus, while peripheral modules may be connected to any of the remaining locations. This makes system assembly and consumer upgrades confusing and potentially more expensive. Moreover, the standard does not support simple and inexpensive device "shell" design as all components on the bus must use the CompactPCI interface. The CSR architecture's use of a time-multiplexed multi-bus electrical interface allows seamless communication between computer and peripheral modules as well as with commodity off-the-shelf components of device "shells".

2.2.3 NuMesh

The NuMesh project proposed the use of Lego-like modules that can be connected together to realize high-performance computational structures. The NuMesh system recognized the bandwidth and latency limitations inherent to serialized backplane bus architectures, and instead suggested a three-dimensional nearest-neighbor topology. Communication between nodes is handled by a simple hardware element that relies on compile-time pre-configuration instead of dynamic run-time configuration. The system combines plug-and-play hardware modularity with efficient inter-node communications [69].

Where NuMesh tries to achieve maximal communications bandwidth over a modular substrate, the CSR architecture stresses simple and low cost device design with bandwidth sufficient for popular multimedia applications.

2.3 Related Work On Design Automation Systems

Previous work on design automation systems has primarily focused on hardware description languages, compilers and synthesis tools [6, 10, 53, 54, 56], as well as black box language abstractions and graphical representation methodologies [20, 48]. As system complexities have increased, more recent work has focused on design verification, including test vector generation and coverage metrics [23, 24].

2.3.1 Design Automation Environments

Although they have simplified and automated many tedious aspects of systems design, previous work has not achieved a sufficiently high design flow. Similar to graphical user interfaces (GUIs) that improved upon command-line-argument-based interaction with application programs, and natural language interfaces that promise to one-up GUIs, a high level design flow for network devices will simplify device development, thereby reducing time-to-market, lowering unit costs, and enabling the deployment of custom devices for finer market segments.

This thesis presents a "natural language" design flow for the development of network devices. This design flow merely expects a designer to convey to the design environment the high level characteristics of the envisioned device. For example, a voice-enabled network radio may consist of the ability to output and receive audio, specified to the design environment as a set of checkboxes. The design environment then converts these checkboxes representing the general characteristics of the device into instantiations of commodity, off-the-shelf analog-to-digital converters (to receive audio), digital-to-analog converters (to output audio), as well as other requisite components for proper operation. The output of the design environment is an Electronic Design Interchange Format (EDIF) [22] netlist that can be routed for printed circuit board (PCB) fabrication and component population.

Literature has not addressed such a high level and holistic approach to systems design. Although this thesis does not present a general solution to the problem of automated systems design based on an enumerated list of device characteristics, it presents a solution for devices utilizing the CSR architecture.

2.3.2 Future Design Automation Environments

Technology developments may one day result in design automation environments capable of synthesizing high-level device characteristics directly into system-on-chip (SoC) designs [11]. These highly integrated mixed signal SoC designs would reduce the energy consumed by devices, and lessen the gap between the energy demands of emerging portable devices and the energy capabilities of batteries [28].

The device design process can be segmented into three distinct pieces: 1) prototype development, 2) prototype iteration, and 3) final product development. Iterating over multiple device prototypes allows designers, working in conjunction with focus groups, to understand whether the device will really serve the needs of the market. Once the proper device has been prototyped, the final product development step begins.

A design automation environment that maps high-level characteristics to SoC designs will most probably be used during the final product development step and not be included in prototype development. This is because there are significant financial resources required to implement mask application-specific integrated circuits (ASICs) based on advanced deep sub-micron technologies (0.35μm and smaller). As transistor sizes continue to shrink faster than metal lines, interconnect delay becomes the dominant component of signal delay, dwarfing gate delay. In order to reduce interconnect delay, additional metal layers are typically used to provide more robust routing resources. Since mask development for each layer incurs significant cost, the use of additional layers increases the non-recurring expense (NRE) of each custom design, and it is common to find NREs in excess of $100,000 [76].

High-volume devices, such as cellular phones, can absorb such large NREs as the NRE is prorated over millions of units. However, as companies migrate away from one-size-fits-all devices to more custom ones [25], the high NREs of custom ASICs will become prohibitively expensive. Even at reasonably high volumes, such as 500,000 units per annum, the unit cost of custom ASICs is at par with more general-purpose platforms, such as field programmable gate arrays [45].

Essentially, the use of SoCs and high-level design environments that rapidly map device characteristics to custom ASIC SoCs will reduce the system's power consumption but will adversely impact its price point. Accordingly, custom solutions will be used to implement extremely high volume devices or devices for which reduced power consumption is critical. Devices not falling into these categories will be based on off-the-shelf general-purpose or domain-specific components.

Chapter 3
Composable System Resources

Composable System Resources (CSR) is a system architecture for simple battery-like usage of computer and peripheral modules in lieu of traditional embedded solutions. Small-sized computer and peripheral modules may be composed together by designers and consumers alike to realize a desired computational system for network devices. Composable building blocks support the development of highly customized network devices that cater to the specifications and needs of finely divided market segments. Instead of mass-produced one-size-fits-all devices that try to amortize the long development time and costs over a large number of devices, custom devices promise to deliver superior interactivity and user satisfaction.

As depicted in Figure 9, CSR is centered on an universal interface specification. This specification is flanked on one side by a device "shell"* and on the other side by CSR modules. This universal interface allows for independent development of the "shell" without specifying apriori the actual computational resources, e.g., processor MIPS, amount of memory, network type, and software environment, of the device. As computing resources become ubiquitous and power all devices, the modularity of CSRs affords:

- Development of a variety of innovative network devices without encumbering designers with the task of specifying and building the underlying computational resources;
- Just-in-time kitting of appropriate computational resources to device shells during manufacturing based on "killer app" content and services available at the time of distribution;
- Recovery of soft packaged CSR modules and their costs from device inventory that cannot be sold;
- Simple means for users to upgrade, augment, and maintain the computational and peripheral resources of expensive or long lifetime device shells; and,
- Pluggable re-use of expensive computational and peripheral resources across multiple devices.

---

* A network device "shell" is comprised of all components of a network device without the actual computer components.

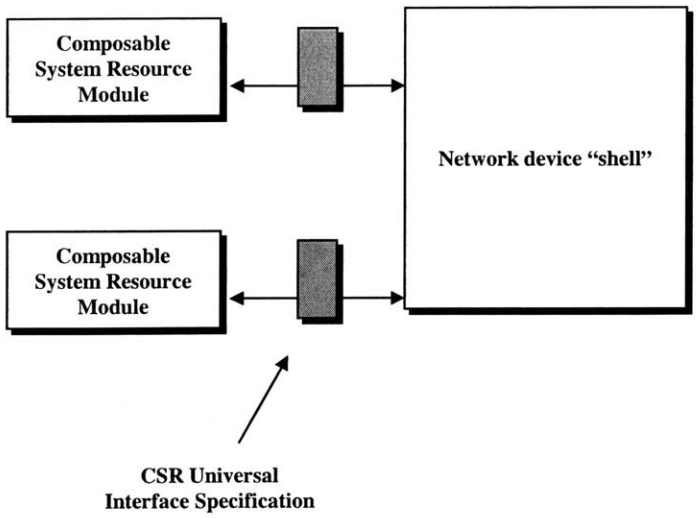


Figure 9: Composable System Resources interfacing with the hardware components of a network device.

3.1 The CSR System Architecture

The CSR System Architecture addresses the need for a set of simple Lego-like building blocks that streamlines the development and maintenance of network devices. The CSR architecture is fundamentally different from most bus architectures that try to achieve maximal performance given a set of constraints, such as datapath width, number of loads, interconnection medium. For instance, the Peripheral Component Interconnect (PCI) bus architecture was borne out of a need for higher bandwidth and lower latency communications to support multimedia applications in personal computers (PCs). The actual device within which the PCI bus will reside as well as the device's functionality is left as an afterthought.

Figure 10 depicts the system architecture for status quo PCs. A pluggable processor module, consisting of Intel's Slot 1 Pentium processor Single Edge Contact Card (SECC) or Socket 7 processor chip, provides the primary computational resource of the system. A pluggable peripheral module, consisting of a PCI or ISA edge card or a PCMCIA PC Card, provides a

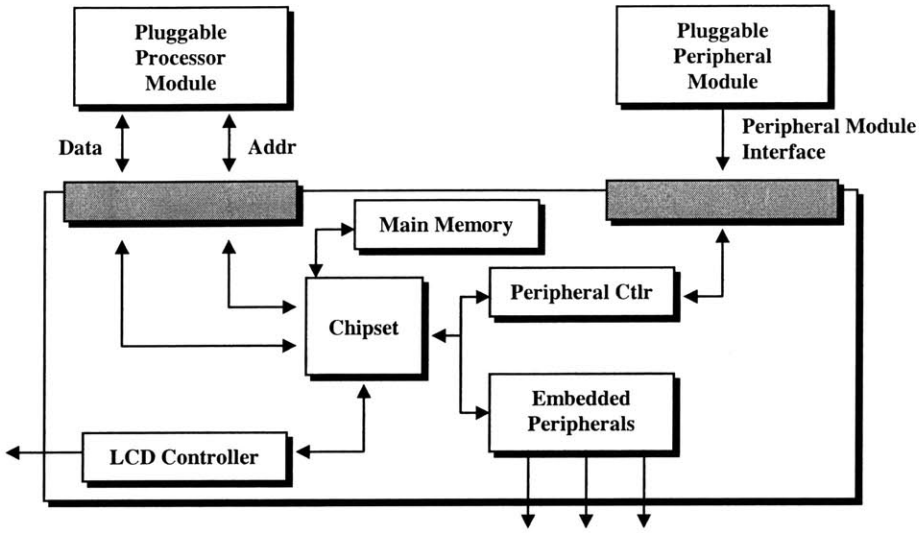


Figure 10: System architecture for status quo personal computers.

means for peripheral expansion. The motherboard houses the chipset, which interfaces the processor bus to the peripheral bus, main memory, and other embedded peripherals. This architecture affords motherboard and system manufacturers the flexibility to differentiate their products with various combinations of chipsets, amount and type of memory, and supported peripherals.

Although attractive for PCs, it is a thesis of this work that this type of architecture is inappropriate for network devices. A more attractive architecture for network devices is based on fully segmenting the device-specific user interface from the underlying computational resources. By eliminating as much high-speed circuitry and electronics in general from the device UI, its unit cost, design complexity and time can be reduced. This, in turn, will spur the development of a variety of innovative user interfaces for interacting with network content and services. Within this flurry of development, some UIs will no doubt fail. By reducing the unit cost of device UIs to a minimum, such a segmented architecture can mitigate development risks and cost. Moreover, by migrating silicon-based circuitry with low shelf life (as dictated by Moore's Law) away from the device UI, upgradeability and flexibility is enhanced.

This dissertation investigates the optimal level at which to segment device UIs from their underlying computational resources. Ideally, device UIs will comprise the ergonomic design, knobs, and buttons that define the physical user interface together with a set of directly interconnected connectors for receiving pluggable computational resources. In reality, however, many device UIs will also include some circuitry required to support the custom physical UI. Oftentimes, this circuitry will be analog components that are necessary to interface with users and their "real world" environment. The architecture will determine the cost and amount of circuitry that is housed by the device UI, and how much of it can be abstracted away to plug-in modules.

The CSR architecture aims to achieve sufficient performance for plug-in modules, but more importantly, aims to provide a structured framework around which the actual functionality of device UIs can be developed. Essentially, the CSR architecture aims to support, simplify, and facilitate device design, development, and maintenance. The primary CSR system architecture design issues include:

- **Support for different back-end processing environments**. The CSR system architecture is more of a computing paradigm than a processor architecture. It promotes the use of hardware and software building blocks for the design, development, and maintenance of devices. To this end, the CSR architecture is an interconnect specification between a variety of back-end processing environments, front-end device user interface "shells", and the associated software mix.
- **Extensible one- and two-socket building blocks**. It is assumed that most CSR-based devices will be built around one or two CSR modules. The size of many mobile devices are strictly constrained such that they can fit easily into shirt or coat pockets. Many stationary devices are also constrained in their size such that they are not overly cumbersome or aesthetically unpleasant. However, some devices, such as set-top boxes, may require a larger number of CSR modules. Nonetheless, many implementations of the CSR architecture will emphasize and optimize for one- and two-socket CSR devices, while providing means for extending the architecture to a larger number of sockets.

- **Native support for the use of commodity components in device design**. Device design is most simple when it is reduced to selecting and interconnecting various off-the-shelf components. In order to support such a simple design flow, CSR modules can directly interface with many off-the-shelf commodity components, such as digital-to-analog converters (DACs) and RS-232 serial transceivers.

- **Ready availability of a large variety of CSR peripheral modules**. The utility of CSR-based devices is greatly improved and their lifetime extended by simply having a variety of interesting peripheral devices available as pluggable CSR modules. Examples of exciting and innovative peripheral devices that are available as pluggable modules include those found as PCMCIA PC Cards and Handspring Springboard modules.

- **High-bandwidth computer-module-to-computer-module communications**. Multiple CSR computer modules may be used within CSR-based devices to economically expand their computational resources. A high bandwidth, low latency communications path between the modules is important to efficiently realize a parallel multi-computer. If PCMCIA PC Cards are used as CSR modules, the 32-bit CardBus interface provides 132 Mbytes/second bandwidth with configurable, guaranteed latency.

- **Sufficient communications performance to support typical multimedia applications**. Devices that host network-based content and services will usually implement a multimedia user interface. These devices will provide a combination of audio, video, and tactile representation of content, and their underlying architecture must support the simultaneous bandwidth requirements for a multiplicity of modalities.

- **System symmetry**. System symmetry refers to the ability to connect both computational and peripheral resource modules to all sockets of a CSR-based device. Symmetric systems allow consumers and manufacturers to swap-in and swap-out CSR modules to any socket, unencumbered by the need to locate computer-only or peripheral-only sockets. Symmetry also enables CSR-based devices to be powered by multiple computational resource modules, with either module providing the primary hosting services.

- **Low cost and small form factor system**. A systems architecture that can be implemented with little cost and in little physical area can be used to build a broad variety of devices, from small to large and from inexpensive to expensive. Accordingly, low cost and small physical size are overriding design criteria of the CSR architecture.

The CSR implementation described herein supports all of these features. CSR currently has been used to develop a palm-sized computing device capable of real-time decoding of MP3 audio as well as supporting interactive games using the device's LCD touchscreen for input. CSR has also been used to develop a set of software-based physical interactive devices and children's toys that are capable of downloading and embodying content. Finally, CSR has been used to develop a multimedia digital picture frame capable of downloading pictures and associated audio streams over a wireless cellular network.

The remainder of this chapter describes the current implementation of the CSR architecture in more detail. The next section describes an implementation of CSR based on PCMCIA PC Cards that leverage the availability of a wide assortment of PC Card peripherals. The section thereafter discusses the implementation of the CSR Interconnection Fabric Backplane (IFB), which is essentially a datapath for connecting together various CSR modules and the network device "shell". The chapter concludes by describing the implementation of various CSR-based network devices, and gives experimental data and analysis of the CSR System.

3.2 PC Card-Based CSR Modules

PC Cards are credit-card-sized integrated circuit cards that allow upgrading and augmenting the peripheral capabilities of mobile computing environments. The Personal Computer Memory Card International Association (PCMCIA) defines and maintains the specification standard and promotes interoperability between computer manufacturers and PC Card peripheral manufacturers.

Today, many peripherals can be found packaged as PC Cards, including network interface cards, joysticks, audio and video accelerators, volatile and non-volatile memories, digital cameras, and wireless communication devices. The convenience and economic benefits of PC Cards have led to their widespread adoption with not only laptop computers supporting the standard, but also some television set-top boxes, DVD players, and digital cameras providing PC Card sockets.

The low power consumption, small credit-card-size, ruggedized packaging and popularity of PC Cards distinguish them from their peripheral brethren, and make them a good candidate on which to base the design of an implementation of the CSR System. A PC Card-based CSR System comprises the following components:

- CSR peripheral modules, which are standard, off-the-shelf PC Card peripherals;
- CSR computer modules, which are computational resources within PC Card packaging, and supporting communications with CSR peripheral modules as well as with off-the-shelf and custom components of device shells;
- CSR Interconnect Fabric Backplane, through which a variety of off-the-shelf PC Card peripherals (or, CSR peripheral modules) and one or more CSR computer modules may be electrically connected together as well as with the device shell to compose together an entire system for a network device.

The following two sub-sections review the two primary PC Card electrical interfaces specified by the PCMCIA, and discuss which peripherals use which interface and why.

3.2.1 16-Bit PC Card Interfaces

The original PC Card electrical interface is based on either an 8- or 16-bit datapath and is similar to the popular desktop ISA bus interface. PC Cards supporting this specification use a simple asynchronous hand-shaking protocol. As the 16-bit PC Card interface was the original PC Card

interface specification as well as the simplest, most off-the-shelf PC Card peripherals available today are based on this interface.

Figure 11 graphically depicts a typical 16-bit PC Card transaction cycle. The handshake protocol is segmented into three phases. The first phase is the Setup Phase, which specifies the transaction type. The second phase is the Command Phase, which actually executes the transaction. The third phase is the Hold Phase, which ensures that the receiving party of the transaction can reliably latch the data and then ends the transaction. Each three-phase transaction can only communicate a single 8- or 16-bit data packet, and the timing for each phase determines the overall bandwidth.

The primary signals involved in a 16-bit PC Card transaction are a 26-bit address (`Addr`) that specifies the desired read or write location, two Card Enable (`CE1#` and `CE2#`) signals that select the desired PC Card and also provide byte addressing information, write enable (`WE#`) that specifies whether the current transaction is a read or write transaction, output enable (`OE#`) that enables and disables data line drivers for the 8- or 16-bit datapath (`Data`).

16-bit PC Cards can transfer a maximum of two bytes per transaction, and have a minimum total transaction cycle time of 100 nanoseconds. This gives a maximum theoretical bandwidth supported by these cards to be 20 Mbytes/second.

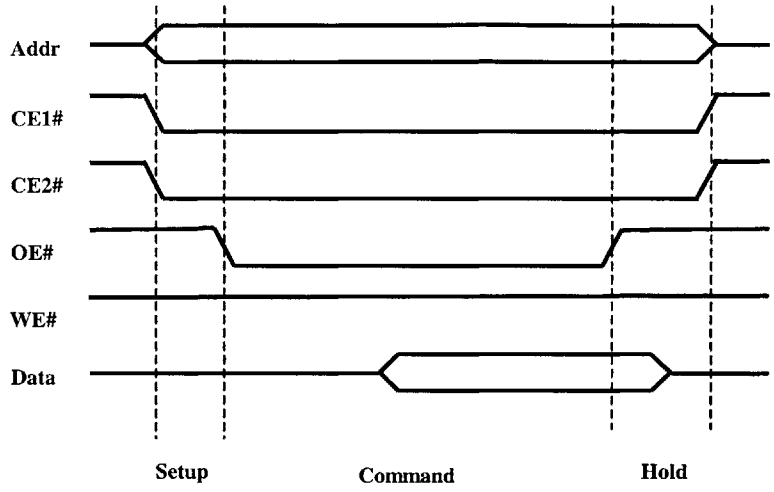


Figure 11: 16-bit PC Card electrical interface and protocol overview for a read transaction.

The simplicity of the 16-bit PC Card interface together with its reasonable bandwidth make it an excellent interface for PC Card peripherals. It is not surprising that most PC Card peripherals are in fact based on the 16-bit interface. PC Card peripherals that require additional bandwidth are based on the CardBus PC Card interface. The next section reviews the CardBus specification.

3.2.2 32-Bit CardBus Interface

CardBus PC Cards use a 32-bit, synchronous communication protocol similar to the popular Peripheral Component Interconnect (PCI) local bus interface found in most contemporary desktop computing environments. Also like PCI peripheral devices, CardBus PC Cards support bus-mastering that enables them to communicate data with lower latencies than simple slave devices that are fully dependent on their host computer system.

Figure 12 shows a typical CardBus transaction. CardBus PC Cards achieve higher bandwidth than 16-bit PC Cards by using a high-speed synchronous clock and by supporting data bursting, which reduces transaction overhead by allowing multiple data packets to be associated with a single address that is automatically incremented after each completed cycle.

The primary signals involved in a CardBus transaction are shown in Figure 12. The synchronous clock signals (CCLK) provides the central reference on which all other signals are based, and all signals are latched on the rising-edge of CCLK. The CardBus Frame (CFRAME#) signal bounds the start and end of each transaction. The first asserted cycle of CFRAME# also specifies the transaction address on the 32-bit CardBus Address and Data (CAD) lines as well as the transaction type on the 4-bit CardBus Configuration and Byte-Enable (CCBE#) lines. After the first asserted cycle of CFRAME#, the 32-bit CAD lines carry data, while the 4-bit CCBE# lines carry byte-enable information. The transaction initiator asserts CardBus Initiator Ready (CIRDY#) on each cycle on which it is driving valid data over the CAD lines. The transaction target asserts CardBus Target Ready (CTRDY#) on each cycle on which it has latched data from the CAD lines.

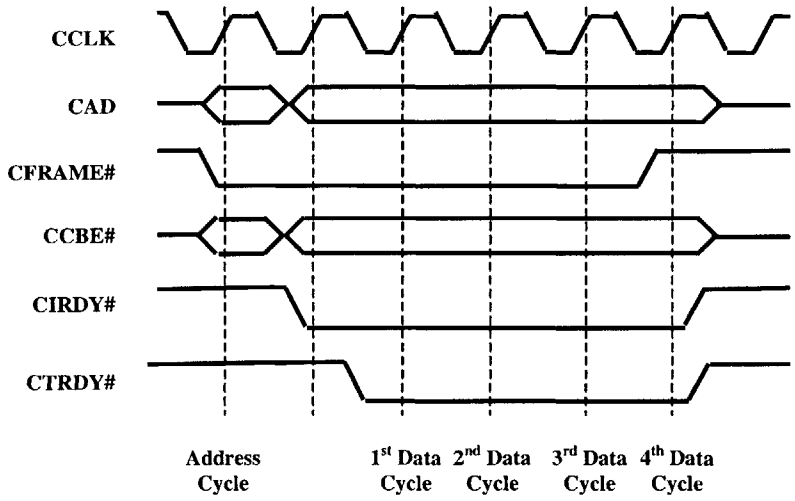


Figure 12: 32-bit CardBus transaction.

Each cycle on which both CIRDY# and CTRDY# are asserted represent a valid transfer of data between the initiator and the target. CFRAME# is deasserted one cycle before the final data transfer. CFRAME# and CIRDY# are deasserted upon the completion of the entire transaction, and signify an idle bus.

CardBus PC Cards can transfer a maximum of four bytes (32-bits) per cycle, and have a minimum cycle time of 30 nanoseconds (33MHz). This gives a maximum theoretical bandwidth supported by these cards to be 132 Mbytes/second. The higher bandwidth of CardBus PC Cards comes at a cost. These PC Cards are more difficult and expensive to design and manufacture. Accordingly, only peripherals that require the increased bandwidth or reduced latency can be found as CardBus PC Cards. The most common CardBus cards are high-speed network interfaces.

Having reviewed the two primary PC Card interfaces, the following section discusses characteristics of CSR Interconnect Fabric Backplanes (IFBs). The IFB architecture determines to a large part the architecture of CSR computer modules and affects many of its design decisions.

3.3 CSR Interconnect Fabric Backplanes

Within the current PC Card-based implementation of the CSR System, off-the-shelf PC Cards provide peripheral resources, while CSR computer modules provide computational resources. An interconnection fabric is necessary to provide electrical connectivity between the peripheral resources and the computational resources, as well as connectivity with the components of the device.

The CSR Interconnection Fabric Backplane (IFB) provides this connectivity. It is a small electrical fabric that is embedded into each device, and comprises receptacles for mateably accepting CSR modules. A typical CSR-based device may support one or two CSR modules, while some may support a larger number.

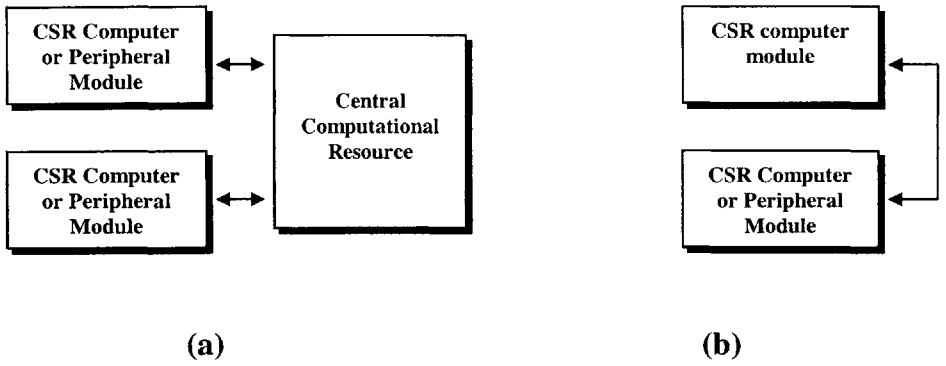


Figure 13: (a) Embedded computational resources for managing CSR Modules versus (b) CSR computer module acting as the central computational resource managing other CSR Modules.

3.3.1 Central Computational Resource

16-bit PC Cards and 32-bit CardBus cards were designed to augment the peripheral resources of mobile computing environments, e.g., laptop and notebook computers. In the CSR architecture, not only must we provide a means to easily augment and upgrade the peripheral resources of a network device, but also its computational resources. In the PC Card-based CSR System, both peripheral resources and computational resources are found within a PC Card packaging – essentially, all resources are PC Cards, with their label distinguishing the various peripherals and computational devices.

The PCMCIA PC Card Standard is based on a central computing resource around which various PC Card peripherals connect. If the CSR System follows this architecture, then CSR computer modules would simply be *peripheral* computational resources that are managed by the central computing resource. This is shown in Figure 13(a). The central computational resource may use a processor whose sole purpose is to manage attached peripheral and computational resources.

An alternative, shown in Figure 13b, is to have CSR computer modules that are both removable and also act as the central computational resource. This solution relieves devices from having any fixedly-attached computational resources that must interface with all CSR modules.

The system of Figure 13(b) requires that a CSR computer module be present to provide computational resources to the network device and a CSR peripheral module can only be used in conjunction with a CSR computer module. Conversely, the system of Figure 13(a) does not place such a restriction and CSR peripheral modules may be used singly without a CSR computer module. Both architectures may support a multiplicity of computational and peripheral resources.

The current implementation of the CSR architecture as described herein is based on the non-centralized bussed architecture of Figure 13(b).

3.3.2 Significantly Bussed Datapath

Assuming a device that supports two CSR modules, Figure 14 shows two possible system architectures for the CSR Interconnect Fabric Backplane. Figure 14(a) shows a point-to-point architecture and Figure 14(b) shows a significantly bussed architecture

Point-to-point datapaths have a single load on its signal pins, while significantly bussed datapaths comprise a majority of shared (or bussed) signals with a small number of point-to-point dedicated signals. The sharing of signals between multiple devices removes the need for an intermediate point-to-point IFB controller. This reduces the pin count of IFB controllers, the number of traces and printed circuit board (PCB) layers of IFBs, which in turn, reduces overall size and cost. A smaller and less expensive IFB is more broadly applicable to devices across a variety of market segments.

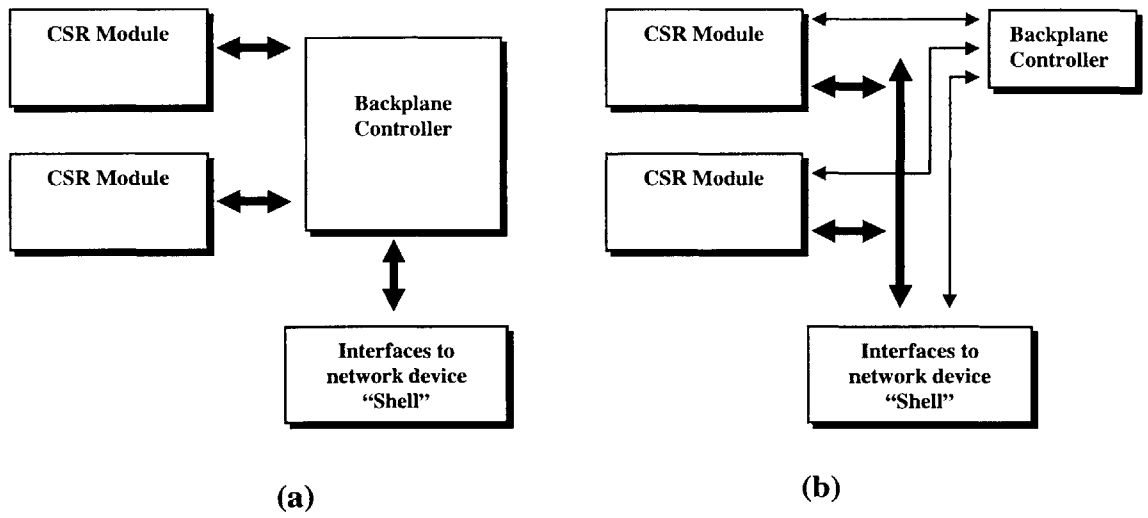


Figure 14: Two types of datapath interconnection architectures. A (a) point-to-point datapath has a single load on each of its signals, while a (b) significantly bussed datapath has multiple loads on most of its signals with just a few signals point-to-point.

A significantly bussed PC Card-based CSR architecture supports both of the primary two electrical interfaces defined by the PCMCIA for PC Cards (16-bit PC Card and 32-bit CardBus cards) and also supports additional interfaces for interconnecting with off-the-shelf integrated circuits, such as digital-to-analog and analog-to-digital converters, infrared transceivers, and light emitting diodes, that may be common and useful to many device shells. Supporting both the 32-bit CardBus interface as well as the 16-bit PC Card interface enhances system flexibility and applicability, while supporting the use of high volume, off-the-shelf integrated circuit chips for device shells keep lead times and development cycles short and production costs low.

In order to support a multiplicity of electrical interfaces over a fixed number of connector pins (e.g., the 68-pin PC Card connector), CSR computer modules and CSR Interconnect Fabric Backplanes support a time-multiplexed multibus interface. The next section describes this

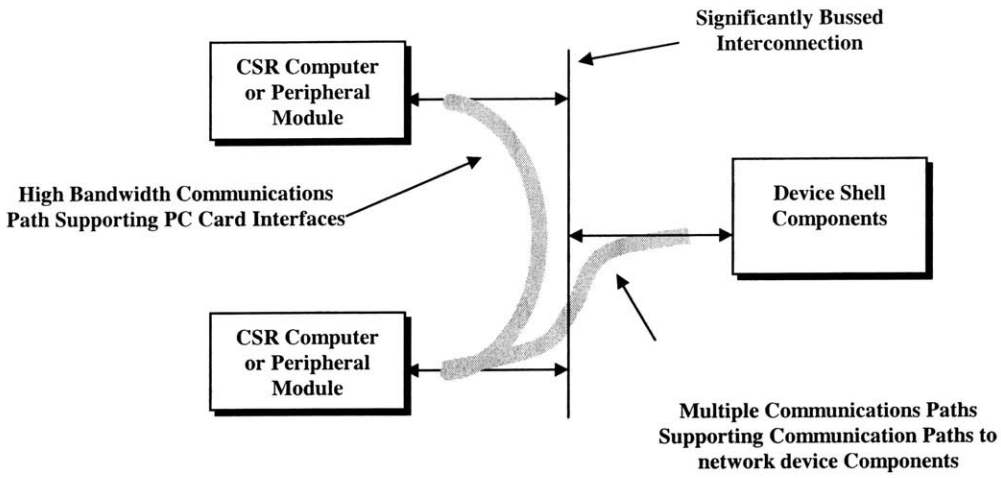


Figure 15: Multiple bus communication protocols over a single bussed environment.

multibus interface, the algorithms and mechanisms for dynamically selecting the appropriate interface, and completely specifies the system architecture for one- and two-socket CSR IFBs.

3.4 Two-Socket CSR IFB Specification

Devices based on two-socket CSR IFBs will perhaps be the most popular. These devices can accommodate two CSR modules, one of which is a CSR computer module, and the other is either a CSR computer module or a CSR peripheral module. As depicted in Figure 15, modules are interconnected, together with the components of the device shell, over a significantly bussed datapath. The CSR modules communicate with one another using a PC Card-like interface, while CSR computer modules communicate with device shell components using electrical interfaces commonly found on off-the-shelf commodity integrated circuits, such as Serial Peripheral Interface (SPI), RS-232, 8-bit parallel interfaces.

Figure 16 depicts the schematic for a two-socket CSR IFB that supports two synchronous serial interfaces (SPI interface) and one asynchronous parallel interface. Most signals are directly bussed, with only a small number of the signals interfacing with the IFB Controller. The thick line between the two CSR sockets represents signals defined by the PCMCIA Standard that are not already shown in the diagram.

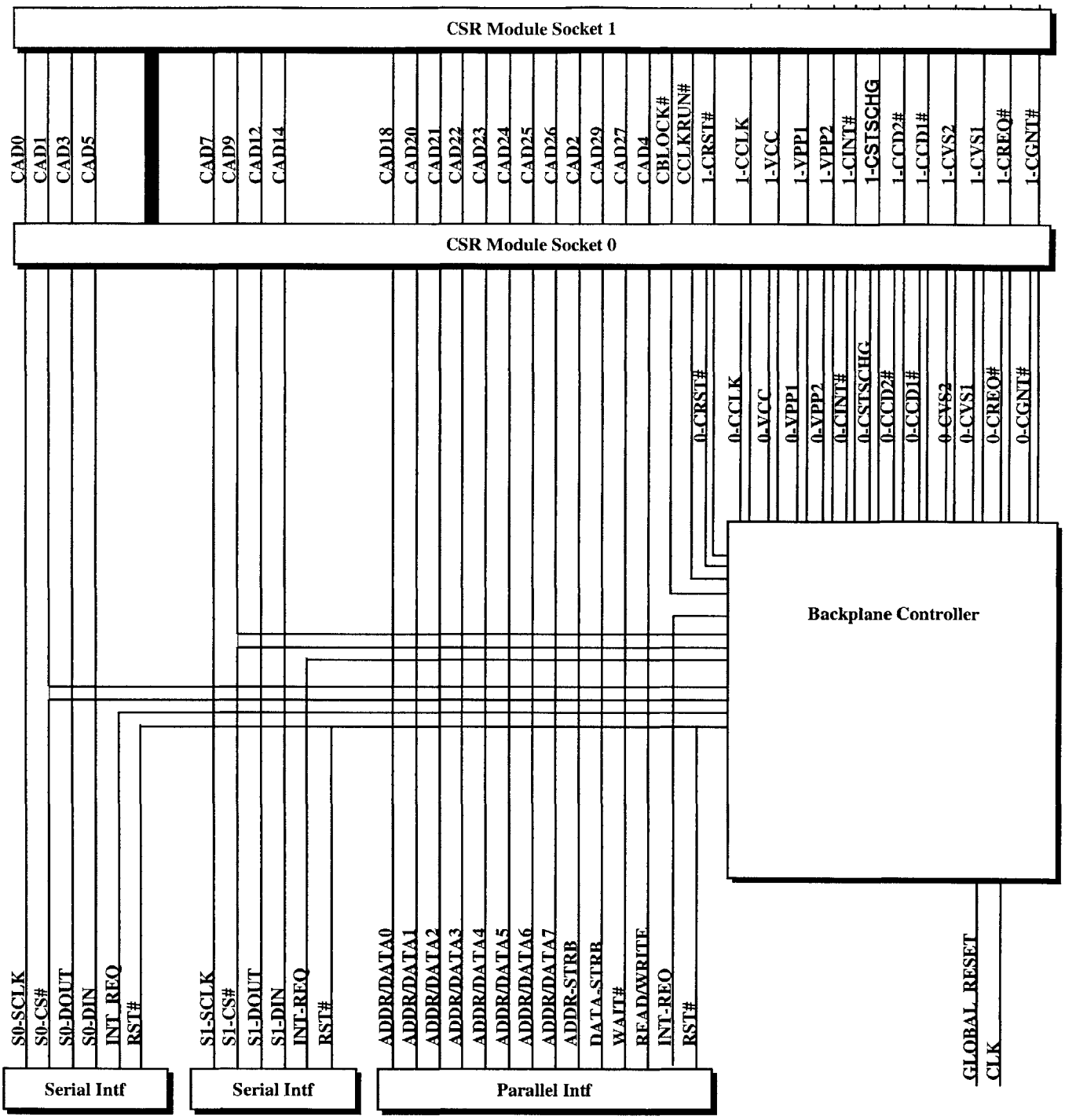


Figure 16: Two-socket PC Card CSR IFB specification.

The following sections specify the current implementation of the two-socket PC Card-based CSR IFB.

3.4.1 Static Reconfiguration For Efficient Module-to-Module Communications

CSR IFBs reconfigure themselves to support the electrical interfaces and protocols of various CSR modules. This reconfiguration is static in that it is initiated upon both sockets being occupied by CSR modules, and the reconfiguration changes only in response to new CSR modules being swapped in or out.

The current implementation supports the two primary interfaces defined by the PCMCIA standard – the 16-bit PC Card interface and the 32-bit CardBus interface. The many differences in these electrical interfaces make designing a bussed IFB difficult. Table 1 shows all possible ways the two sockets may be populated, and the communication interface that is used for module-to-module communication. The sockets are fully symmetrical in that CSR computer or peripheral modules may be connected to either socket equally well. 16-bit PC Cards and 32-bit CardBus cards are not supported in the absence of a CSR computer module, but two computer modules can be connected together to form a multi-computer.

Socket	Socket	Selected Interface[†]
Empty	Empty	None
CSR computer module	Empty	None
CSR computer module	CSR computer module	32-bit CardBus
CSR computer module	32-bit CardBus PC Card	32-bit CardBus
CSR computer module	16-bit PC Card	16-bit PC Card

Table 1: Possible different ways the sockets of a two-socket IFB can be populated, and the interface protocol that is used for communication between CSR modules connected to those sockets.

[†] The 16-bit PC Card and 32-bit CardBus protocols and electrical interfaces used by CSR computer modules and CSR IFBs are not exactly as specified by the PCMCIA. Slight variations as discussed herein are required to support the CSR architecture. However, standard off-the-shelf 16-bit PC Cards and 32-bit Cardbus cards can be used without modification.

CSR computer modules use the 32-bit CardBus interface as its default communication mode. This takes advantage of the high bandwidth and low latency nature of the CardBus interface during not only computer-module-to-CardBus-module transactions but also computer-module-to-computer-module transactions. Only if a 16-bit PC Card occupies an IFB socket does the system reconfigure itself to use the slower 16-bit asynchronous interface.

In order to reconfigure itself for different CSR modules, IFB controllers employ an interface discovery mechanism to determine the electrical interface supported by newly connected CSR modules. This interface discovery mechanism is based on the two pairs of card detect (CD1 and CD2) and voltage sense (VS1 and VS2) signals defined by the PCMCIA Standard. PCMCIA defines an interpretation of interface type and initial voltage requirement based on the state (e.g., VCC, GND, or flow-through) of these pins. The information about module interface type, in conjunction with the information in Table 1, is used to reconfigure the IFB Controller as well as CSR computer modules so as to support module-to-module communications. Unsupported (IFB specific) combinations of CSR modules (e.g., one 16-bit PC Card peripheral and one 32-bit CardBus peripheral) may be handled by simply not delivering power to the sockets.

Since most signals are bussed directly between sockets, and are not connected to the IFB Controller, the IFB must only re-configure a small number of signals. Reconfiguration of the IFB Controller involves changing from a mode that supports 32-bit CardBus transactions to one that supports 16-bit PC Card transactions. The following list enumerates the 32-bit CardBus signals that are affected during static reconfiguration.

CardBus Bus Request (CREQ) and Bus Grant (CGNT). The IFB Controller acts as the central resource arbiter for CardBus cards and CSR computer modules, and provides a simple round-robin arbitration mechanism. Both pairs of these signals are point-to-point lines between each socket and the IFB Controller. However, in the 16-bit PC Card mode, the request and grant signals become the Input Port Acknowledge (INPACK#) and Write Enable (WE#) signals, respectively. Accordingly, reconfiguration to the 16-bit PC Card mode requires that the IFB Controller make the Write Enable signal a flow-through signal, with the command flowing from

the socket occupied by the CSR computer module to the socket occupied by the 16-bit PC Card. Similarly, the Input Port Acknowledge signal is made a flow-through signal, with the command flowing from the socket occupied by the 16-bit PC Card to the socket occupied by the CSR computer module. Routing CREQ and CGNT through the IFB controller prevents false module-to-module transactions from being initiated during CSR module removal due to electrical signal disruptions over the bussed IFB. Other implementations may choose to directly bus CREQ and CGNT between the two CSR sockets with the leader computer module providing system arbitration.

CardBus Synchronous Clock (CCLK). The CardBus Clock signal is the clock around which all interface transactions are synchronized. CCLK is provided by the IFB Controller as two separate point-to-point signals to each socket. However, in 16-bit PC Card mode which is asynchronous, the signal becomes the 16^{th} bit of the address (ADDR16). During reconfiguration, then when the IFB Controller configures the signal as a flow-through lines, with data flowing from the socket occupied by the CSR computer module to the socket occupied by the 16-bit PC Card.

CardBus Clock Run (CCLKRUN). The CardBus Clock Run signal is used to implement a low power mode that can slow or completely stop the system clock (CCLK). In 16-bit PC Card mode, this line becomes the Write Protect (WP) signal. During reconfiguration, the IFB Controller configures the signal to be a flow-through line, with data flowing from the socket occupied by the 16-bit PC Card to the socket occupied by the CSR computer module.

Card Reset (CRST). Card Reset is used to globally reset both 16-bit PC Cards as well as 32-bit CardBus cards. The polarity of the signal differs with each type of card, and the IFB Controller must be configured to deliver the appropriate polarity reset signal. 16-bit PC Cards require an active-high reset signal, while 32-bit CardBus cards require an active-low signal.

Essentially, the IFB Controller is configured such that the bussing of signal lines between the two sockets is completely transparent to off-the-shelf 16-bit PC Cards and 32-bit CardBus cards.

3.4.2 Dynamic Reconfiguration for Narrowpath Communications and Target Disambiguation

Narrow datapath communication channels (hereinafter narrowpaths), such as serial and 8-bit parallel channels, require just a few signal lines and support interfacing with off-the-shelf I/O components. Most off-the-shelf commodity components, such as digital-to-analog converters (DACs), analog-to-digital converters (ADCs), and codecs, use narrowpath interfaces to reduce pin count and overall integrated circuit (IC) package size and cost.

CSR computer modules can directly interface with off-the-shelf ICs and components using narrowpath interfaces, thereby eliminating the additional cost and size of glue-logic and other bridging components. Allowing CSR computer modules to directly communicate with off-the-shelf narrowpath components also reduces design time, design complexity, and errors borne out of complexity.

Referring again to Figure 16, narrowpath channels on CSR IFBs may include serial interfaces such as Serial Peripheral Interface (SPI), Microwire, and RS-232, parallel interfaces, such as 8-bit micro-controller interfaces and Enhanced Parallel Port (EPP), and single bit-toggling interfaces. Each narrowpath channel is completely independent, and can execute asynchronous with neighboring narrowpath channels.

In order to support both wide datapath PC Card interfaces as well as narrowpath interfaces over a single bussed medium, CSR IFBs use a target disambiguation mechanism to clearly identify the target of each transaction. This disambiguation mechanism effectively reconfigures the IFB, allowing communication to proceed between the appropriate source and target agents. The current implementation of IFB Controller uses a set of chip enable signals for enabling and disabling narrowpath channels as well as enabling and disabling module-to-module communications.

CSR computer modules that are connected to multi-socket, bussed IFBs cannot simply use the shared bussed lines as narrowpath chip selects as these lines are used for module-to-module communications as well. In the current implementation of PC Card-based CSRs, the leader

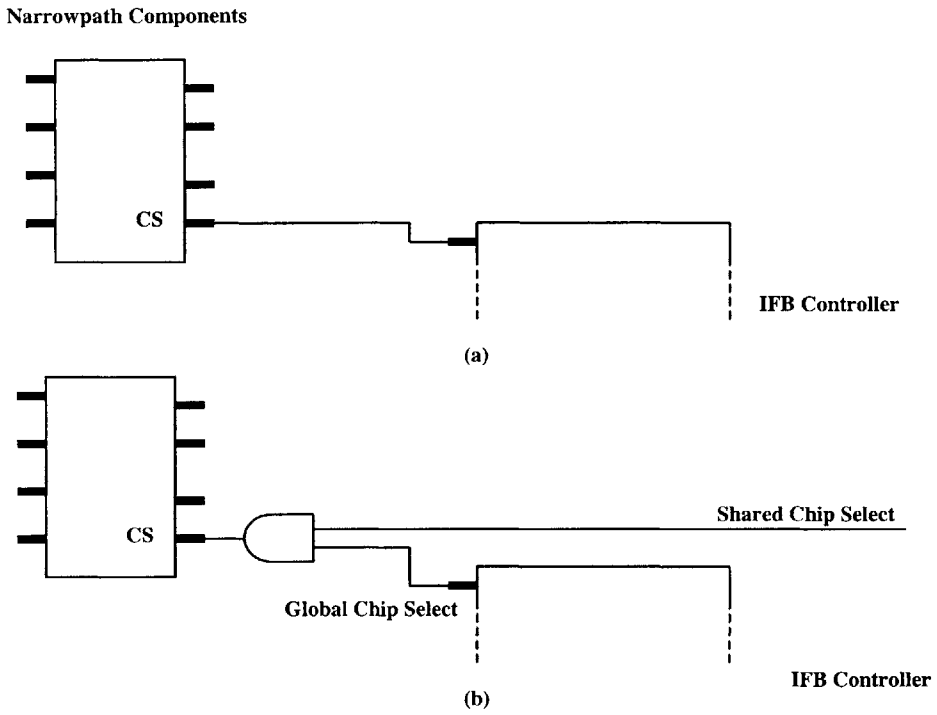


Figure 17: Target disambiguation mechanism implemented through (a) a dedicated chip select signal, and (b) a global chip select signal ANDed together with a shared chip select..

computer module asserts the appropriate chip select signals by first communicating with IFB Controller. In turn, IFB Controller delivers the chip select signals to the appropriate narrowpath components. Once the computer module has finished its transactions, it ceases communications over the bussed signal lines, and instructs the IFB Controller to de-assert the narrowpath chip enable signals.

This chip enable mechanism for narrowpath transactions may be implemented using a variety of means. Figure 17 depicts schematically two such mechanisms. Figure 17(a) shows the use of point-to-point chip enable signals between the IFB Controller and each narrowpath component. With such an implementation, the leader computer module communicates with the IFB Controller using the event messaging subsystem (as described in Section 3.4.5) and instructs it to assert or de-assert the chip enable signals for the appropriate narrowpath channels. Conversely,

Figure 17(b) shows the use of a global chip select signal in conjunction with individual bussed chip enables. In this mechanism, using the event messaging subsystem, the leader computer module instructs the IFB Controller to assert the global chip select signal, while the individual narrowpath channels assert their own chip select signals. Assuming an active-high logic level for each of the chip select signals, a logical AND gate is used to deliver the actual chip select to each of the narrowpath components. The need for the discrete logical AND gate can be eliminated by capturing the functionality within the IFB Controller. That is, by making each of the shared chip select signals point-to-point with the IFB Controller and moving the logical AND gate and the global chip select into the IFB Controller, a single chip select signal can be delivered to each narrowpath component.

3.4.3 CSR Computer Module Leader Election

The two-socket CSR IFB described herein supports up to two CSR computer modules. In order to facilitate configuring the entire system, the IFB Controller selects a leader computer module to set up the device shell and the other CSR module upon startup, as well as to maintain system integrity during normal operations.

Leader computer modules have expanded responsibilities, and enjoy enhanced benefits. The leader computer module initializes the other CSR module by partitioning the IFB address space and programming its hardware to respond to the allocated address partitions. The leader module also reads information about the device shell through the IFB Controller. This information allows the leader computer module to configure its own hardware interface as well as its software structures to communicate with the various components of the device shell. The communication mechanism between leader computer modules and IFB Controllers is described in the next section. The leader computer modules also communicate with and transfer data between narrowpath components.

The actual election process can be a simple one based on geographical position on the IFB. One implementation of the leader election process is as follows:

- If there is just one CSR computer module connected to the IFB, then that computer module is the leader.
- If there are two CSR computer modules connected to the IFB, then the computer module occupying the lowest numbered socket is the leader.

The IFB Controller-selected leader computer module may communicate with the IFB Controller to request that leadership be passed to the other computer module. This may be useful if the other computer module is in fact a more resource-rich module, and can more effectively undertake the leadership responsibilities.

Migrating leadership status can also be used to improve system performance. For instance, consider a two-socket IFB that is populated with two CSR computer modules. The first module is optimized for running complex security applications with minimum power consumption, while the second module is optimized for graphics-intensive applications (e.g., games). Both modules have provisions for handling leader module duties. As the mix of applications that are executed by the device migrate from graphics-intensive to security-intensive, migrating leadership from the graphics module to the security module can also improve performance and lower system power consumption. This is primarily because the leader computer module communicates with and transfers data between the narrowpath components (this is a restriction of the current implementation that simplifies system arbiter design). Transferring leadership status to the module that most frequently interacts with narrowpath components reduces unnecessary data copies, whereby improving performance and lower power consumption.

3.4.4 System ROM Interface

Since CSR modules are fully generic, they require a means to understand the device-specific implementation details of each device to which they are connected. For instance, before a CSR computer module connected to an CSR-based audio device can properly function, the module must first understand how to communicate with the device shell's digital-to-analog converter and the actual format and bit resolution of the data to be delivered to the DAC.

Such device shell-specific information is stored in a location accessible by the leader CSR computer module. This information may include the number of CSR modules supported by the IFB, the number and type of components connected to the narrowpath interfaces, the capabilities of the IFB Controller, and information relevant for software environments such as key device drivers. This information may be stored locally, or the local storage may simply be an identifier that can be de-referenced to locate the relevant information. One common possibility is a network-based database that can be located and indexed using such an identifier.

3.4.5 Interrupt and Event Messaging

The interrupt and event handling mechanisms supported by the CSR architecture provide means for asynchronous messaging between CSR modules as well as between CSR computer modules and IFBs (narrowpath components and IFB controller). Messaging needs between CSR modules include:

- Interrupts defined by the PCMCIA and used by off-the-shelf PC Cards
- Synchronization between multiple CSR computer modules

Messaging needs between CSR computer modules and IFBs include:

- Transferring information between IFB Controllers and CSR computer modules. This information may include system ROM information, announcements by modules that it is in fact a CSR computer module, computer module leader election results, new module connect/disconnect signals and configuration information, as well as initiation signal for dynamic or static reconfiguration.
- Interrupt requests from device shell components using narrowpath interfaces.

In order to limit the pin count of IFB Controllers, interrupt and event messaging between IFB controllers and each CSR module socket is implemented by using a serial protocol. This RS-232 serial protocol is layered on top of the status change and interrupts request pins of CSR computer modules. These pins function as specified by the PCMCIA for off-the-shelf 32-bit CardBus cards

and 16-bit PC Cards, but are seamlessly and automatically redefined for CSR computer modules when inserted into a CSR IFB.

The interrupt request and the status change lines are specified as output pins (from the perspective of PC Cards) by the PCMCIA. CSR computer modules may be implemented so as to use one of the pins (e.g., the interrupt request pin) as an input signal and the other pin (e.g., the status change pin) as an output signal. Then, a serial communication protocol may be layered on top of these two pins to achieve a two-way communication mechanism. These communication channels may carry encoded interrupt and event messages.

With point-to-point lines between the IFB controller and each socket's interrupt request and status change notification signals, standard 16-bit PC Cards and standard 32-bit CardBus cards are able to use interrupt requests and status change notifications as specified by the PCMCIA. Since IFB controllers track which sockets are occupied by standard 16-bit PC Cards, 32-bit CardBus cards, and CSR computer modules, it is able to selectively communicate interrupt and event messages with CSR computer modules.

IFBs may also support interrupt requests from device shell ICs and components. Narrowpath interrupt request are point-to-point signals between IFB controllers and narrowpath components. Level or pulse interrupts may be supported by IFB controllers, which are then encoded and redirected to CSR computer modules.

Different interrupt and event messages may be encoded differently over the serial communication channel. High priority communications, such as interrupt requests, may need to be short and support low-latency message delivery. Lower priority messages, such as leader election results, do not have stringent timing requirements, and may be delivered using a slower encoding. For instance, the first three bits of an eight-bit packet, may encode up to eight high priority event types, with the following five bits conveying information specific to that high priority event. If the first three bits of the packet of a communication are not asserted, then the latter 5 bits may convey the command. The command encoded within the latter 5 bits of the first packet may be followed by more packets that contain data related to that command. This is more appropriate for slower and longer messages.

3.4.6 Connection and Disconnection of CSR Modules

Connection and disconnection of CSR modules over the bussed IFB environment may cause electrical disruption, resulting in potential data corruption or false transactions.

PC Cards implement different pin lengths for power and ground as well as certain card detect pins. These pin length differences can be leveraged to defined and build an early warning system for bussed IFBs. Card detect pins are the shortest pins, and PC Cards make contact with these pins last, while power and ground pins are the longest and make contact first. The majority of signal pins are medium length pins.

CSR IFBs may define and use card detect pins to provide an early warning of CSR module connection into an IFB socket. IFB controllers may determine that a CSR module is being connected by using a few of the ground pins as input signals. This mechanism assumes that all ground pins on the PC Card are connected to a single ground plane (or at least are connected together). The IFB socket may drive most of the ground pins to ground. However, for a few ground pins (say 2 pins, on extreme ends of the connector), the PC Card socket may pull-up (with a resistor) the pins to Vcc. When these two ground pins are detected as ground (low), the IFB may assume that a PC Card is being connected to the PC Card socket.

Once the PC Card has been fully seated (e.g., the card detect lines show that the PC Card has been fully seated into its socket), the IFB may then again drive the card connect pins to ground (e.g., not using them as inputs and not pulling the lines up to Vcc). When the IFB detects that the card is being removed (e.g., by monitoring the card detect pins), the IFB may wish to again enable the card connect pins.

In CSR IFBs, when a PC Card is being connected (e.g., card connection event detected) the IFB may notify CSR computer modules that there will shortly be electrical disturbances on the bussed signal lines. This may allow CSR computer modules to halt any new transactions from starting over the bussed signal lines.

If the IFB is in the 32-bit CardBus mode, the IFB controller may simply remove bus grant from all modules, effectively halting all communications within a predetermined number of cycles. The bus arbiter may again be enabled once the electrical disturbances have subsided.

It is important to note that during the electrical disturbances associated with the connection and disconnection of CSR modules over the bussed lines of the IFB, important signal lines may momentarily glitch, causing false transactions to be signaled. Accordingly, CSR computer modules ignore key transaction control lines (e.g., CFRAME# in CardBus mode and Card Enable 1 and 2 in 16-bit PC Card mode) while a module connection or disconnection event is happening.

3.5 One-Socket CSR IFB Specifications

One-socket CSR IFBs that support a single CSR module are similar to two-socket IFBs with just a single computer module connected. In this case, the single CSR computer module only uses the narrowpath interface channels to communicate with the components of the device shell.

Figure 18 depicts the schematic of a one-socket CSR IFB that supports two synchronous serial and one asynchronous parallel narrowpath interface. Since there is only one CSR module socket and there is no need to support communication between 16-bit PC Cards or 32-bit CardBus cards, many signal lines may be eliminated from interfacing with the IFB Controller.

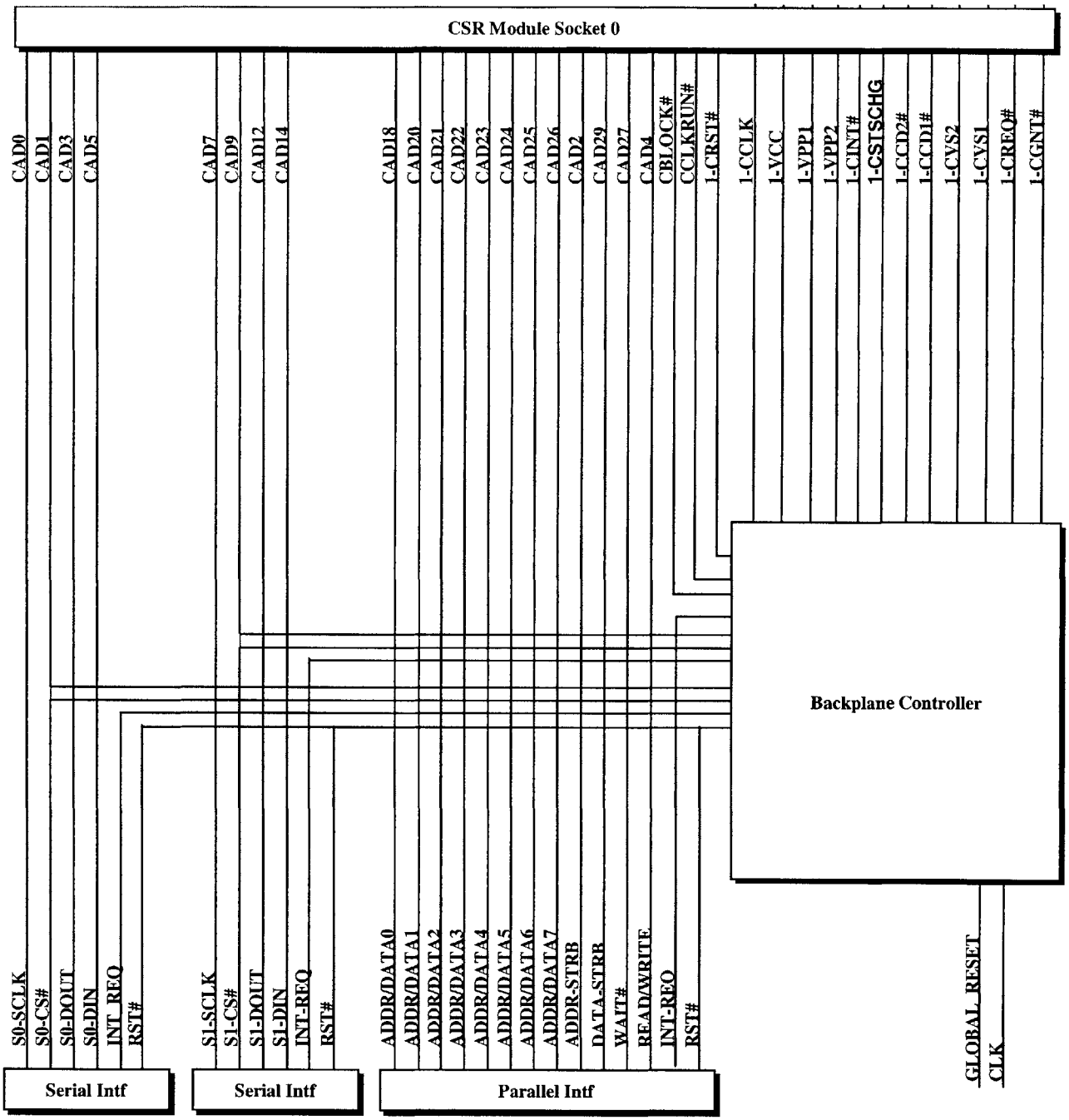


Figure 18: One-Socket CSR Interconnect Fabric Backplane.

3.6 System Extensibility

Presumably, one- and two-socket IFBs will be most popular for building a variety of mobile and small home and office devices. There are, however, instances where a larger number of sockets may be desirable. More sockets provide more flexibility for using various peripherals and additional computational resources, and are attractive for use in larger devices, such as set-top boxes. This section describes two methods for supporting a larger number of CSR modules in devices. The first method is based on linking together one- and two-socket bussed IFBs. The second is based on a switched fabric IFB design that inherently scales to support a large number of CSR modules.

3.6.1 Bussed IFB Extensibility

Capacitive loading limitations prevent the one- and two-socket bussed PC Card-based CSR IFB architecture from scaling in a straightforward manner. However, one reasonable means of scaling the architecture is through the narrowpath interfaces. Buffers using narrowpath interfaces may be used to synchronize and to communicate data between multiple one- and two-socket CSR IFBs. This system architecture is shown in Figure 19. Different supported narrowpath interfaces and groups thereof (based on required bandwidth) may be used for linking multiple IFBs together, and a variety of linking topologies may be used, such as a linear cascade or a star topology. Communication packets destined for a linked IFB are directed to the appropriate narrowpath ports, where the packets are buffered and then re-transmitted to its intended target.

Consider building a four-socket CSR IFB based on interconnecting a pair of two-socket IFBs. If the buffers used support a high-speed 10 MHz SPI interfaces with 18- and 16-bit total packet size and payload, respectively, the maximum theoretical payload bandwidth between the linked IFBs is:

$$\frac{\frac{10^9\,\text{ns}}{\frac{1}{10\,\text{MHz}}\times 18\,\text{bits}}\times 16\,\text{bits}\times 2\,\text{Ports}}{8\,\text{bits per byte}} = 2.22\,\text{Mbytes per second}$$

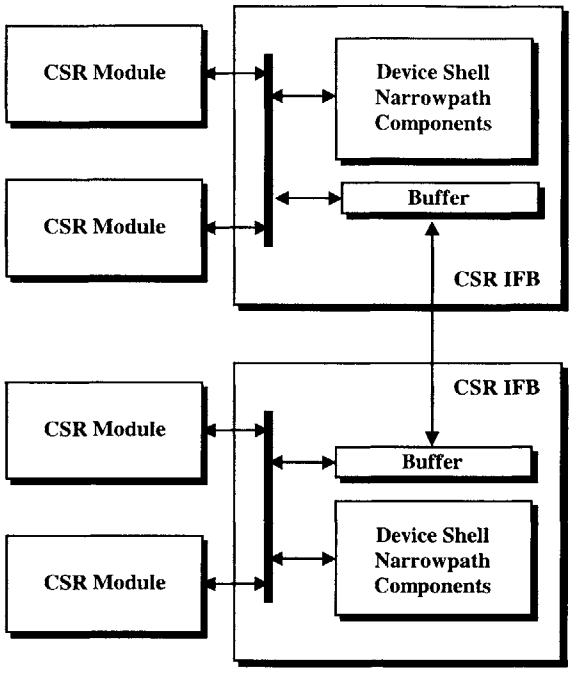


Figure 19: CSR architecture scalability based on coupling a pair of two-socket CSR IFBs together.

This, however, assumes that the narrowpath interfaces fully monopolize the IFB bus. If we assume the narrowpath interfaces control the IFB bus 33% of the time, the maximum theoretical bandwidth is simply one-third, or 741 Kbytes per second. Using a number of parallel narrowpath interfaces, increasing the clock frequency, or a combination thereof can achieve larger bandwidths.

3.6.2 Switched Fabric IFB

A circuit switched IFB can be used to interconnect a large number of CSR modules together over a single fabric. Figure 20 depicts such an architecture. The IFB implements an n x n socket-to-socket circuit switched connection that is very similar to the two-socket bussed IFB depicted in Figure 16 and described herein. The point-to-point signals between each socket and the IFB

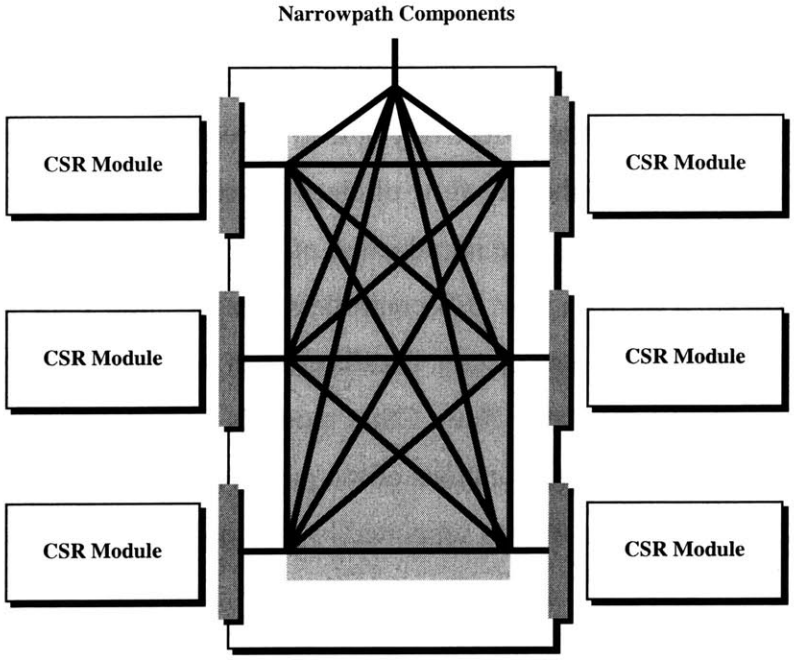


Figure 20: A circuit switched IFB supports communication between a large number of CSR modules.

controller that are shown in Figure 16 remain as point-to-point signals for each of the *n*-sockets of the circuit switched IFB. The bussed signals between the two sockets shown in Figure 16 are socket-to-socket signals in the circuit switched IFB. The switching of circuits between sockets can be changed dynamically on each transaction, or it can be set statically once the parties (e.g., CSR modules) of the communication have been connected to the IFB.

CSR modules also communicate with narrowpath components in a similar fashion. Typically, the leader CSR computer module communicates with other CSR modules as well as with the narrowpath components. Accordingly, the IFB switches signals between CSR module targets and narrowpath component targets.

3.6.3 Interface Extensibility

Bussed IFBs and circuit switched IFBs that re-drive signals from one CSR module to another, unimpeded and uninterrupted provide a means to change the electrical interface and protocol of source and target CSR modules. These effectively bussed traces taken together can be thought of as a passive backplane that allows the free flow of electrical signals between two modules. The free flow of electrical signals supports the introduction of new interfaces between CSR modules, without having to retrofit IFBs with new IFB controllers. That is, new CSR modules, possibly with more efficient protocols or proprietary interfaces, can be immediately introduced into already-deployed CSR-based devices. New CSR modules and their associated electrical interfaces need only adhere to the specifications of the point-to-point lines that connect with the IFB Controller and the total number of pins supported by the connector.

3.7 PC Card-Based CSR Computer Module Specification

PC Card-based CSR computer modules provide host- or peer-computational resources within PC Card packaging. The computational resources of these modules may be based on general-purpose microprocessors and digital signal processors (DSPs), reconfigurable processors, or application-

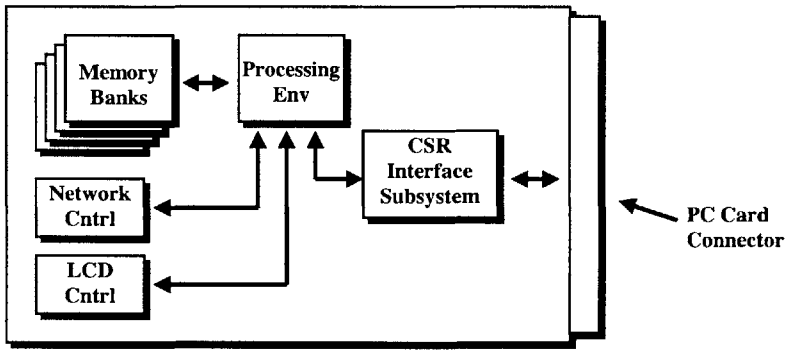


Figure 21: High-level block diagram of an implementation of the PC Card-based CSR computer module.

specific integrated circuits (ASICs). Each module may further comprise various types of volatile memory and non-volatile storage, and networking capabilities. A high-level block diagram of a PC Card-based CSR computer module is shown in Figure 21.

The interface to the components of a CSR computer module is through the CSR interface subsystem. The subsystem receives and delivers communication packets to their intended target in the appropriate format.

3.7.1 Processing Environment

Any processing environment can be used for the back end system of CSR computer modules. The CSR architecture simply defines and specifies the interface. A family of CSR computer modules that are based on a variety of backend environments can be used to configure devices for optimum performance with respect to user preferences.

[19] predicts that beyond the year 2000, 90 percent of computer cycles will be spent on multimedia applications. The processing resources for these applications can be delivered through general-purpose microprocessors, such as Intel Pentiums [31] and StrongARMs [30] or Transmeta Crusoe [18, 44] processors. However, application-specific processors can yield performance necessary for multimedia applications more efficiently with respect to performance,

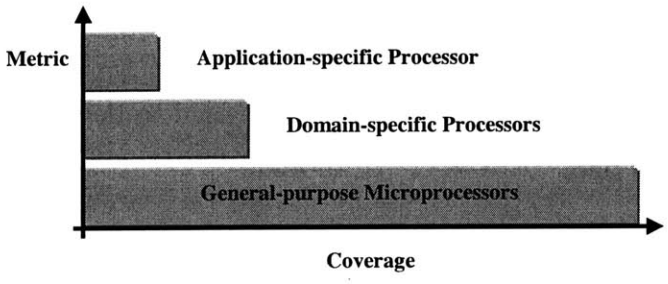


Figure 22: Spectrum of computational resource module processor architectures.

energy consumption, and physical area. These custom processors come at the price of less flexibility. Figure 22 graphically depicts energy efficiency of different classes of processor architectures, and the applicability of those architectures to a wide variety of applications [1]. The figure shows that dedicated ASICs provide the best energy efficiency (and therefore, battery life) but have little coverage. Metrics other than energy efficiency can also be used. These include security, amount of memory available for caching network content, size and weight, and interactive performance. Based on individual preferences (which can change over time), users can use CSR computer modules with the back-end processing environment that meets their needs.

Multiple CSR computer modules connected over an IFB can also be used to coordinate applications running on distinct operating systems. Consider a CSR computer module based on an x86 processor running a Microsoft Windows environment, while another module is based on a PowerPC processor running MacOS. These two CSR modules can be connected over an IFB such that the device runs software and services from both environments. Since the CSR architecture is based on data communication and it specifies packets at the protocol level, operating system and application differences (e.g., between Windows and MacOS) are transparent and seamlessly eliminated before they are delivered to the device shell.

3.7.2 Real-time Narrowpath Channels

CSR computer modules support direct interfaces with common off-the-shelf commodity components through standard interfaces, such as Serial Peripheral Interface (SPI), Microwire, RS-232, 8-bit parallel, and Enhanced Parallel Port (EPP). Allowing computer modules to directly interface with off-the-shelf components obviates the need to develop glue-logic circuitry, and eliminates their cost and size.

The most inexpensive of these components are simple resources, such as DACs and ADCs that perform a function as directed by a controller. Some DACs convert the digital value presented to them into its equivalent analog value. These components are typically not intelligent and do not support a notion of time, which is oftentimes important to properly using them.

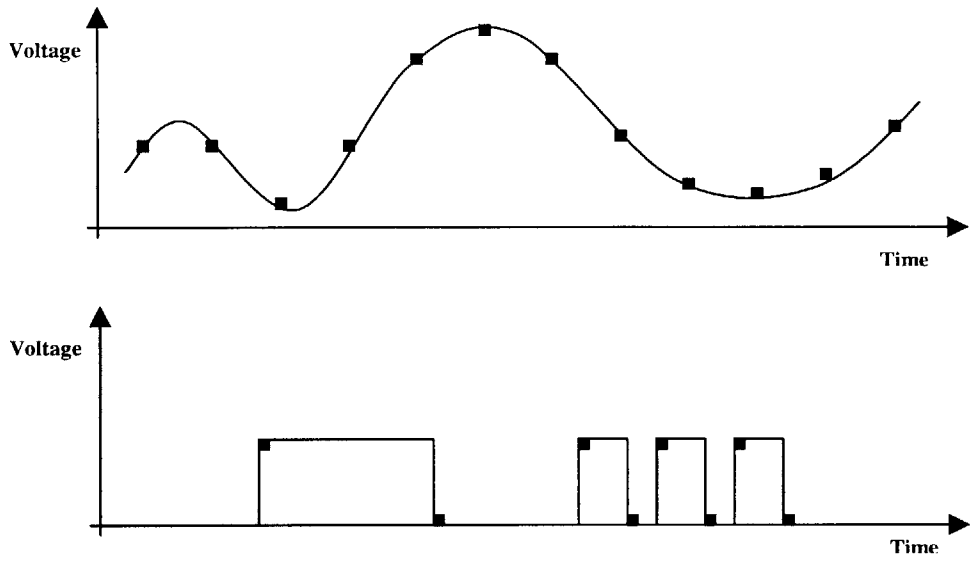


Figure 23: (a) Writing data samples using a digital-to-analog converter at constant predetermined time intervals to reconstruct an original audio waveform. (b) Writing data samples to a digital-to-analog converter at precise variable time intervals to create a time-division multiplexed signal.

Consider, for example, the DAC that is used to create the analog waveforms shown in Figure 23. Figure 23a depicts digital values are converted at equal time increments into an analog waveform that faithfully reconstructs an original audio waveform, while Figure 23b shows how digital data converted into analog signals at precise varying time intervals can be used to create time-division multiplexed (TDM) signal. The TDM signal can be used to convey encoded messages to a receiver, or to activate and de-activate LEDs, motor drivers, or switches.

Reading or writing data packets at particular time instances can be accomplished using a variety of mechanisms, including software timers, interrupts from real-time clocks, and dedicated peripheral controllers. Real-time operating systems notwithstanding, neither software timers nor interrupts from real-time clocks can offer fine granularity over the timing. Instead, dedicated peripheral controllers are typically used to precisely time communications between a microprocessor and a peripheral circuit. Peripheral controllers usually also contain data buffers

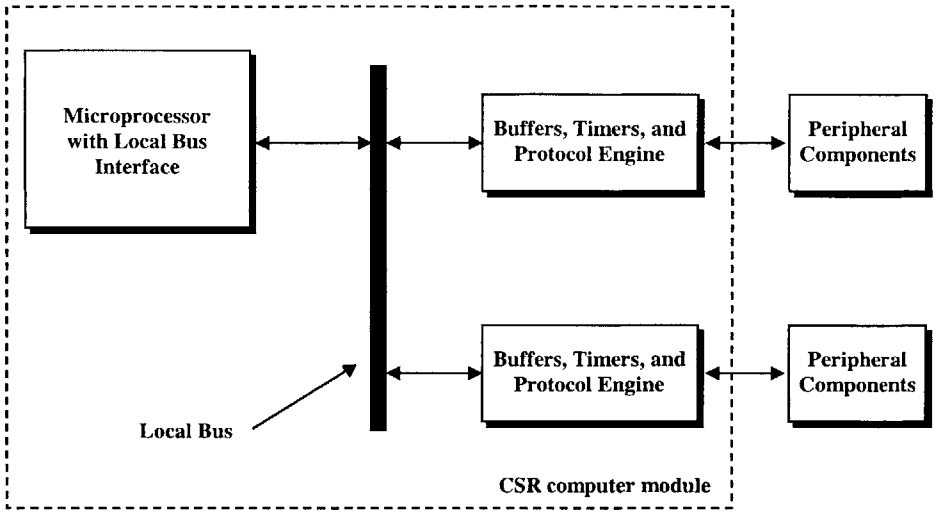


Figure 24: CSR device architecture using simple resources.

so that the microprocessor, which may be running at a much faster speed, can efficiently transfer large blocks of data to and from the peripheral circuit.

Although these dedicated controllers are relatively simple, they do consume expensive real estate on a printed circuit board and add cost to device shells. Ideally, device shells should comprise only simple and inexpensive resources, and should not require the use or development of custom buffers and timer circuits. These should be available through the CSR computer modules, and configurable to meet the needs of the particular device shell. Figure 24 graphically depicts this architecture.

CSR computer modules provide mechanisms for buffering communications between the backend processor environment and the device shell components, and also provide means for flow control (or, packet rate control) so that direct electrical connection may be achieved between the removable CSR computer modules and the components of the device shell.

The primary challenge in developing the buffering and flow control mechanisms within CSR computer modules stems from each CSR computer module's use of a multibus time-multiplexed electrical interface. These mechanisms must provide guarantees that real-time deadlines are met

even with the use of a time-multiplexed interface. To this end, we first discuss the narrowpath channels in more detail, and, in the next section, describe the facilities developed to address the issues arising from the time-multiplexed interface.

Typical integrated peripheral controllers, such as those supported by the StrongARM, provide mechanisms to delay each packet from the one preceding it. The SPI interface supported by the StrongARM SA-1100 allows software to specify the serial clock frequency so that the start of each packet is appropriately aligned with its real-time constraints. That is, the *i*th packet is stretched sufficiently so that the (*i*+1)th packet meets its timing. This approach is inappropriate for CSR computer modules because time-slots that are unused by narrowpath channels are used for PC Card transactions. Accordingly, narrowpath transaction times must be reduced to as small as possible.

The mechanisms provided by the StrongARM also do not support dynamically changing time-delays. Consider a device shell built using a quad digital-to-analog converter, which comprises four digital-to-analog converters in a single package with a single back-end electrical interface. Quad DACs are inexpensive and use less PCB area than four individual DACs. Each of the four analog outputs may be used to generate four unique and different waveforms. Depending on the actual waveform mix, the data presented to the quad DAC may not have a fixed frequency, but rather the frequency may change over time as needed to generate the desired waveforms. For instance, the time-delay between the first packet and the second packet transmitted to the quad DAC may be 20 microseconds, but the time-delay between the second packet and the third packet may only be 11 microseconds.[‡] Providing only the simple means of adjusting the serial clock speed of the SPI channel to meet real-time constraints would not support the use of quad DACs.

CSR computer modules provide more robust mechanisms for supporting direct real-time communications with off-the-shelf components. These mechanisms are centered on the use of

[‡] Time-delay refers to the delay in units of time between the start of the *i*th packet and the start of the *(i+1)*th packet. The actual time to transmit the bits is not considered. An alternate means of specifying time-delay is to use the time gap between the end of the *i*th packet and the start of the *(i+1)*th packet.

delay packets in addition to normal data packets. Through the use of delay packets, the processor environment can explicitly and dynamically specify the rate or time-delay between subsequent communication packets. For example, a data packet followed by a delay packet of ten microseconds followed by another data packet instructs the narrowpath channel to start to output the second data packet ten microseconds after the start of the first data packet. This way, the microprocessor can transfer in blocks large numbers of packets to be output by each narrowpath channel.

Reading and writing of data packets are handled similarly. Reads first require a read-request to be generated by the processor, which is eventually filled with valid data from the narrowpath component, and returned to the processor. The actual read data is returned to the microprocessor through an inbound buffer. Once data is available, the inbound buffer interrupts the processor and requests that the data be copied and purged from the buffer.

In order to maintain system timing, inbound buffers use time- and space-based watermark mechanisms. As inbound (to be delivered to the processor) data packets are accumulated in the inbound buffer and the buffer reaches a particular threshold, an interrupt is generated to avoid overflowing the circular buffer. Space-based watermark mechanisms rely on backend "push" to deliver packets on time. The sporadic nature of interactive devices may not always provide sufficient "push". A time-based watermark mechanism generates an interrupt a fixed amount of time after the entry of the first packet. In order to reduce the number of time-based interrupts generated, both the waiting period and the threshold (number of entries) can be specified and configured based on the application mix. A time-based watermark mechanism, used in conjunction with a space-based watermark mechanism, guarantees the timely delivery of temporally sensitive communication packets to their intended target.

Although the use of delay packets provides flexibility, they nonetheless represent an overhead – from the additional packets transferred from the processor to each narrowpath channel buffer and from the use of limited buffer entries. Waveforms, such as audio signals, use a set frequency or time-delay, which can be leveraged to minimize the number of delay packet used and their

associated overhead. This is accomplished by simply programming each narrowpath channel with the fixed amount of time to delay each packet from the one previous to it.

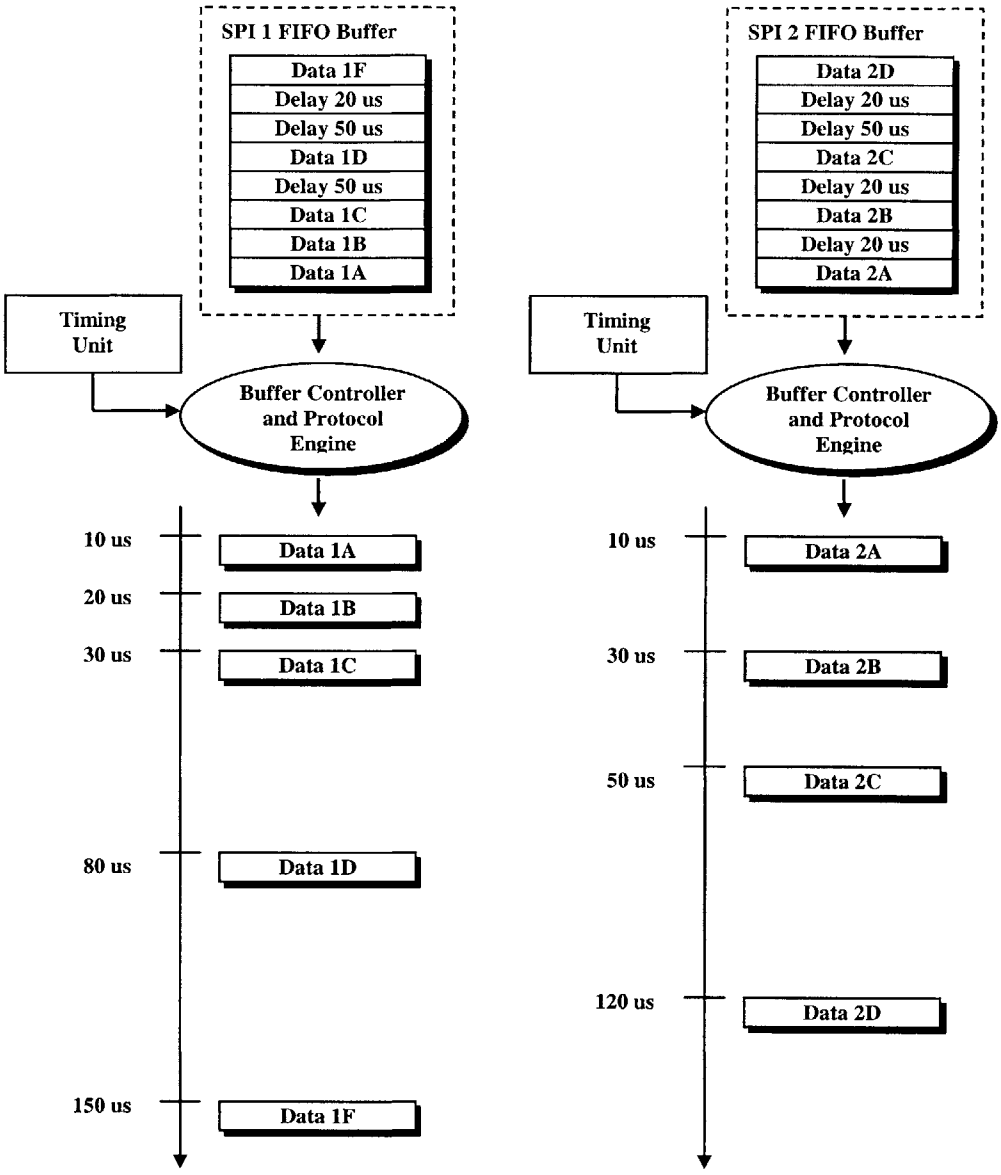


Figure 25: Mechanisms for fine granularity control over communications.

Delay packets and pre-programmed time delays can be used together to build a more robust and flexible system. In this scheme, pre-programmed time-delays reduce the number of delay packets that are sent from the microprocessor, while the occasional use of delay packets allow introducing a longer or shorter delay between two subsequent packets. The appearance of a delay packet in a narrowpath channel nullifies the pre-programmed time delay between those two

packets. Figure 25 depicts the use of delay packets and pre-preprogrammed delays for two SPI narrowpath channels. Each channel has a pre-programmed delay of 10 microseconds (us) with individual inter-packet delays overriding this default.

3.7.3 Multibus Interface Configuration and Bus Selection

Each CSR computer module supports the use of a variety of bus interfaces and protocols. These interfaces may include standard interfaces, such as SPI or EPP, or they may be proprietary interfaces. Figure 26 shows a schematic representation for the internal architecture for CSR modules and depicts how a multiplicity of bus interfaces is supported.

Each of the bus interface blocks is fully self-sufficient. They have the ability to communicate data and commands between it and the processor environment, and also comprise the buffers and protocol engine required to properly format and communicate packets to and from device shell components.

The interface controller block is perhaps the most important block and implements the most interesting functionality. The interface controller has two primary responsibilities:

- Configure the bus interface of CSR modules once they are connected to a CSR-based device shell (e.g., IFB); and,
- Manage the dynamic switching-in and switching-out of bus interfaces such that all real-time constraints are met and system integrity is maintained.

To this end, the interface controller controls the individual enable signals for each of the bus interface blocks, and also communicates with the host environment to which the CSR module is connected. These connections are shown in Figure 26.

Since CSR computer modules are general processing environments and can be used with a variety of device shells, each module must configure its bus interface and internal structures such that it can appropriately control and coordinate the individual components of each device shell to which it is connected. This requires a communication protocol between CSR computer modules

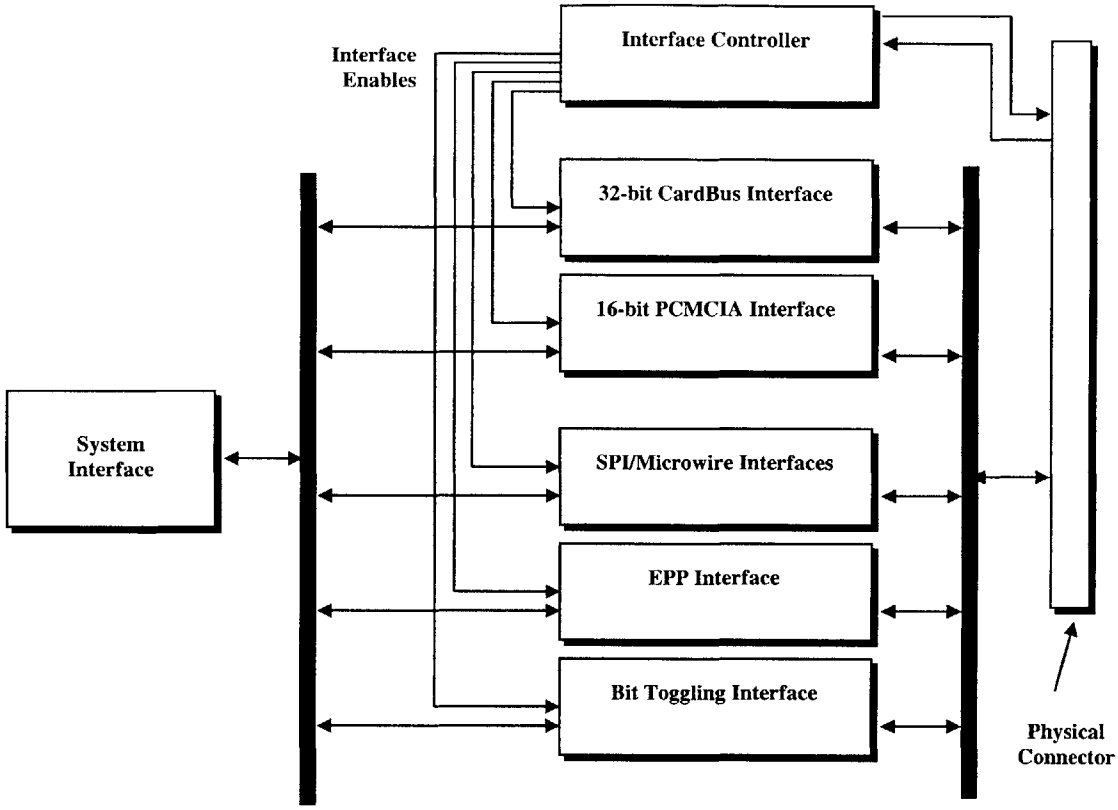


Figure 26: CSR Computer Module Internal Structures for static- and dynamic-interface reconfiguration.

and device shells. Once a computer module has been connected to a device shell and it has been reset, the interface controller immediately tries to communicate with the device shell. This communication aims to determine the exact characteristics of the device shell.

The interface controller of the computer module first transmits a predetermined message encoding announcing itself (e.g., "I am a CSR computer module") to the host environment using the interrupt and event messaging subsystem (as described in Section 3.4.5). This notifies the IFB that the connected module is in fact a CSR computer module, and is available to be selected as the leader computer module for the device.

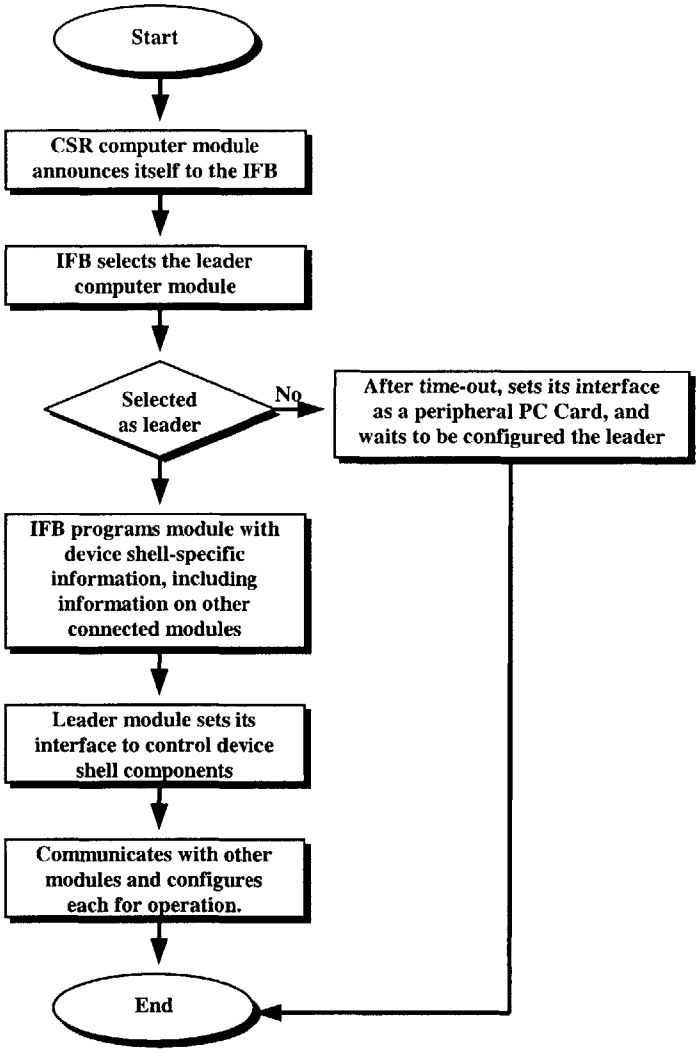


Figure 27: System configuration flow graph for CSR computer modules.

Once the IFB has selected the leader computer module, it then programs that module with information about the device shell. This could simply involve transferring an identifier that the computer module can de-reference (either locally or remotely over a network connection) to ascertain the exact specifications of the device shell. Otherwise, the IFB can also store locally the specification information and transfer it to the leader module as necessary. This information is decoded by software running on the local module, and used to configure the interface and timing

structure of each narrowpath channel. The semantic structure and methodology underlying the generation and use of CSR device shell specifications is discussed in more detail in Chapter 5.

The IFB also notifies the leader computer module about information pertaining to other modules that are currently connected (or become connected or disconnected). This information primarily pertains to the bus interface supported by the other modules, and whether the other modules are CSR computer modules. This information is used to configure the bus interface to use either the 16-bit PC Card or the 32-bit CardBus interface. With its bus interface configured, the leader computer module initiates communication with any other connected modules, and begins to configure them for proper operation. This is accomplished in accordance with the PCMCIA specification. This entire flow is depicted in Figure 27.

Once the static interface configurations have been completed, the role of the interface controller of CSR computer modules changes to managing the dynamic switching of the selected static interfaces. The application mix and its associated I/O characteristics determine to a large part the dynamic switching of PC Card and narrowpath interfaces.

The dynamic nature of switching-in and switching-out bus interfaces of CSR computer modules is dictated by the need to meet narrowpath channel timing. The application mix that is to be executed by the system determines the packet size, the effective clock frequency, and the packet period. As soon as all of the narrowpath channels are idle (for that cycle), the statically selected PC Card interface – either 16-bit PC Card or 32-bit CardBus – can be switched-in.

Once the PC Card interface is switched-in, the leader CSR computer module must take steps to guarantee that PC Card communications do not monopolize the IFB and the real-time constraints of all of the narrowpath channels are met. Since 16-bit PC Cards can never initiate transactions, CSR computer modules have full control as to the total time utilized. The number of 16-bit PC Card transactions possible between narrowpath transactions is simply the total time available divided by the cycle time for each transaction.

32-bit CardBus transactions pose a bigger challenge to maintaining narrowpath timing since these cards can initiate transactions. CardBus cards implement a latency timer register that can be

programmed with the maximum number of cycles a card can communicate at any one time. This register was originally envisioned to guarantee that other CardBus cards can initiate transactions without having to incur an exorbitant latency period. The CSR architecture takes advantage of the latency timer register to divide multi-packet CardBus transactions into a set of smaller burst size transactions. These smaller sets are communicated back-to-back to transfer the original amount of data. The number of these sets that are in fact transferred between two narrowpath transactions is controlled by the CardBus arbiter. The arbiter provides a grant signal to 32-bit CardBus cards, knowing that the de-assertion of grant will halt the transaction within the latency timer number of cycles from the start of the transaction.

The I/O characteristics of some application mixes may not be supported by the PC Card-based CSR architecture if their bandwidth needs or real-time constraints are too demanding. If the I/O characteristics of the application mix is within the tolerances of the PC Card-based CSR architecture, they can be analyzed and used to optimize the performance of the device.

These optimizations fall into two categories: (a) power consumption and (b) quality adjustments. Power consumption can be minimized by reducing the number of packets that are transferred over the PC Card connector between the CSR computer module and the device shell. For instance, if an application is decoding and transmitting to a DAC packets that represent voice-quality audio, the packet frequency can be reduced, thereby reducing I/O bandwidth and power. In the same vein, packet frequency (as well as other techniques such as reducing bit width) can be reduced so that a larger amount of time is available for PC Card transactions.

3.7.4 PC Card-based CSR Computer Module Signaling Specification

CSR computer modules place computational resources within a PC Card packaging. Although the architecture supports the use of off-the-shelf PC Cards and CardBus cards, some signals of CSR computer modules may not fully conform to those specified by the PCMCIA. The following chart enumerates and specifies the input/output direction of the implementation described herein

for each of its supported interfaces. The narrowpath interfaces specify the signal for two SPI interfaces (prefixed by S0_ and S1_) and one 8-bit asynchronous parallel interface.

Pin	16-bit PC Card Interface				32-bit CardBus Interface		Narrowpath Interfaces	
	Memory-Only		I/O and Memory					
	Signal	I/O	Signal	I/O	Signal	I/O	Signal	I/O
1	GND	DC	GND	DC	GND	DC		
2	D3	I/O	D3	I/O	CAD0	I/O	S0_CLK	O
3	D4	I/O	D4	I/O	CAD1	I/O	S0_CS	O
4	D5	I/O	D5	I/O	CAD3	I/O	S0_DOUT	I
5	D6	I/O	D6	I/O	CAD5	I/O	S0_DIN	O
6	D7	I/O	D7	I/O	CAD7	I/O	S1_CLK	O
7	CE1#	O	CE1#	O	CCBE0#	I/O		
8	A10	O	A10	O	CAD9	I/O	S1_CS	O
9	OE#	O	OE#	O	CAD11	I/O		
10	A11	O	A11	O	CAD12	I/O	S1_DOUT	I
11	A9	O	A9	O	CAD14	I/O	S1_DIN	O
12	A8	O	A8	O	CCBE1#	I/O		
13	A13	O	A13	O	CPAR	I/O		
14	A14	O	A14	O	CPERR#	I/O		
15	WE#	O	WE#	O	CGNT#	I		
16	READY	I	IREQ#	I	CINT#	I		
17	VCC	DC IN	VCC	DC IN	VCC	DC IN		
18	VPP1	DC IN	VPP1	DC IN	VPP1	DC IN		
19	A16	O	A16	O	CCLK	I		
20	A15	O	A15	O	CIRDY#	I/O		
21	A12	O	A12	O	CCBE2#	I/O		
22	A7	O	A7	O	CAD18	I/O	ADDR/DATA0	I/O
23	A6	O	A6	O	CAD20	I/O	ADDR/DATA1	I/O
24	A5	O	A5	O	CAD21	I/O	ADDR/DATA2	I/O
25	A4	O	A4	O	CAD22	I/O	ADDR/DATA3	I/O
26	A3	O	A3	O	CAD23	I/O	ADDR/DATA4	I/O
27	A2	O	A2	O	CAD24	I/O	ADDR/DATA5	I/O
28	A1	O	A1	O	CAD25	I/O	ADDR/DATA6	I/O
29	A0	O	A0	O	CAD26	I/O	ADDR/DATA7	I/O
30	D0	I/O	D0	I/O	CAD27	I/O	WAIT	I
31	D1	I/O	D1	I/O	CAD29	I/O	DATA_STRB	O
32	D2	I/O	D2	I/O	RFU			
33	WP	I	IOIS16#	I	CCLKRUN#	I/O		
34	GND	DC	GND	DC	GND	DC		
35	GND	DC	GND	DC	GND	DC		
36	CD1#		CD1#		CCD1#			
37	D11	I/O	D11	I/O	CAD2	I/O	ADDR_STRB	O
38	D12	I/O	D12	I/O	CAD4	I/O	READ/WRITE	O
39	D13	I/O	D13	I/O	CAD6	I/O		
40	D14	I/O	D14	I/O	RFU			
41	D15	I/O	D15	I/O	CAD8	I/O		
42	CE2#	O	CE2#	O	CAD10	I/O		
43	VS1#		VS1#		CVS1			
44	RFU		IORD#		CAD13	I/O		

45	RFU		IOWR#		CAD15	I/O
46	A17	O	A17	O	CAD16	I/O
47	A18	O	A18	O	RFU	
48	A19	O	A19	O	CBLOCK#	I/O
49	A20	O	A20	O	CSTOP#	I/O
50	A21	O	A21	O	CDEVSEL#	I/O
51	VCC	DC IN	VCC	DC IN	VCC	DC IN
52	VPP2	DC IN	VPP2	DC IN	VPP2	DC IN
53	A22	O	A22	O	CTRDY#	I/O
54	A23	O	A23	O	CFRAME#	I/O
55	A24	O	A24	O	CAD17	I/O
56	A25	O	A25	O	CAD19	I/O
57	VS2#		VS2#		CVS2	
58	RESET	I	RESET	I	CRST#	I
59	WAIT#	I	WAIT#	I	CSERR#	I/O
60	RFU		INPACK#	I	CREQ#	O
61	REG#	O	REG#	O	CCBE3#	I/O
62	BVD2	I	SPKR	I/O	CAUDIO	I/O
63	BVD1	O	STSCHG#	O	CSTSCHG	O
64	D8	I/O	D8	I/O	CAD28	I/O
65	D9	I/O	D9	I/O	CAD30	I/O
66	D10	I/O	D10	I/O	CAD31	I/O
67	CD2#		CD2#		CCD2#	
68	GND	DC	GND	DC	GND	DC

Table 2: Signal specifications for PC Card-based CSR computer modules.

3.7.5 Two-Way Configuration Registers

Configuration registers of CSR computer modules are accessible through the local interface (i.e., by the local processor of the computer module) and through its remote interface (i.e., the interface presented outward from the module's PC Card connector). In the case of leader computer modules connected to CSR IFBs, software running on the local processor of the computer module may choose to disallow access to all or some registers through the remote interface. However, if the computer module is connected to a standard PC Card socket then the computer module allows access to at least the PCMCIA-defined configuration registers through its remote interface.

3.7.6 Conclusion

This chapter described and specified one implementation of the CSR architecture, which is based on pluggable computational resource modules as well as interconnect fabrics that provide a means for module-to-module communication. The CSR architecture also simplifies device development through a structured device design cycle that supports the use of off-the-shelf, commodity components to implement each device's functionality. The next chapter evaluates the implementation of the CSR architecture described herein with respect to a variety of metrics, including power consumption, cost, and performance.

Chapter 4

CSR System Evaluation

In order to evaluate the CSR architecture and CSR-based devices, an entire prototype system and testbed has been developed. This section describes the implementation details of our prototype CSR IFBs and computer modules, as well as the implementation details of one of our testbed CSR-based devices – a wireless network-based multimedia picture frame.

Next, we evaluate the PC Card-based CSR architecture described herein with respect to cost, performance, and power consumption. The CSR architecture was primarily motivated by the need for a simple, yet formalized, design methodology for creating a variety of user interfaces (devices) through which to interact with network content and services. Since these devices fundamentally enable the delivery of content and services to consumers, their primary design goal is low cost. At the same time, the devices themselves must not retard the development or deployment of new content and services because of inadequate local (client-side) computational resources or communications performance. Accordingly, we evaluate the PC Card-based CSR architecture's performance, and show it to be sufficient to support a variety of multimedia applications. Finally, since many of these devices will have limited energy sources (e.g., from a battery) or have constrained physical size (e.g., for aesthetic reasons), power consumption is also evaluated. We show that the overhead introduced by the architecture is minimal, while in fact the

use of domain- and application-specific CSR modules caters to user preferences and improves performance along different dimensions.

4.1 Prototype System Development

4.1.1 Prototype Two-socket CSR IFB

Figure 28(b) depicts the schematic representation of our prototype two-socket CSR IFB. The significantly bussed implementation is simple with most traces directly connected between the two sockets and the device shell components. System power is delivered through a set of standard cell batteries connected to power and ground planes of the IFB.

A Xilinx XC4013XLT-PQ208-09 [45] Field Programmable Gate Array (FPGA) is used to implement the IFB Controller. Thirty-eight signal pins of the FPGA are used to interface with IFB and device component signals. The thick arrow line in the figure shows data and control signals between each device component and CSR modules, whereas the thin arrow line shows the chip select and interrupt request signals interface through the IFB Controller.

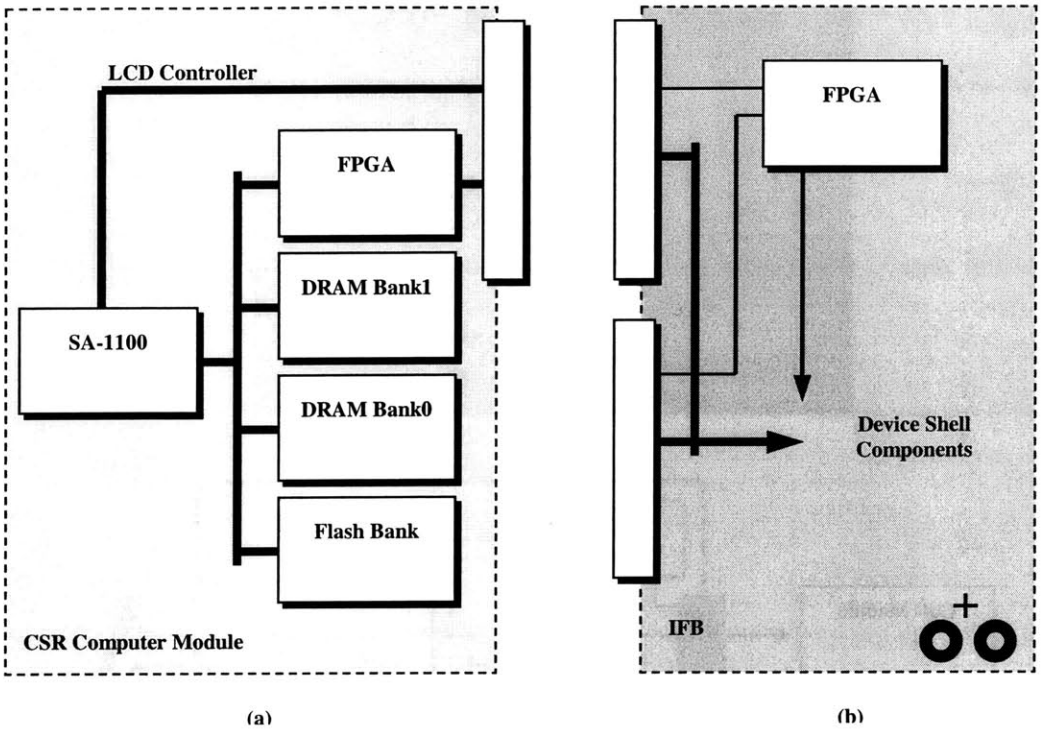


Figure 28: (a) Prototype implementation of an Intel StrongARM-based CSR computer module, and (b) a two-socket bussed CSR IFB.

4.1.2 Prototype CSR Computer Module

Figure 28(a) shows the schematic representation of our prototype CSR computer module. The module is based on an Intel StrongARM SA-1100-AA [30] microprocessor, with 16 Mbytes of self-refresh dynamic random access memory (DRAM) and 4 Mbytes of Flash. The CSR interface is implemented using a Xilinx XC4062XL-BG-432-09 FPGA [45] that straddles the PC Card connector on one side and the system memory bus on the other side. The FPGA appears to the SA-1100 hardware and to the operating system as the third DRAM bank with a standard DRAM interface on the system memory bus.

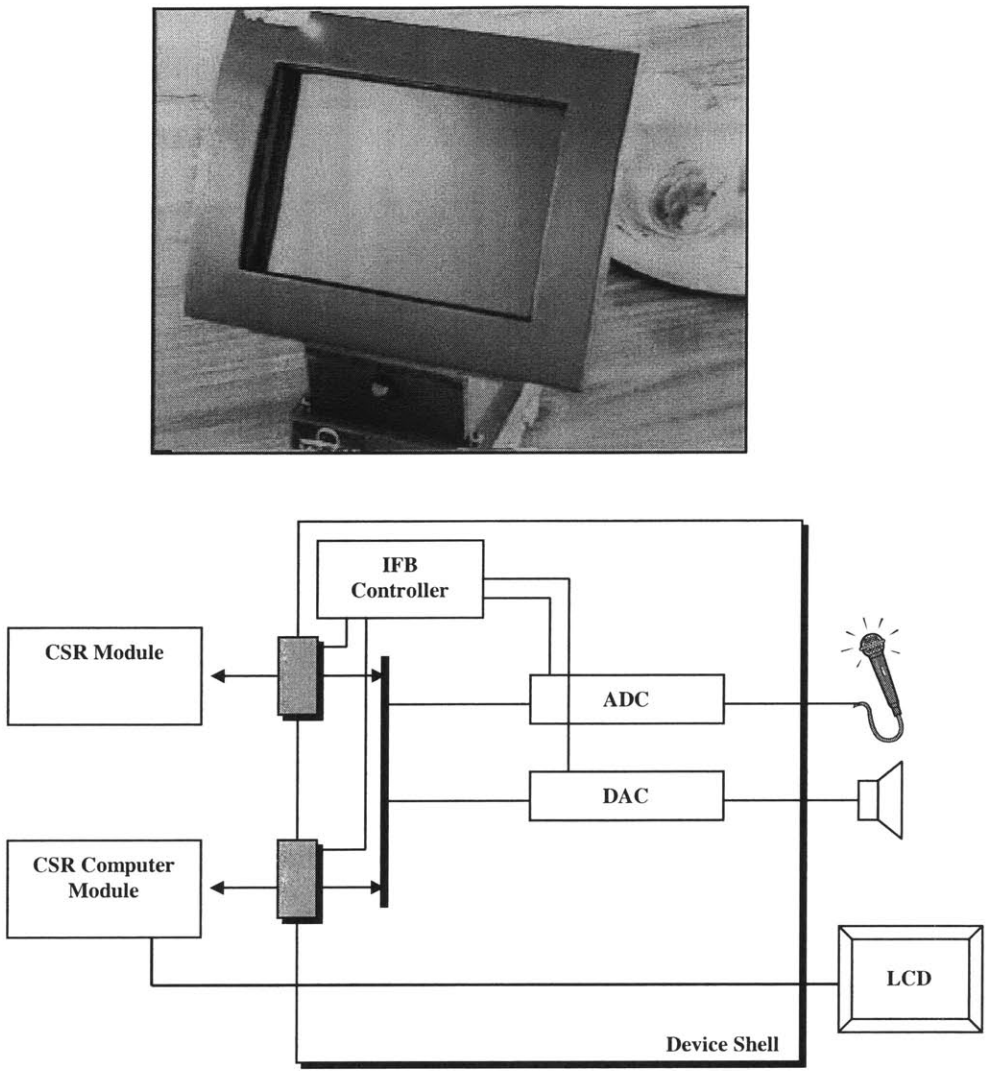


Figure 29: A photograph of the prototype CSR-based multimedia picture frame (top) and the schematic diagram for the system (bottom).

4.1.3 CSR-based Multimedia Picture Frame

Figure 29 depicts a simple schematic diagram of a prototype CSR multimedia picture frame, as well as a photograph of the actual implemented device. The device is capable of receiving

network-based multimedia data that can then be decoded and rendered through the picture frame. Not only can the device display images, e.g., in JPEG or GIF format, but it can also render audio material associated with the images. The device is capable of automatically looping through sets of multimedia content, or the user can use voice commands to select particular pictures. The small footprint and familiar shape of the device (e.g., similar to standard picture frames) gives it an intuitive user interface and makes it appropriate for use in locations within homes and offices where a bulky personal computer would not be appropriate.

4.2 System Analysis and Metric Evaluation

The CSR System grew out of a need for an architecture for developing a variety of network devices, each with custom user interfaces that facilitate interaction with different classes of content and services. Recognizing that a single computational platform cannot sufficiently optimize different user-defined parameters such as interactive performance and battery life, the CSR architecture proposed a segmentation between device UIs and their underlying computational resources. This segmentation is placed close to the device-specific UI components so as to simplify device UI development and lower unit costs. Additionally, with little electrical circuitry a part of the device UI, each CSR computational resource module can more fully optimize the device. This section evaluates the PC Card-based CSR architecture along these motivating goals.

The first part of this section summarizes the device design cycle inspired by the CSR architecture, and provides simple design time and design complexity analysis. We also analyze device deployment models to investigate opportunities to reduce the cost of CSR modules and CSR-based devices.

The second part of this section evaluates the bussed implementation of the PC Card-based CSR architecture described herein. The bussed implementation (as compared to a point-to-point implementation) was motivated by a desire to reduce the cost and size of CSR IFBs that must be embedded into device shells. Smaller and less expensive IFBs can be embedded within a larger

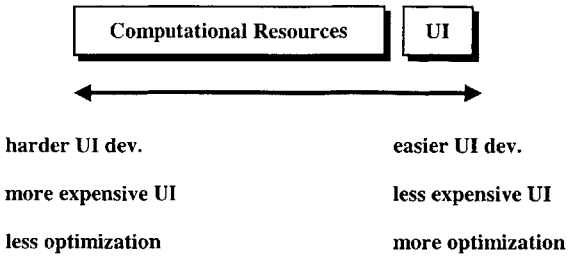


Figure 30: The segmentation proposed by the CSR architecture for the development of network content devices.

variety of device shells, thereby making the architecture more broadly applicable. At the same time, the bussed implementation must provide performance sufficient to support typical multimedia applications. To this end, we present a comparison of both cost and size between IFBs based on a bussed architecture versus those based on a point-to-point architecture, and demonstrate that the bussed implementation is less expensive while providing sufficient performance for typical multimedia applications.

The third part of this section analyzes the power consumption of a bussed PC Card-based CSR architecture. The modular nature of CSRs and the use of connectors is evaluated from a power consumption perspective. The power consumption of CSRs is compared with that of more traditional embedded solutions, and hybrid architectures that offer lower power consumption but with lower system flexibility are explored.

4.3 Cost Analysis

Cost is one of the most important factors to consider in systems design, and oftentimes motivates making trade-offs in other areas. Figure 30 depicts the positioning of the segmentation between device UIs and computational resource modules as selected by this implementation of the CSR architecture. The division is chosen to be close to the device UI so that UI development is simple and unit costs are low. Simple device UI development spurs the creation of a variety of custom UIs, while still catering to compressed time-to-market requirements. Since CSR modules are used to optimize the performance of device UIs with respect to user preferences and changing

network content, minimizing the amount of circuitry and complexity of device UIs allow these optimizations to affect a larger part of the device (as per Amdahl's Law).

In the next sub-sections, we evaluate in detail various aspects of the device architecture and design cycle that contribute to overall device cost.

4.3.1 CSR Device Design Cycle

The PC Card-based CSR architecture presents an abstraction layer to the design of devices, and structures the design process. The abstraction layer presented by CSR is similar to that of standard battery cells. A designer must merely understand the voltage requirements and current consumption of a design, choose the appropriate cells (e.g., AA, AAA, C, D), and place the proper set of battery clips. The structure placed on the power system design because of standard cell batteries greatly limits design complexity, eliminates many design errors, and reduces the qualifications, the training, and the number of required personnel.

To design a power system with batteries, a designer need only understand two simple rules: (a) place batteries in parallel to increase current; and (b) place batteries in series to increase voltage. With these two simple rules, a designer can build a complete and reliable power system. The designer need not understand the actual workings of a battery cell, nor must he concern himself with choosing the optimal battery type for the device. The use of standard battery clips will allow manufacturers and consumers alike to seamlessly and cost-effectively track innovation in the battery industry.

Device design with the CSR architecture is similarly easy. A designer must follow these steps to design a CSR-based device:

(1) Select a one- or two-socket CSR IFB to embed into the device;

(2) Build the functionality of the device using discrete off-the-shelf integrated circuits and components, making sure that each IC or component uses one of the many CSR-supported back-end electrical interfaces;

(3) Place these ICs and components on the IFB, connecting the back-end interface signals to the appropriate data and control paths as specified in Figure 26.

(4) Ascertain that all bandwidth requirements, real-time constraints, and overall quality-of-service needs of each of the device components are met. A set of simple-to-use rules as well as an entire design automation environment are described in Chapter 4.

The structured and reasonably simple design flow of the CSR architecture simplifies device design and eliminates much design complexity (and therefore errors), thereby reducing development and maintenance costs.

Using the CSR architecture and its structured design cycle, a developer with little training in hardware systems was able to design and fully build a personal digital assistant (PDA) with stylus-based touchscreen and LCD, audio output capability, and wireless networking. Another similarly trained developer was able to build a small autonomous vehicle capable of self-navigation using infrared sensors and GPS positioning data. Each of these projects was effectively realized within three weeks of design and development time. However, it is important to note that much of this time was spent in stabilizing the CSR computer module and CSR IFB prototypes.

A veteran designer of 20 years predicts a 38% reduction in overall design and development time for intelligent, interactive toys and devices built using the CSR architecture [62]. The compression in development time is primarily attributed to the following characteristics of the CSR architecture:

- Simplicity of device UI development. The use of CSR modules eliminates the large amount of time that is spent in building the device's system platform and booting the operating system environment. Instead, device designers are able to immediately build innovative and ergonomic device UIs around the CSR interface. Custom applications for that device UI can also be built immediately on top of the CSR module's operating system environment.
- Fast interaction with working device UIs. Designers and focus groups are able to "play with" new device UIs, quickly decipher which UIs and play patterns are appropriate, and iterate

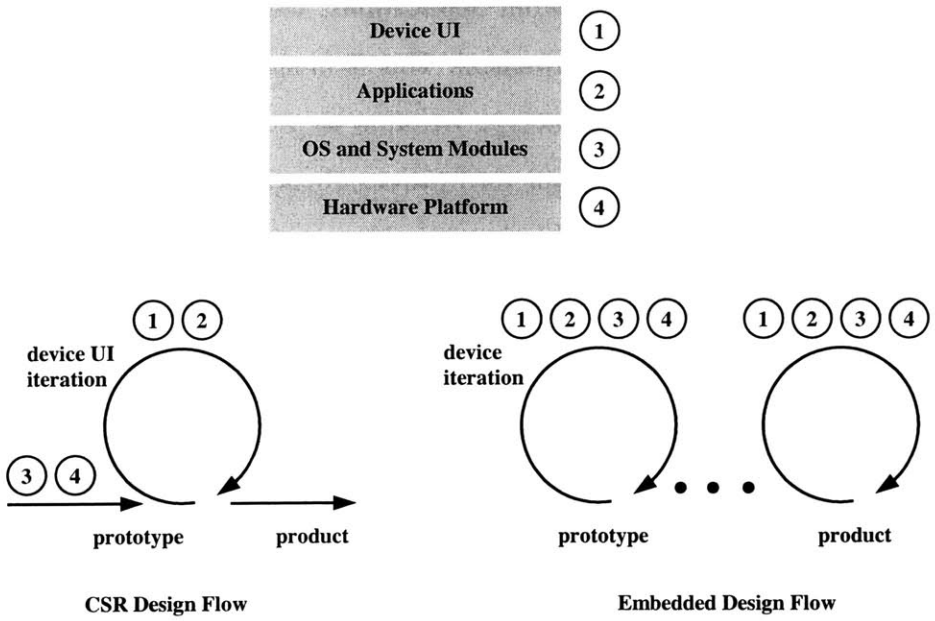


Figure 31: Comparison of device design cycle based on CSR architecture and status quo embedded architectures.

over different models. Since iteration over refined UIs is an inherently sequential process, a rapid development time for each device UI iteration is important.

- Seamless transition and design flow from prototype to final product. CSR devices are based on a modular architecture where a device UI hosts one or more CSR resource modules. This modularity simplifies prototype development but also immediately leads into final manufacturable product design.

Figure 31 shows a comparison between the design flow of devices based on the CSR architecture and status quo embedded architectures. Device design is segmented into four independent development blocks – the hardware platform, the operating system (OS) and system software modules, the custom applications, and the device user interface (UI). In the CSR design flow, device UI and custom applications can be immediately developed and hosted by existing hardware platforms and operating system environments. Iterations over the devices are simply

iterations over the device UI and custom applications, without modification to the hardware platform or the operating system environment. This is in stark contrast to status quo embedded systems development where each prototype iteration as well as the final product design step involves modifications to and subsequent verifications of each of the four development blocks.

Essentially, the CSR architecture extends the component software model [41, 42] to include hardware, and in doing so, realizes many of the benefits of the component software model. These benefits include ease-of-development around well-defined application programming interfaces (APIs), more rapid development times, and reduced development costs. Component re-use enables development costs and times to be prorated across multiple products, and supports more robust development and quality assurance measures.

Chapter 4 investigates the design flow inspired by the CSR architecture in more detail, and proposes an automated design environment that further simplifies overall CSR-based device design and development.

4.3.2 Device Deployment Cost Models

CSR modules are fully generic, and can make up the computational and peripheral resources for most device shells. Since CSR modules can be produced independently of the actual device shell, and the modules can cut across multiple market segments, the total achievable yearly volume for CSR modules can be extremely large.

Manufacturing learning curves and economies-of-scale can be leveraged to reduce the unit cost of each CSR module, and therefore the overall cost of each CSR-based device. Cost estimation and learning curve models state that beginning from a critical minimum volume, each incremental doubling of volume reduces units costs by 20% [58, 59, 66]. The solid line in Figure 32(a) shows the relationship between unit cost and total volume, assuming that the critical minimum volume for CSR modules is two million units. The horizontal dotted line depicts a constant $20 amount that would have to be paid per unit system if modular resources were not used. The figure shows that the unit cost of CSR modules drop from an initial value of $20 to

$16 for four million units, which is further reduced to $12.80 for eight million units. In this case, the hard costs of the computational resources of a device are reduced by $7.20, or 36%.

The cost savings of such a modular solution comes with the requirement for a set of connectors and associated circuitry. The gray line (middle) in Figure 32(a) shows the total unit cost for a CSR module and a pair of connectors. After 2.8 million total unit volume, the cost of the modular CSR solution is identical to that of an embedded solution. After 8 million units, the cost savings of the CSR solution is $5.92 and at 16 million units, the cost saving is $8.74, or 43.7%.

The modularity between CSR modules and device shells also supports the use of just-in-time inventory models [5, 40]. Using a just-in-time model, device shells can be manufactured, with the appropriate CSR modules assembled together just prior to device fulfillment. This allows companies to efficiently allocate limited assets only to the development of device shells, with the additional cost of CSR modules being only incurred upon fulfillment.

The just-in-time model may also reduce device inventory risks. With unpredictable consumer demand and fad-like consumer purchasing cycles, device manufacturers are oftentimes left with unsaleable inventory in their warehouses and retail channels. In order to recoup some of their costs, manufacturers sometimes slash the price on these unsaleable inventory lots. For CSR

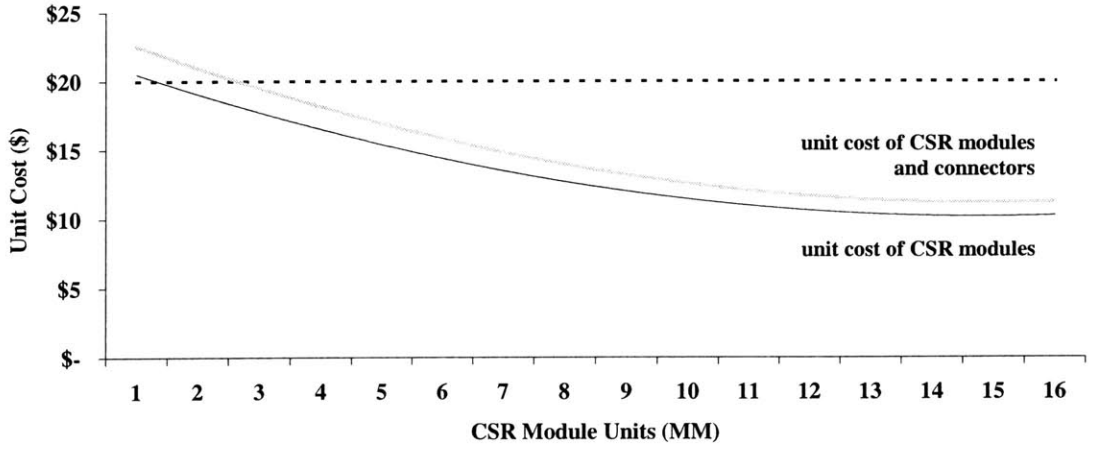


Figure 32: Device deployment cost analysis.

device inventory based on a just-in-time model, unsaleable inventory only includes device shells. It is not sufficient to slash the price of only device shells, as CSR modules must be bundled with the shells.

The following analysis shows that it may sometimes be more cost-effective to simply discard device shells, rather than bundling them with CSR modules and selling them at a reduced price. Let r represent the percentage of inventory that is unsaleable, b represent the percentage of total device cost that is represented by the CSR modules, and s represent the percentage of total revenue that is recoverable by using a modular architecture such as CSR. The relationship between these variables is represented by the equation:

$$s = r \times b$$

As increasingly more complex computational resources are used within devices, the value of b will continue to edge upwards. Assuming a constant value of r, the increasing value of b will drive potential economic benefits of using the CSR architecture linearly upwards. Conversely, the use of rapid-turnaround just-in-time models for the development and delivery of device shells may drive the value of r downwards. The simplicity of device shells will encourage the use of these rapid-turnaround just-in-time models. However, the need to have inventory on-hand to offer customers superior service will prevent r from ever reaching 0%. Nonetheless, as long as the price reduction required to liquidate inventory is greater than the percentage of total device cost represented by CSR modules b, it is economically more attractive to discard device shells all together rather than bundle shells together with CSR modules and sell them at a discount. This again reinforces the decision to push the segmentation between device UIs and CSR modules close to the device-specific UI.

4.3.3 Device Life Cycles and Total Cost of Ownership

Table 3 enumerates the lifecycle of different classes of devices. These lifecycles are affected by a variety of factors, including:

Product Class	Avg Life (yrs)
Cordless telephone	10
Color TV	8
Camcorder	7
CD player	7
VCR	6
PC	6
Telephone answering machine	5
Fax machine	4

Table 3: Average lifecycle for products in various classes.

Industry Clockspeed. Competitive pressures and innovation cycles differ by industry to industry. A slower rate of fundamental innovation in products negatively affects consumer purchasing cycles, and lead to longer device lifecycles [25].

Device Cost. More expensive devices have longer lifecycles, as it is economically difficult for consumers to purchase new models.

Personal Factors. Personal factors also affect individual product lifecycles. A particular device that has sentimental value to a family will have a longer lifecycle, in spite of new product innovations and feature introduction.

In most cases, homes, offices, and mobile environments will have a broad array of devices, each with its own lifecycle. However, unlike today's standalone devices, tomorrow's network-centric products will rely on network content, applications, and services. The computational and peripheral resources of longer lifecycle devices may have to be upgraded or augmented in order to allow consumers to take advantage of new services, or simply to enjoy these services at a comfortable interactive speed (e.g., upgrade for performance only).

The cost savings of simply being able to upgrade devices without having to always fully replace them (some of which are irreplaceable) is governed by:

$$t = u \times (1 - b)$$

where b is the percentage of total device cost that is represented by the CSR modules, u is the percentage of devices whose CSR modules are replaced either for upgrades or for maintenance, and t is the total cost savings of upgrading CSR modules instead of replacing complete devices. Expensive devices, such as those with large, high-quality LCDs, have large (1-b)s, and incur tremendous cost savings by simply upgrading CSR modules.

For many network devices, the device is merely the enabler for the delivery of network content and services. In these cases, revenue is derived primarily from the delivery of content and services, and not from the sale of network devices. In fact, most of these devices will be sold at cost, if not below cost. Accordingly, it is in the interest of content distributors to lower the total cost of ownership t of network devices by allowing consumers to simply swap CSR modules.

4.3.4 Pin Count, Size, and Cost

This section analyzes the cost savings of the bussed PC Card-based CSR architecture as compared to a point-to-point implementation. The bussed implementation fundamentally affects the IFB, and in particular, the IFB Controller. The pin count of CSR modules is determined by the connector used, and therefore, is fixed to 68 pins in the case of PC Cards.

Pin count primarily determines the cost of an integrated circuit. Not only are pins expensive, but the packaging needed to accommodate a larger number of pins is also expensive. Pin count also determines the minimum size a chip's silicon die since the die must be large enough to accommodate that number of I/O pads.

Pin count of a chip also affects the overall size of a system. A large pin count chip has a larger number of traces that must be routed on the printed circuit board (PCB), which affects not only the size of the PCB but also its number of layers. Moreover, even if a particular device implementation does not utilize all the narrowpath channels, a point-to-point IFB controller would need to implement all the pins, and therefore increase size and cost needlessly.

Metric		Bussed	Point-to-Point	Notes
Two-socket Interconnect Fabric Backplane Controller				
Size	Signal Pin Count	38	164	
	Standard Package Pin Count	PC44 CS48	PQ208	Assumes 20% of pins for VCC and GND.
	Gate Equivalents	39,400	n/a	
	Package Size	17.5x17.5 mm^2 7x7mm^2	30.6x30.6mm^2	
Package Cost		< $1	$ 2.38	
One-socket Interconnect Fabric Backplane Controller				
Size	Signal Pin Count	25	104	
	Standard Package Pin Count	PC44	TQ128	Assumes 20% of pins for VCC and GND.
	Gate Equivalents	26,200	n/a	
	Package Size	17.5x17.5 mm^2	16x22mm^2	
Package Cost		< $1	> $1	

Table 4: Size and cost analysis for one- and two-socket CSR IFB Controllers.

Table 4 compares the pin count, package size, and package cost of our bussed PC Card-based CSR implementation to that of a point-to-point implementation [65]. The comparison is shown for both the two-socket and the one-socket CSR IFB.

Having discussed the cost and strategic advantages of the CSR architecture, the remainder of this chapter evaluates the performance of the herein described PC Card-based CSR implementation.

4.4 Performance Analysis

4.4.1 Scheduling and Bandwidth Performance

The implementation of the PC Card-based CSR architecture aims to minimize overall IFB cost and size by building a bussed interconnect fabric. Multiple interfaces traverse the shared fabric in a time multiplexed manner, with each interface only receiving a series of allocated time slots for transactions. This section analyzes the effect time multiplexing has on application performance and available bandwidth.

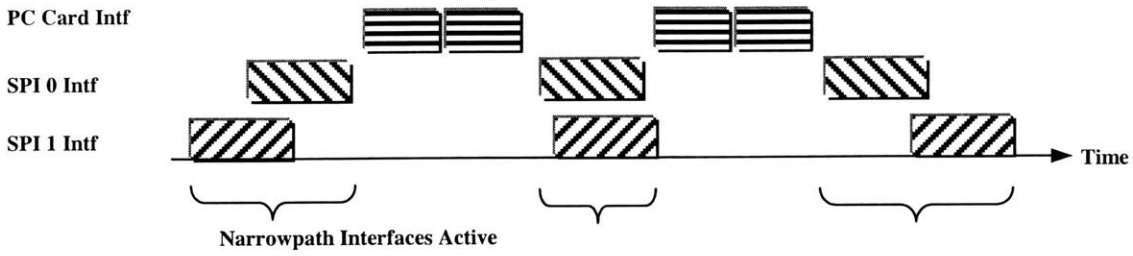


Figure 33: PC Card-based CSR Multibus Interface Scheduling.

Figure 33 depicts the time-multiplexed scheduling between narrowpath and PC Card interfaces. The figure shows two narrowpath interfaces – both using SPI – active simultaneously, but working asynchronously and with different time frequencies. The frequency of individual narrowpath interface is dictated by its associated narrowpath component. For instance, if a DAC component with a backend SPI protocol is used to render audio through a speaker, each packet may need to be delivered to the DAC in 30 microsecond intervals. The number of bits transmitted to the DAC together with its clock frequency determines the time duration (e.g., width) of each narrowpath transaction. The union of the time slots allocated to the two SPI components represents the total time dedicated to the narrowpath interface. It is important to note that the total narrowpath time segment is a dynamically varying quantity, depending on the alignment of the independent narrowpath transaction packets.

All remaining time is used for PC Card transactions. Narrowpath interfaces are scheduled first as each narrowpath transaction may have individual real-time deadlines. PC Card transactions, on the other hand, may simply have deadlines for the delivery of blocks of data. In order to guarantee that PC Card transactions do not delay narrowpath transaction, the maximum transaction time (in clock cycles) may be specified and constrained. PC Cards may utilize larger blocks of free time by cascading smaller transactions back-to-back.

Figure 34 shows bandwidth distribution to narrowpath and PC Card interfaces for simple applications using a variety of 16-bit PC Cards and 32-bit CardBus cards. The device is a simple network radio that receives digital content, decodes it, and outputs it to a digital-to-analog converter, which, in turn, converts the data to an analog signal and outputs it to a speaker. The

figure examines the effect on PC Card bandwidth with changing narrowpath conditions. The variables of interest are (a) the frequency of packet delivery to the DAC, which affects audio quality, (b) the clock speed of the DAC, which affects the time required to transmit each packet to the DAC and the amount of time available for PC Card transactions, (c) the clock speed of the PC Card, which affects PC Card communication bandwidth.

The figure and its associated table show actually used narrowpath bandwidth and total available PC Card bandwidth values for voice- and CD-quality audio. CD-quality audio requires 44kHz frequency, whereas voice-quality audio requires just 11kHz sampling frequency. Since the number of narrowpath transactions and the amount of data transferred during each transaction is fixed, increasing the clock speed of the DAC simply makes available more bandwidth for the PC Card. Increasing the effective clock frequency of the PC Card, obviously increases only the total bandwidth available to the PC Card.

The two plots below the table graphically depict the presented data for CD- and voice-quality audio devices. The graphs demonstrate that increasing the clock speed of PC Cards (or the effective clock speed for asynchronous PC Cards) is more effective in increasing total bandwidth available for PC Card transactions than increasing the clock speed of narrowpath components. For instance, using a 33 MHz 32-bit CardBus card instead of a 10 MHz (100 ns cycle time) 16-bit PC Card increases available bandwidth by 567%, whereas using a 10 MHz SPI DAC instead of a 2 MHz SPI DAC increases available bandwidth for PC Cards by 7%. It is important to note that using CardBus cards are especially effective not only because of their higher clock speed but also because of their wider 32-bit datapaths.

Device Characteristics		Pkt Size (Bits)	Payload (Bits)	Freq (MHZ)	Bandwidth (MBps)	Mux Time (ns)	%
Voice Quality, Low Speed DAC with Slow PC Card	NP	16	12	2	0.02	8,000	9%
	PC		8	2	1.82	82,000	91%
CD Quality, Low Speed DAC with Slow PC Card	NP	16	12	2	0.09	8,000	35%
	PC		8	2	1.30	15,000	65%
Voice Quality, High Speed DAC with Slow PC Card	NP	16	12	10	0.02	1,600	2%
	PC		8	2	1.96	88,400	98%
CD Quality, High Speed DAC with Slow PC Card	NP	16	12	10	0.09	1,600	7%
	PC		8	2	1.86	21,400	93%
Voice Quality, Low Speed DAC with Fast PC Card	NP	16	12	2	0.02	8,000	9%
	PC		16	10	18.22	82,000	91%
CD Quality, Low Speed DAC with Fast PC Card	NP	16	12	2	0.09	8,000	35%
	PC		16	10	13.04	15,000	65%
Voice Quality, High Speed DAC with Fast PC Card	NP	16	12	10	0.02	1,600	2%
	PC		16	10	19.64	88,400	98%
CD Quality, High Speed DAC with Fast PC Card	NP	16	12	10	0.09	1,600	7%
	PC		16	10	18.61	21,400	93%
Voice Quality, Low Speed DAC with CardBus Card	NP	16	12	2	0.02	8,000	9%
	PC		32	33	121.48	82,000	91%
CD Quality, Low Speed DAC with CardBus Card	NP	16	12	2	0.09	8,000	35%
	PC		32	33	86.96	15,000	65%
Voice Quality, High Speed DAC with CardBus Card	NP	16	12	10	0.02	1,600	2%
	PC		32	33	130.96	88,400	98%
CD Quality, High Speed DAC with CardBus Card	NP	16	12	10	0.09	1,600	7%
	PC		32	33	124.06	21,400	93%

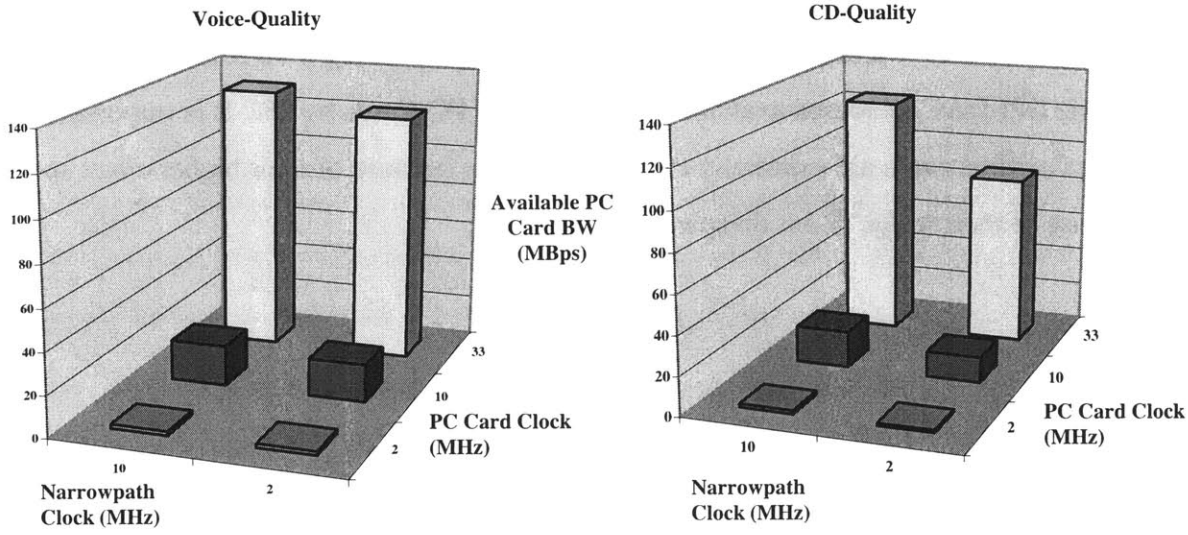


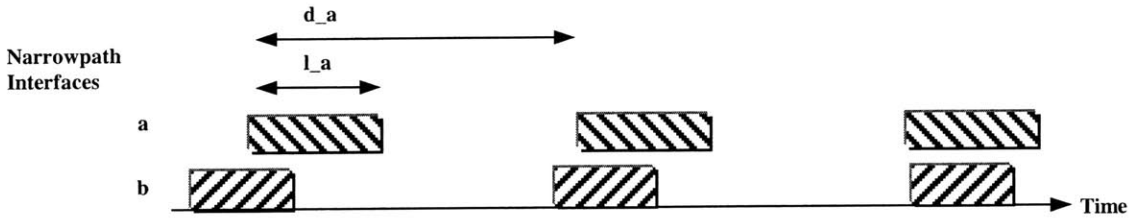
Figure 34: Total bandwidth utilization by narrowpath channels and maximum bandwidth available to CSR modules for two configurations of a simple CSR-based audio device.

The C-like pseudo-code shown in Figure 35 describes a simple algorithm for discerning the time available between multiple independent sets of narrowpath interfaces. Each independent narrowpath interface is assigned a subscript character. The variable `l_a` denotes the length of each transaction packet, and is determined by the number of bits transmitted in the packet and the clock speed of the transmission. In asynchronous hand-shaking transactions, such as those using parallel and RS-232 serial interfaces, `l_a` is the entire cycle time of the transaction. The variable d_a denotes the periodicity of transaction packets, and is determined by the packet's target (or source) I/O component, e.g., digital-to-analog converter. For instance, a audio output application may deliver digitized audio level packets to a digital-to-analog converter on 30 microsecond intervals.

$$l_a(synchronous) = bitwidth_a \times \frac{1}{clk_a}$$

$$d_a(synchronous) = \frac{1}{f_a}$$

Each independent narrowpath channel and the application or sets of applications that use each channel are characterized by different `l_a` and `d_a`. Accordingly, the `pccard_gap` value is a dynamically changing value depending on the alignment of transaction packets from the various narrowpath channels.




```
occupied = FALSE;
state = UNOCCUPIED;

for (i = 0 ; i < (d_a * d_b * ...) ; i++)
{
  if (remainder (i / d_a) <= l_a)then occupied = TRUE;
  if (remainder (i / d_b) <= l_b)then occupied = TRUE;
  .
  .
  .

  if (state == UNOCCUPIED AND occupied == TRUE)
  {
    unoccupied_stop_time = i;
    state = OCCUPIED;
  }
  if (state == OCCUPIED AND occupied == FALSE)
  {
    unoccupied_start_time = i;
    state = UNOCCUPIED;
  }
  pccard_gap = unoccupied_stop_time - unoccupied_start_time;
}
```

Figure 35: C-like psuedo-code that describes how to determine the available time between narrowpath transactions, which in turn can be utilized for module-to-module communications.

In order to maximize bandwidth available to PC Cards, each of these dynamically changing `pccard_gap` values must utilized to the fullest and using the most efficient PC Card transactions. CardBus cards support burst transactions in which a single address packet is following by a number of payload or data packets. The overhead of burst transactions is simply the address packet, which is amortized by a large number of data packets. The desire to use large burst transactions is tempered by the desire to fill as many `pccard_gap` time slots between narrowpath transactions as possible with PC Card transactions.

Figure 36 shows the effective payload bandwidth achievable for various burst transaction sizes. Multiples of small burst sizes, such as using two payload packets following the address packet, can more fully take advantages of the time gaps between narrowpath transactions, but each of these transactions have large amount of overhead (e.g., 33%). Once the burst size becomes too large, many time gaps are left unutilized, and overall bandwidth suffers.

Figure 36(a) depicts the optimal burst size for a CSR-based audio device based on two SPI narrowpath components. The first component is a digital-to-analog converter that can receive 24 bits of data with a maximum clock frequency of 4 MHz. To realize CD-quality audio, data packets must be delivered to the DAC every 23 microseconds. The second component is an analog-to-digital converter that can receive 16 bits of data at a maximum clock frequency of 2 MHz. It is also sampled every 23 microseconds. The audio device also uses a 32-bit CardBus network card, capable of transferring data between an Ethernet network and the connected CSR computer module. The optimal burst size, as shown in the figure, is 283[§]. This burst size with a 33MHz 32-bit CardBus achieves a maximum bandwidth of 78 Mbytes per second between the network card and the CSR computer module.

Figure 36b shows the optimal burst size for the same audio device as in Figure 36a, but with the sampling period of the ADC increased to 90 microseconds, and with independent left- and

[§] CardBus cards support burst sizes in multiples of 16. Accordingly, the actual optimal burst size is 256 with an aggregate bandwidth of 72 Mbytes per second.

right-stereo channels. Figure 36c shows the optimal burst size for an audio device capable only of mono audio output (i.e., the ADC has been eliminated).

These slight variations in hardware configuration may simply be a result of changes in application software. For instance, an application that allows digitizing of analog music may select a greater ADC sampling rate than that used by a voice recognition application. Figure 36 shows that such changes in software may result in corresponding changes to the optimal burst size for CardBus transactions. This allows the application mix and its associated I/O characteristics to be analyzed apriori (e.g., at compile or configuration time) so as to configure the CSR architecture for optimal module-to-module performance. Even if the optimal bandwidth is not required for any particular application mix, the optimization process minimizes power consumption from module-to-module I/O communication by eliminating unnecessary overhead packets.

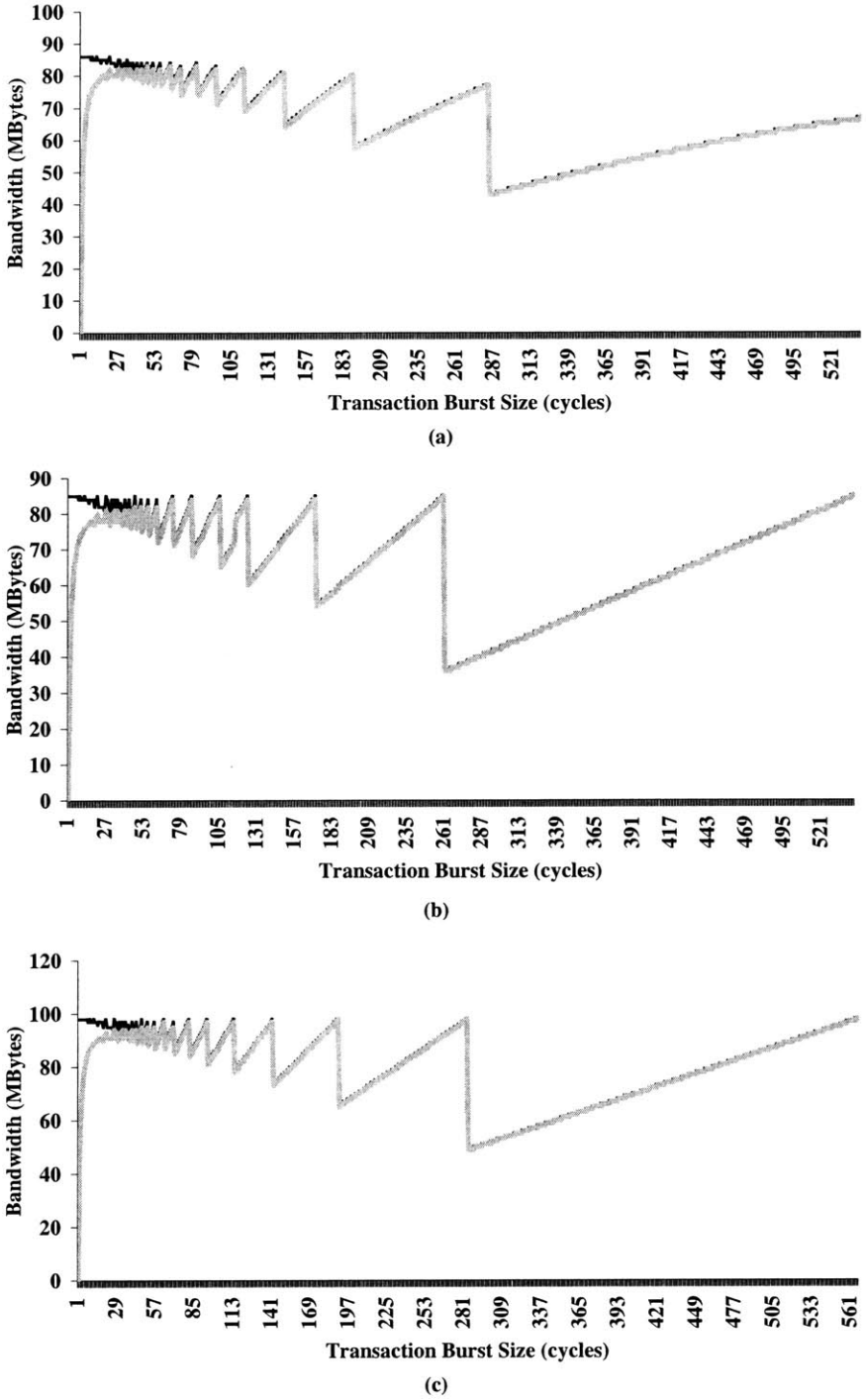


Figure 36: The optimal CardBus transaction burst size for achieving maximum module-to-module communication bandwidth.

Only 32-bit CardBus cards support burst transactions. 16-bit PC Card transactions support the transfer of a single packet of data per complete cycle. Figure 37 shows the bandwidth available for module-to-module communications using 16-bit PC Cards for the same device and configuration as that of Figure 36. Data for a variety of different transaction cycle speeds as specified by the PCMCIA are presented.

Next, we analyze bandwidth for IFB-to-IFB communications. As described in Section 3.6, sets of one- and two-socket IFBs can be connected together to realize multi-socket IFBs that are appropriate for larger devices, such as set-top boxes and hubs. In this architecture, IFBs communicate with one another using the narrowpath channels, but there typically are no real-time deadlines for the delivery of individual packets. Instead, there are real-time deadlines for the delivery of blocks of data. Accordingly, IFB-to-IFB communications can be inlined with other narrowpath communications. This mechanism makes IFB-to-IFB communications transparent, effectively hiding intra-IFB communications such that PC Card transaction time is not reduced.

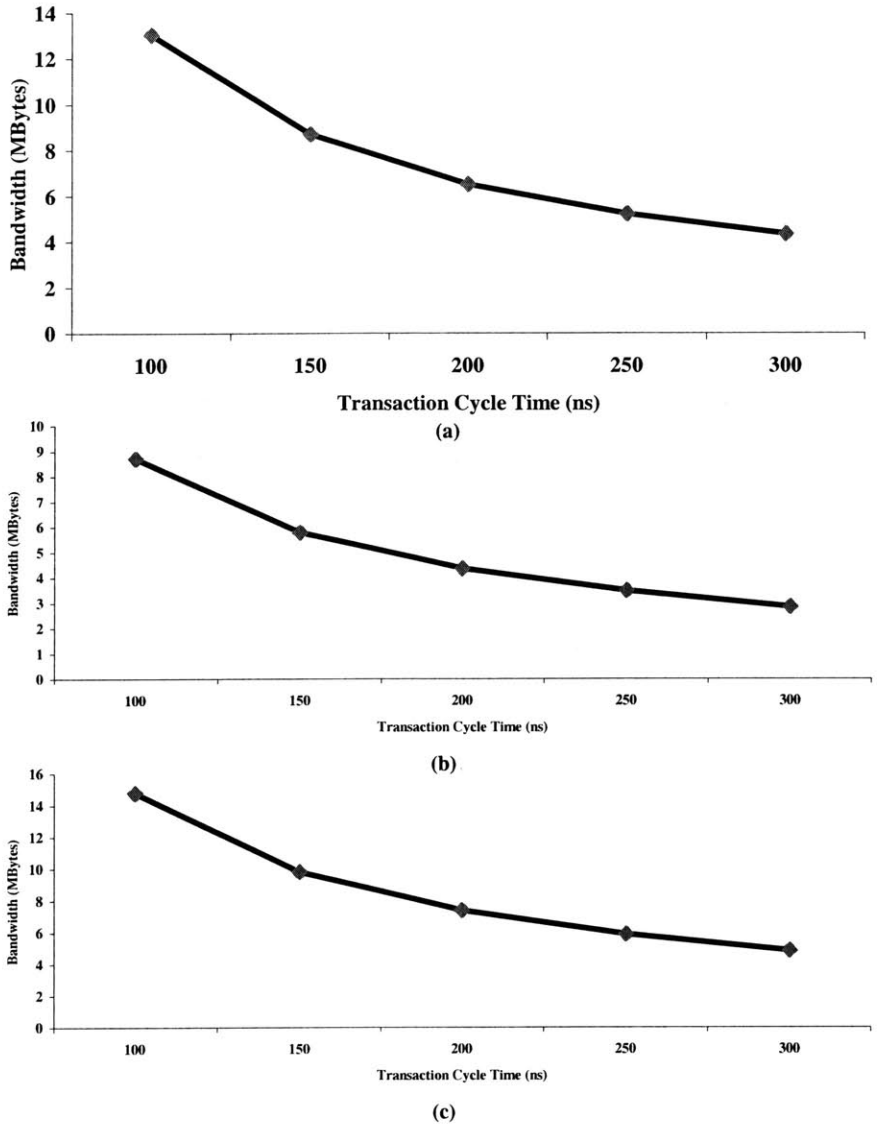
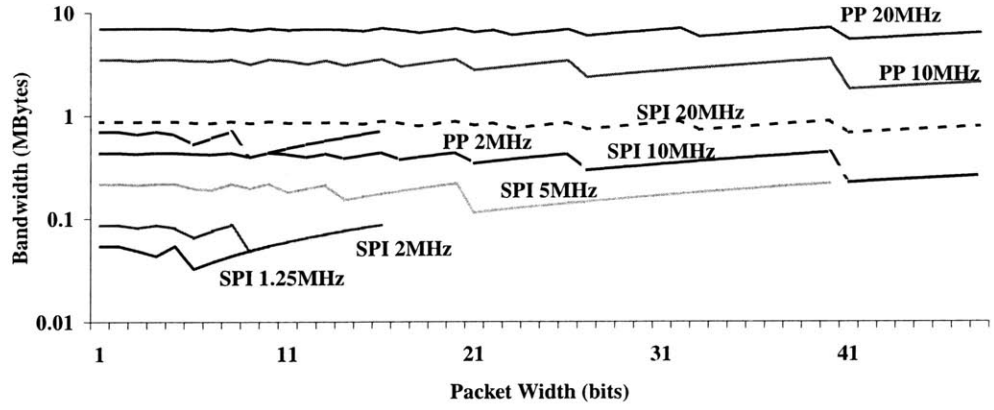


Figure 37: Maximum module-to-module bandwidth achieveable with 16-bit PC Cards for various cycle speeds.

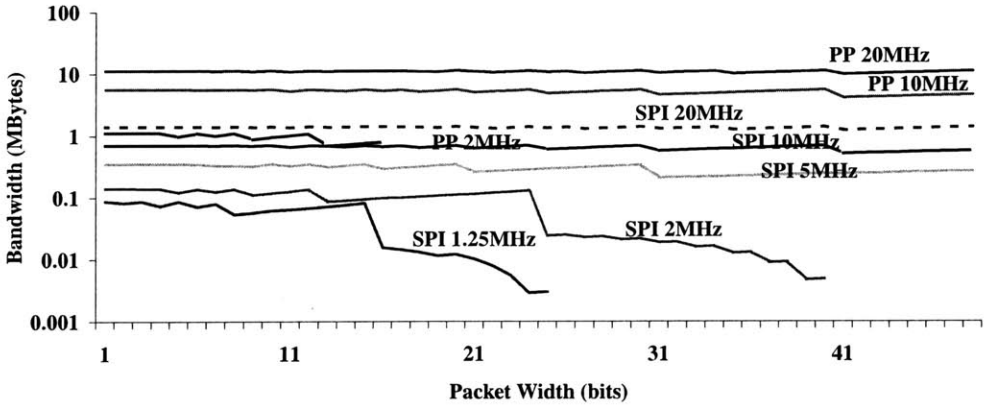
Figure 38 shows the maximum bandwidth achieved by inlining IFB-to-IFB communications with standard narrowpath channels for the audio device configurations associated with Figure 36 and Figure 37. Each graph shows a variety of narrowpath interfaces and cycle times (a SPI interface with 1.25 MHz, 2 MHz, 5 MHz, 10 MHz, and 20 MHz as well as a 8-bit Parallel Port

interface with 2 MHz, 10 MHz, and 20 MHz) plotted against packet length. Since the standard narrowpath channels determine the total amount of time available for intra-IFB communications, maximum bandwidth is achieved by varying the packet length for each particular cycle time. This scheme most fully utilizes the time available for intra-IFB communications and results in maximum bandwidth.

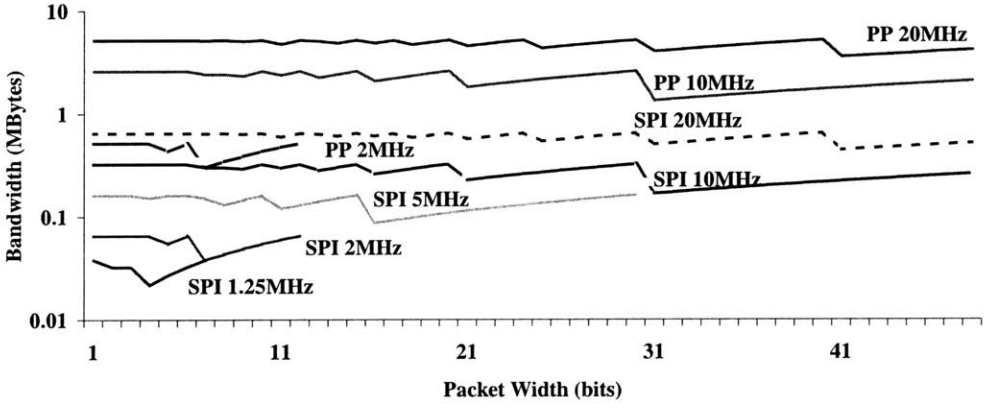
The data shows that standard 8- and 16-bit packets are not always optimal. Instead, for the system configuration depicted in Figure 37(a) and using a 1.25 MHz SPI interface, a 10-bit packet results in the maximum bandwidth. The data presented assumes that each packet has no overhead (i.e., 100% payload) once the channel has been configured. Moreover, the data represents the bandwidth achievable by the source IFB only. Effective IFB-to-IFB bandwidth is limited by the minimum achievable bandwidth of both the source and the target IFB, and the application mix I/O characteristics of each. Increased bandwidth can be realized by dedicating narrowpath channel time to intra-IFB communications (i.e., make IFB-to-IFB communications non-transparent by explicitly allocating additional time than that already warranted by standard narrowpath channels). Differences in packet length and cycle time between the source and the target IFB can be rectified by the narrowpath buffers used.



(a)



(b)



(c)

Figure 38: Maximum bandwidth available for IFB-to-IFB communications for a simple CSR-based audio device plotted against packet width (in bits) the figures assume that IFB-to-IFB communications are fully inlined with other narrowpath communications.

Subsystem	Required Bandwidth (MB/s)
Graphics	30 to 40
Full-motion video	2 to 9
Ethernet	2
Hard disk	5 to 20
CD quality audio	1
Cellular data network	0.001
3G cellular data network	0.125 to 0.250

Table 5: Bandwidth requirements for various multimedia subsystems.

Having analyzed the maximum achievable bandwidth for module-to-module and IFB-to-IFB communications, we next discuss the bandwidth required for typical multimedia subsystems. Table 5 lists the bandwidth for these typical subsystems. Devices may be based on a single listed subsystem or a combination thereof.

The module-to-module bandwidth data shown in Figure 36 demonstrates that sufficient bandwidth exists over a bussed IFB to support each of the subsystems listed in Table 5. The bandwidth of the lower speed and narrower datapath 16-bit PC Cards as shown in Figure 37 cannot support the requirements for a graphics subsystem nor can it support a high-performance hard disk drive. Similarly, the IFB-to-IFB bandwidth data presented in Figure 38 demonstrates that by using a high-speed parallel narrowpath interface, all but the graphics and high-performance hard disk drive subsystems can be supported.

4.4.2 Isochronous and Asynchronous Communication Limits

The relationship between isochronous narrowpath communications and asynchronous PC Card communications was shown in Figure 33. Since narrowpath communications have real-time constraints, they are scheduled first. Any remaining time is allocated to PC Card communications. The following equations show the relationship between the amount of time allocated to narrowpath communications and that allocated to PC Card communications.

$$f_k(t) = \begin{cases} 1 \text{ when } \mathrm{Rem}(\frac{t}{p_k}) < c_k \\ 0 \text{ when else} \end{cases}$$

$$L_{PCCard_{avg}} = (p_1 \times p_2 \times ... \times p_n) - \sum_{t=1}^{(p_1 \times p_2 \times .. \times p_n)} f_1(t) \mid f_2(t) \mid ... \mid f_n(t)$$

where $f_k(t)$ states whether the kth narrowpath channel is active at time t. c_k is the total cycle time for data transmission as determined for synchronous interfaces by the product of the number of serial bits transmitted and the transmission clock period. p_k is the worst-case periodicity of the transmission. Figure 35 depicts c_k and p_k as l_a and d_a, respectively. L_{PCCard_avg} relates the average percentage of time that is allocated to PC Card transactions as opposed to narrowpath transactions.

L_{PCCard_avg} provides a simple rule-of-thumb for determining the amount of bandwidth that is available for module-to-module communications once a set of narrowpath components have been selected. However, it is important to note that L_{PCCard_avg} is only an approximation, as the actual amount of *useable* time for module-to-module communications is constrained by each module's minimum transaction time. If a module's minimum transaction time is larger than the time available between two narrowpath transactions, then that time is not useable and cannot

contribute to overall module-to-module bandwidth. The PCMCIA defines multiple minimum transaction times for 16-bit PC Cards, including 100 ns, 200 ns, 250 ns, 500 ns, and 600 ns. The 32-bit CardBus interface uses an address cycle followed by a set of data cycles. The minimum transaction time is therefore the sum of the address cycle and a single data cycle. Assuming a 33 MHz CardBus clock, the minimum transaction time is 60 ns (2 cycles at 33 MHz).

4.4.3 Application I/O Performance

This section analyzes the performance of the PC Card-based CSR architecture from a software application perspective. The round-trip time to read data from I/O resources – either other CSR modules or device shell components – must be sufficiently small to enable the simple and reliable design of software applications. A large round-trip time limits the utility of interactive applications.

Table 6 enumerates the round-trip time to read data from both CSR modules and device shell components. The first two rows show round-trip times for reading configuration data from a 3Com Megahertz 10/100 LAN CardBus card (Model 3CXFE575BT). The card was clocked at 10 MHz, which is significantly slower than its peak frequency of 33 MHz. Each read request from configuration space returns a single 32-bit data word, which specifies salient information about the CardBus card.

The timing data is gathered from the delivery of the first read request from the SA-1100 to the CSR computer module FPGA (see Figure 28) to the completion of the transaction as signaled by the FPGA back to the SA-1100. The single packet round-trip time demonstrates the importance of using a time- and space-based watermark mechanism in the inbound FIFO as described in Section 3.7.2. After a set elapsed time, even if the FIFO is not nearing capacity, software is notified of the inbound data. The 256-packet read overflows both the inbound and outbound FIFO buffers, and must be segmented by software (e.g., based on FIFO ready interrupts) and delivered to the circular FIFO buffer. It is important to note that the current implementation of CSR computer modules add reasonable overhead to each read or write transaction. This overhead

Read Location	Transaction Size and Count	Round-trip Time (ns)	Notes
3Com 10/100 LAN CardBus Card	1 32-bit packet	480	Single data packet is delivered rapidly because of time- and space-based inbound FIFO watermark
3Com 10/100 LAN CardBus Card	256 32-bit packets	66,230	Multiple outbound and inbound packets take advantage of circular FIFO buffer
SPI-based Touchscreen Controller	1 16-bit packet	8,690	

Table 6: Round-trip times for application software running on a CSR computer module to read data from another CSR module and narrowpath component.

derives from outbound and inbound interrupts received by the StrongARM 1100, as well as from memory reads and writes. An integrated ASIC implementation will reduce these overheads.

The last row in Table 6 shows the round-trip time to read data from a device shell component using the SPI narrowpath interface. The component is a touchscreen controller that returns the *x*- and *y*-coordinates of the stylus on a resistive touchscreen. The value of the touchscreen controller buffer is returned immediately upon a read request (e.g., there is no delay to read the touchscreen).

4.5 Power and Energy Analysis

It is important to study power consumption because many CSR-based systems will be mobile devices that rely on a battery source for energy. Minimizing the power consumed by a system is important for the following reasons:

- Increased battery life. Energy-efficient devices reduce the number of times that batteries must be swapped in and out, thereby increasing convenience.
- Reduced fully loaded device weight. Batteries contribute a significant percentage of the weight of a mobile device. Since power efficient devices can last hours if not days on a reasonable sized battery, the weight of the device and the battery is much less than if the device were to use a larger and more bulky energy source.

- Reduces the amount of heat generated by the system, and supports the development of mobile devices that do not need built-in cooling mechanisms.
- Reduces the possibility of system failure from overheating.
- Reduces overall device costs by limiting the cost of batteries and by eliminating the cost of cooling mechanisms, such as fans.

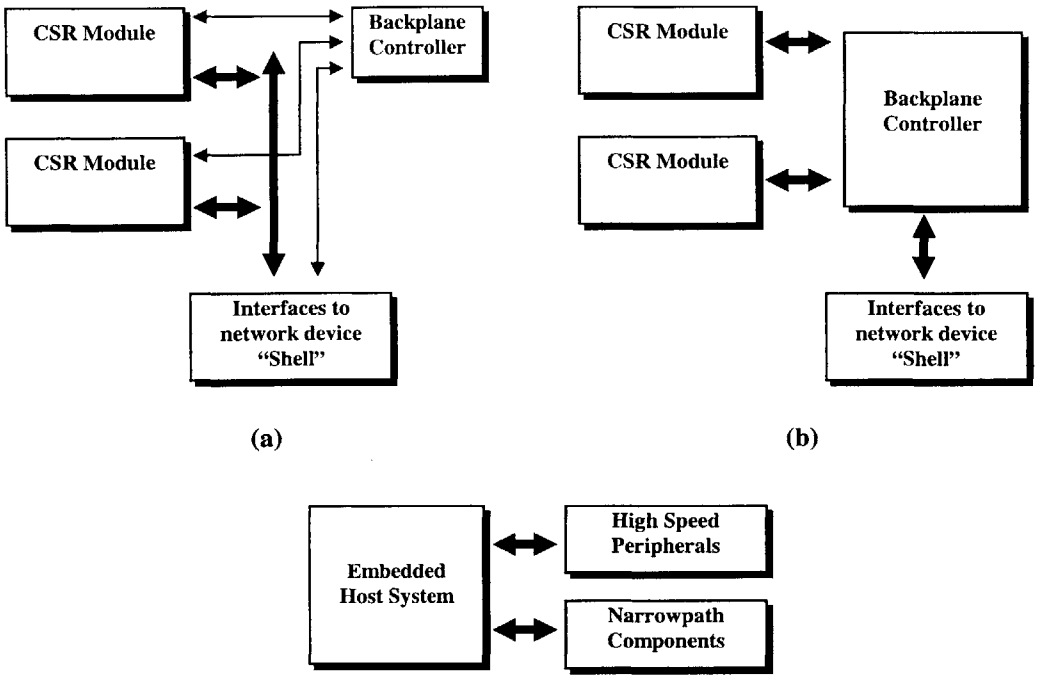


Figure 39: The three architectures compared for power analysis.

In our study and evaluation of power consumption, the three architectures shown in Figure 39 are compared. The first architecture, shown in Figure 39(a) depicts a CSR architecture based on a bussed IFB as described herein. The second architecture, shown in Figure 39(b), is a CSR architecture based on a point-to-point IFB. The point-to-point IFB eliminates the need to time-multiplex multiple interfaces over a shared bus, and also reduces the capacitive load that must be switched during each transaction. The third architecture, shown in Figure 39(c), is a status quo custom embedded architecture. In this architecture, there are no pluggable modules, and the host system uses a wide datapath such that direct and dedicated paths exist between each peripheral and the host system.

The power consumed by a system is proportional to:

$$C \times V^2 \times f$$

where C is the effective capacitance switched, V is the voltage at which the circuitry operates, and f is the average signal switching frequency. Therefore, reducing any or all of C, V, or f will reduce the power consumption of a system. C and f each have a linear relationship to power, whereas V has a squared relationship and can impact system power more aggressively. Changes in V may also affect signal propagation speed, and therefore limit the maximum system operating frequency.

Table 7 shows the power consumption of a PC Card-based CSR radio device in various operational states. The first row shows that power consumption for the processor alone is minimum. In this state, the processor is simply fetching and executing instructions from Flash memory. Next the processor configures the entire system, including the dynamic random access memory (DRAM) banks. Since DRAMs must continuously be refreshed, they consume a significant amount of power. Moreover, DRAMs operate at a reasonably high frequency, and the processor must switch the capacitance of all components on the memory bus at that frequency. Since, in our current implementation, the FPGA is located on the main memory bus, it, together with the DRAM banks, Flash banks, and associated PCB traces, present a large capacitive load to the processor. An application-specific integrated circuit (ASIC) implementation of CSR computer modules in which the FPGA circuitry is integrated together with the processor core presents an opportunity to reduce power consumption significantly. Communications between

Description	Current (mA)
Processor boot up with DRAMs idle.	180
System configuration with DRAMs active.	420
Software MP3 decode with no output to DAC.	510
Software MP3 decode with stereo DAC connected.	780

Table 7: Power consumption of a PC Card-based CSR radio device in various operational states.

the processor core and the FPGA circuitry would be on-chip, and therefore could be based on a lower voltage and with lower capacitance. The external memory bus would only house the DRAMs, Flash, and ROMs.

Power consumption can also be reduced and performance enhanced by using a reconfigurable co-processor. The co-processor may comprise a set of reconfigurable blocks, each of which can be configured by software to implement a particular function, such as MP3 decoding. The primary processor, on the other hand, can execute software that implements control and management functions. By using dedicated functional cores realized within a set of reconfigurable processors, runtime performance can be enhanced while power consumption is reduced.

Another significant jump in power consumption results from the CSR computer module driving data and commands across its connector and onto the digital-to-analog converter (DAC) of the radio, which is physically located on the device shell. Since compressed audio files, such as MP3, are decoded by the processor and then transmitted to the DAC as raw audio packets, the I/O bandwidth is large. The large number of I/O communications over the capacitive connectors and their associated traces also contribute to the jump in power consumption.

[57] presents an evaluation of the power consumption of 3Com's Palm Pilot. For the 19MHz Palm Pilot, power consumption is between 152 mW to 188 mW. At 3 volts, this is 50.7 mA to 62.7 mA of current consumption. Comparing these results to those of the 190 MHz CSR-based system enumerated in Table 7 shows that power consumption is roughly an order-of-magnitude less for the Palm, which is what we would expect for the 10X reduction in frequency. It is important to note that although power consumption is proportional to the frequency f, the switching capacitance C, and the square of the voltage V ($P \propto CV^2f$), the time necessary to complete any task is proportional to the inverse of the frequency f ($t \propto 1/f$). Now since, energy E is proportional to the power consumed P times the total time t, energy consumption is theoretically invariant with frequency f. But a reduction in frequency typically increases energy consumption as it extends the time that associated components remain powered. In fact, for

	Modular			Embedded
	Bussed		Point-to-Point	
Avg trace length	5 p-to-p to IFB Cntrl	1000 mils x 2	1000 mils x 2	1000 mils
	42 lines between modules	1000 mils		
	9 lines to narrowpath	2000 mils		
Avg capacitive load per pin	5 p-to-p to IFB Cntrl	12 pF x 2	12 pF x 2	12 pF
	42 lines between modules	12 pF		
	9 lines to narrowpath	24 pF		
Avg capacitive load per pair of connector pin		4 pF	4 pF	n/a
Total capacitive load per pin	5 p-to-p to IFB Cntrl	17 pF x 2	17 pF x 2 17 pF, 13 pF	13 pF
	42 lines between modules	21 pF		
	9 lines to narrowpath	34 pF		

Table 8: Capacitive loading from pin input capacitance and PCB trace lengths for each of the architectures of Figure 39.

interactive devices, such as personal digital assistants, in which burst transactions are dominant, a higher clock frequency will reduce overall energy consumption.

Next, we take a closer look at the sources of power consumption in the CSR architecture, and present a comparative analysis between the power consumed by features of the CSR architecture and that of more traditional architectures. A primary difference between the CSR architecture and traditional embedded solutions is the modularity between CSR modules and device shell components. This results in an increase in capacitance, and therefore, a proportional increase in power consumed, by the connectors and associated traces of the modular CSR architecture. Additionally, the implementation of a bussed CSR architecture as described herein contributes to power consumption in two ways. First, the shared bus of the IFB houses multiple capacitive loads, each of which is switched during module-to-module and module-to-device-shell communications. Second, the time multiplexing of multiple interfaces over the shared bus affects the frequency by which each signal line is switched. A discussion of both capacitive loading and switching frequency issues and their effect on power consumption by the bussed CSR architecture follows.

Table 8 enumerates the capacitances associated with the implementation of each of the three architectures of Figure 39. The modular two-socket bussed architecture (depicted in Figure 39(a))

has 5 point-to-point signal lines (not counting DC power and ground lines) that interface with the IFB controller. Assuming a device shell comprised of three SPI components, there are 9 signal lines with three (two CSR modules and one narrowpath component) loads each, and 42 signal lines that are bussed directly between the two CSR modules. The capacitances associated with each of these lines are shown in Table 8 assuming nominal values of 12 picofarad input capacitance and 1000 mil (1 inch) trace length per pin.

The modular point-to-point architecture requires a retransmission of each transaction by the IFB controller. Module-to-module communications must traverse two pairs of connectors, while module-to-narrowpath communications only traverses a single pair of connectors. Accordingly, module-to-module communications incur 17 pF of capacitive load twice. Module-to-narrowpath communications incur 17 pF of load for module-to-IFB-controller communications, and an additional 13 pF of load for IFB-controller-to-narrowpath communications. We find that as long as more lines are bussed only between the two CSR modules than between the two CSR modules and the narrowpath components, the modular bussed architecture incurs a slightly lower capacitive loading on average.

The custom embedded solution incurs the lowest capacitive load, as single, direct paths exist between each two potential communicating parties. Moreover, since modular connectors are not used, no additional capacitive loading is incurred. Each pin of this architecture drives just 13 pF of capacitive load. It is important to recognize that the high volumes of CSR modules (as opposed to typical custom embedded solutions) will afford manufacturers the time to optimize each module's power consumption (e.g., from improved ASIC construction and superior PCB layout and routing), thereby offsetting some increase in capacitive loading.

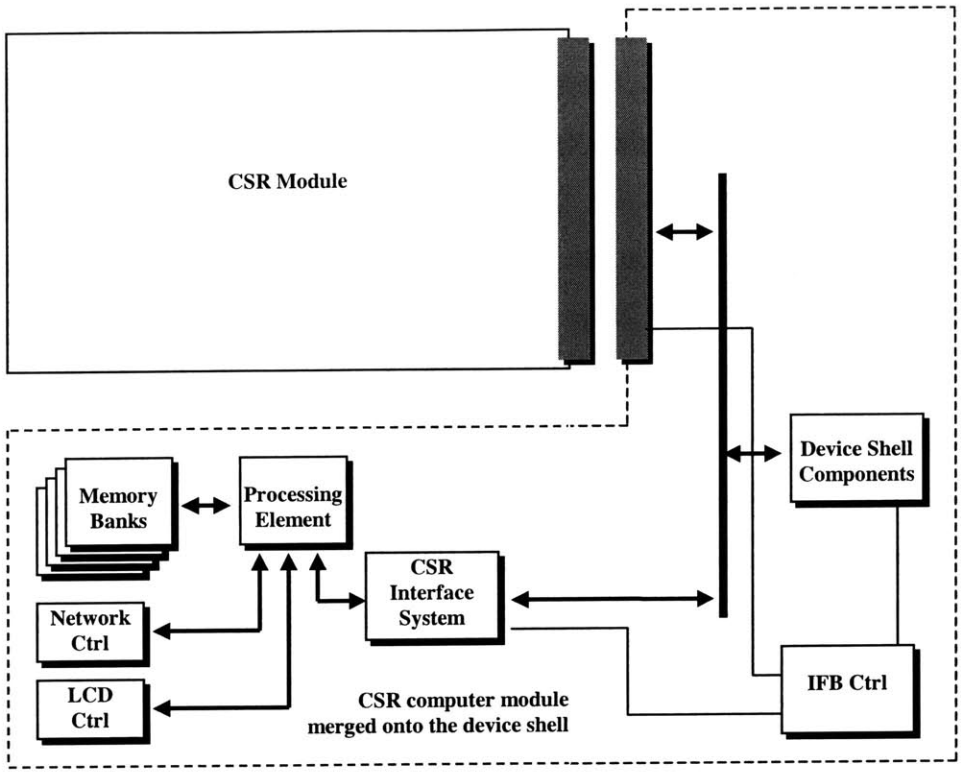


Figure 40: Embedding CSR computer modules into device shells to lower signal capacitance.

Interestingly, the additional capacitance of using connectors and a modular solution can be reduced by simply embedding at least one CSR module into the device shell. In this scheme, a single connector may be used to support an additional CSR module. This architecture still maintains a simple and seamless flow from prototype development to final product release, while optimizing the capacitive load. This is shown in Figure 40.

The capacitive loading incurred during narrowpath communications can be reduced by providing point-to-point buffers for those signals. For example, assuming that a system uses three SPI narrowpath channels, these nine signals can be buffered and re-driven to their intended target as shown in Figure 41. The buffer simply routes and re-drives data (with no format conversion) to the narrowpath ports when narrowpath communications are to transpire and re-drives data to the other CSR module otherwise. Determining when narrowpath communications are to be

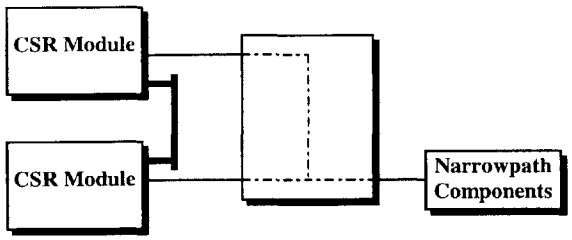


Figure 41: Buffering selected narrowpath signals to lower capacitive loading for narrowpath communications.

switched in and a mechanism with which to configure the IFB is discussed in Sections 3.4.2 and 3.7.3.

Having quantified the amount of capacitance that must be switched for each signal class, we next evaluate the actual switching frequency. Since CMOS-based systems consume power only when a signal is switched, comparing the switching frequency of the CSR architecture to that of status quo embedded solutions provides a valuable benchmark.

Custom architectures oftentimes provide dedicated paths for classes of signals so as to reduce switching frequency. For instance, consider the effect of separating the address bus from the data bus. If the address is incremented consecutively, only a few bit toggles are necessary to realize each subsequent address. The unrelated data values do not corrupt the address bus, and bit-wise proximity of each subsequent address value is leveraged to reduce overall switching frequency. Dedicated busses for different signal classes reduce frequency at the cost of a larger number of signal paths and their associated pins, ASIC size, and required routing area. All of these contribute to an increase in unit cost.

A tabulation of raw bit switching count for a split-bus architecture (labeled "Dedicated Paths") as well as that for a PC Card-based CSR bussed architecture (labeled "Time-multiplexed Paths") are shown in Table 9. A theoretical maximum switching count that is based on all time-multiplexed bits switching on each cycle is also shown. The percentage increase (labeled "% Incr") is with respect to the switching count of the split-bus dedicated architecture. The data presented is from a simulated audio device comprising two SPI-based DACs for stereo audio output and one SPI-based button controller capable of controlling a few interactive buttons (e.g.,

play, stop, pause). The data underlying the switching count is from one minute of streaming sinusoidal data, consisting of two 44 kHz sampling streams, each of which are fetched by a CSR computer module from either a 16-bit PC Card or a 32-bit CardBus card, buffered by the computer module, and then decoded and output at the appropriate time to both DACs.

32-bit CardBus cards implement an address and data bus which time-multiplexes address and data values over a common 32-bit bus. Since the first cycle of a CardBus transaction is an address cycle followed by a burst of data cycles, one expects little change in switching count from time-multiplexing narrowpath transactions in between two CardBus transactions. This is verified by the data presented in Table 9.

16-bit PC Cards, on the other hand, implement an address bus and a separate data bus. 24-bits of address and up to 16-bits of data can be driven over the 40-bit split bus. Correlations between and bit proximity of subsequent addresses may be reduced by time-multiplexing narrowpath transactions over the address bus. One would expect less disruption by time-multiplexing narrowpath transactions over the data bus since typically, there is little correlation between subsequent data values. This is especially so if the data is compressed, encrypted, or generally encoded.

	Dedicated Paths	Time-Multiplexed Paths		Theoretical Maximum	
	Switching Count	Switching Count	% Incr.	Switching Count	% Incr.
16-bit PC Card with 3 SPI channels time-multiplexed with the PC Card address bus	550,144,903	552,182,882	0.37%	597,663,121	7.95%
16-bit PC Card with 3 SPI channels time-multiplexed on every other cycle with the PC Card address bus	439,112,130	440,131,205	0.23%	462,871,290	5.13%
16-bit PC Card with 3 SPI channels time-multiplexed only as necessary with the PC Card address bus	328,383,741	328,386,531	0.00%	328,448,829	0.02%
16-bit PC Card with 3 SPI channels time-multiplexed with the PC Card data bus	550,391,431	542,706,981	-1.3%	597,663,121	7.95%
16-bit PC Card with 3 SPI channels time-multiplexed on every other cycle with the PC Card data bus	439,235,452	435,079,581	-0.9%	462,871,290	5.13%
16-bit PC Card with 3 SPI channels time-multiplexed only as necessary with the PC Card data bus	328,384,075	328,373,210	0.00%	328,448,829	0.02%
32-bit CardBus card with 3 SPI Channels time-multiplexed on each cycle	458,420,524	450,840,599	-1.6	505,938,724	9.4%
32-bit CardBus card with 3 SPI Channels time-multiplexed only as necessary	337,938,438	337,937,510	0.00%	337,939,260	0.00%

Table 9: Bit switching count for a split-bus dedicated architecture and for a time-multplexed bussed architecture. A theoretical maximum based on all time-multiplexed bits switching on each cycle is also shown. The percentage increase is with respect to the split-bus dedicated architecture switching count.

The data in Table 9 shows that there is a slight increase (less than one-half of one percent) in switching count when the narrowpath transactions are time-multiplexed over the 16-bit PC Card's address bus. Interestingly, there is a slight decrease in switching count when the narrowpath transactions are time-multiplexed over the data bus. Either way, the theoretical maximum switching count increase is under 8%.

The data presented also shows that switching count can be lowered further by simply reducing the number of times that PC Card transactions are interspersed with narrowpath transactions. That is, it is more efficient to simply increase the number of PC Card transactions each time the PC Card interface is activated than it is to space-out PC Card transactions equally following every narrowpath transaction. The third and sixth rows of Table 9 show that required module-to-module bandwidth is maintained while switching count is reduced dramatically by fully utilizing the time between subsequent narrowpath transactions. This optimization can only be used if

sufficient buffer space exists in CSR computer modules to temporarily store the data fetched from the other CSR module.

The foregoing discussion on capacitive loading and switching frequency has shown that capacitive loading is the dominant factor contributing to the increase in I/O power consumption by the PC Card-based bussed CSR architecture. Referring back to Table 8, the loading of a majority of signals increases by 62% (from 13 pF to 21 pF) while the loading of the small number of narrowpath signals increase by 162% (from 13 pF to 34 pF). The I/O power consumption of narrowpath communications is primarily affected by the additional loading of other CSR modules, while that of module-to-module communications is primarily affected by the loading incurred by the use of PC Card connectors. It is important to note that the capacitive loading data is based on maximum rated values for the PC Card connectors, associated traces and vias, CSR modules (and PC Cards), and typical narrowpath components. The actual values will be smaller in most cases.

Figure 40 and Figure 41 depicted systems based on the CSR architecture but with reduced capacitive loading from the elimination of a set of connectors and multiple loads, respectively. Each of these techniques can be used to reduce the power consumption overhead of using the PC Card-based CSR architecture and pluggable CSR modules.

4.6 Discussion

This chapter presented the specifications for the CSR architecture and an implementation of the PC Card-based CSR architecture. The primary design goal of CSRs was to create a system architecture that facilitates the rapid development of devices such that a flurry of easy-to-use devices that implement a platform for the delivery of a variety of network content and services can be quickly realized. Such a system architecture would not only allow network content providers to differentiate themselves through their content and service offerings, but also through the user interface of the device through which the content is delivered.

Such a system architecture must take a holistic view incorporating the many metrics and tradeoffs involved in systems design. The three primary metrics are cost, power consumption, and performance. Each of these metrics trade off with one another. For instance, power consumption can be reduced arbitrarily by simply trading off performance and cost. Performance can be increased by simply trading off cost and power consumption. Although all of these three metrics are important, perhaps the most important and the one that the PC Card-based CSR architecture stresses, is cost. Essentially, if a device is too expensive, no matter how attractive its performance and power consumption are, it will not be deployed in mass quantities, and it will not become a platform for mass network content and services.

Traditional embedded systems optimize for cost by amortizing design time over a large volume of devices. This allows designers to leverage longer design cycles to also optimize for performance by painstakingly tailoring software to the embedded system and for power consumption by customizing any ASICs and their printed circuit board layout. However, as content providers strive to offer superior service and customize their offerings to thinly sliced market segments, and sometimes even to individual consumers, the volume over which an embedded system can be amortized becomes increasingly small. This raises unit costs, and because of the compressed design cycle, performance and power consumption also suffers. This is depicted in Figure 42.

The CSR architecture is based on a high volume platform (CSR modules) that can simply be "skinned" with device shells to create low volume custom devices. The relationship between embedded systems and CSR-based devices along the dimensions of power consumption, performance, and cost is discussed next.

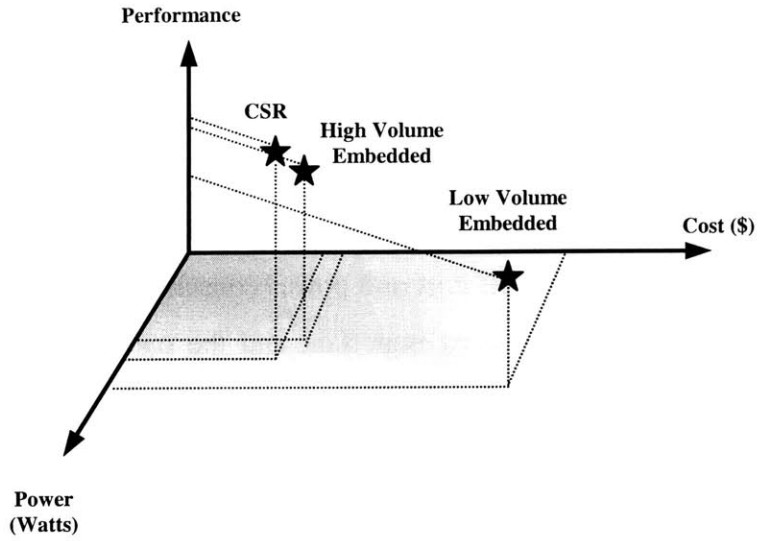


Figure 42: The trade-offs between power consumption, unit cost, and performance for high- and low-volume embedded solutions as well as for modular CSR-based systems.

Cost. The added cost of using the modular CSR architecture is the cost of connectors and the cost of assembling device shells and CSR modules with local labor. Assuming the cost of connectors is bounded at 10% of the cost of CSR modules (e.g., $2 for connectors for $20 CSR modules) and assuming the cost of local assembly is bounded at 5% of the cost of CSR modules ($1 for local assembly for $20 CSR modules), total additional cost is bounded by 15% of the cost of CSR modules. Recalling from Section 4.3.2, that unit costs of CSR modules drop by 20% with each incremental doubling of volume after a critical minimum, the added costs of using CSR modules is more than fully recovered in a mature CSR-based device market. The significant reduction in design time demonstrated in Section 4.3.1 and Chapter 5, the small size and cost of a significantly-bussed IFB implementation as shown in Section 4.3.4, the decline in total cost of ownership by manufacturers and consumers swapping CSR modules discussed in Section 4.3.3, and the cost savings from the use of just-in-time assembly and cost recovery from unsaleable devices described in Section 4.3.2 further bolster the position of CSR-based devices as a system architecture whose cost savings rival that of high volume embedded systems.

Power Consumption. The power consumed by CSR-based devices is higher than that of custom embedded devices. Section 4.5 showed that the primary contributor to this increase in power is the additional capacitive loading incurred from the use of connectors between CSR modules and device shells, as well as from multiple loads (CSR modules and narrowpath components) being placed upon CSR signal lines. Although the use of the CSR architecture introduces additional power consumption, it also presents the opportunity to use domain- and application-specific CSR modules that cater to user preferences and more aggressively reduce power consumption.

Performance. Sections 4.4.1 and 4.4.3 demonstrated that sufficient module-to-module, IFB-to-IFB, and module-to-narrowpath bandwidth exists to implement most popular multimedia applications using the current bussed PC Card-based CSR system implementation. Moreover, CSR-based devices offer improved performance over their lifetime by providing a mechanism to upgrade the computational and peripheral resources of the device as dictated by changes in "killer app" content and service offerings.

Chapter 5

Characteristic-based CSR Device Design

A simple device design flow enables Internet portals and any company whose core competence is not in developing devices to build *branded, custom, and content-specific* devices. Such a design flow removes the dependence and uncertainties associated with contracting a third-party device manufacturer, as well as lowers unit costs and affords the portal to choose from a variety of business models that make sense for the device.

If the device design flow is sufficiently simple then designers themselves can fully develop a device, without having to involve a set of engineers. Removing the need to communicate creative ideas and concepts from designers to engineers reduce overall time-to-market and system errors borne out of miscommunication. In the Post-PC Internet, as the characteristics of content-specific devices are used to enhance and differentiate content, and as more of our analog devices become digital, a simple design flow empowering creative designers themselves to fully develop devices will be necessary.

The CSR architecture provides an opportunity to enable such a simple device development flow. Essentially, the CSR architecture abstracts away the computational resource platform of

devices, and also provides a structured framework around which the actual user interface of the device can be built. The abstraction presented by the use of pluggable CSR modules eliminates the need to make many, and oftentimes, very difficult design decisions. The difficulty in making these decisions stems primarily from the need to develop high-volume one-size-fits-all devices that appeal to the majority of consumers while accommodating a large variety of content and services. Since CSR modules allow consumer to build custom platforms that cater to their needs and preferences, these design decisions and their underlying analyses can be eliminated.

Next, the set of independent narrowpath channels implemented by the CSR architecture simplifies device UI development. Since narrowpath channels are based on many standards-based electrical interfaces and protocols, off-the-shelf commodity components that provide the desired functionality (e.g., motor control, digital-to-analog conversion) can be coordinated together to realize the desired overall UI. The use of off-the-shelf components further reduces the risks and costs of device development, while providing rapid time-to-market.

The next sections describe a design flow and process that is enabled by the CSR architecture.

5.1 Desired Design Flow

A desirable design flow is one in which the intricacies of computer systems design are abstracted away so that only those features that are most important to the designer (and his or her company) remain. Some of these features include:

- Custom device user interface (e.g., plastic look-and-feel, form factor, ergonomics, knob and button placement, and overall size)
- Device functionality
- Quality (e.g., LCD resolution, audio speaker size, and digital-to-analog converter bit resolution)
- Component lead times and development quantity desired

- Power consumption

Figure 43 depicts an interactive session between a device designer and a back-end design tool. In the first step (step A), the designer enumerates the desired device characteristics as well as implementation metrics (e.g., two week lead time, four hour device life on a pair of C cell batteries) with which to gauge the device implementation. In the next step (step B), the tool returns implementation scores for each of the specified metrics. Usually, not all of the metrics will be met completely, and the designer will have to choose those metrics for which optimize. The designer will have to iterate through multiple implementations of the design by specifying new or optimized sets of metrics and characteristics until she is content with the implementation score. Finally, the designer must request the tool to output information necessary to successfully build the device.

Consider an example interaction between a device designer and the design tool. Suppose a designer must develop a set of 10 prototype devices to showcase to management prior to getting approval for mass deployment. She enumerates the desired device characteristics as well as the implementation metrics of two week lead time and four hour battery life on a pair of C cells. On the first iteration, the tool returns a score for one implementation that requires seven weeks of component sourcing lead-time but offers 6.3 hours of battery life. The designer must decide whether the additional five weeks of lead time merit the reduced power consumption of the implementation. Given that this is a prototype, she may choose to optimize for lead-time, and specifies so to the tool. On this iteration, the tool returns a score for a second implementation that requires just 10 days of component lead time, but increases power consumption such that only two hours of battery life can be sustained on a pair of C cells. Content with this score, the designer requests an EDIF-compliant netlist for that implementation that can simply be sent to a printed circuit board (PCB) fabrication and assembly facility for device manufacturing.

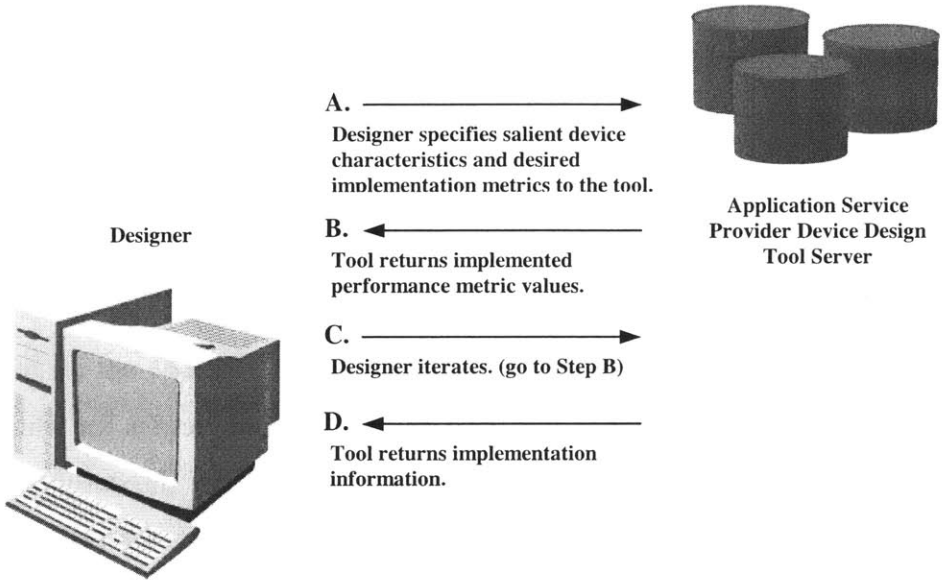


Figure 43: Desired interactivity between designer and backend server.

The described design tool and design flow can be achieved with the CSR architecture. The CSR architecture is in fact a greatest-common-denominator architecture in which the system resources common to most devices are CSR modules. Only the custom characteristics and I/O interactivity of the device must be designed and developed; the appropriate CSR modules, which form the basis for the device platform, are simply purchased.

5.2 Characteristic-based CSR Device Design

The CSR architecture structures device design around its configurable universal interface, and greatly simplifies the design and development cycle, while providing sufficient flexibility to support a wide assortment of devices. In fact, this structured design flow can be automated to a large degree and further simplified.

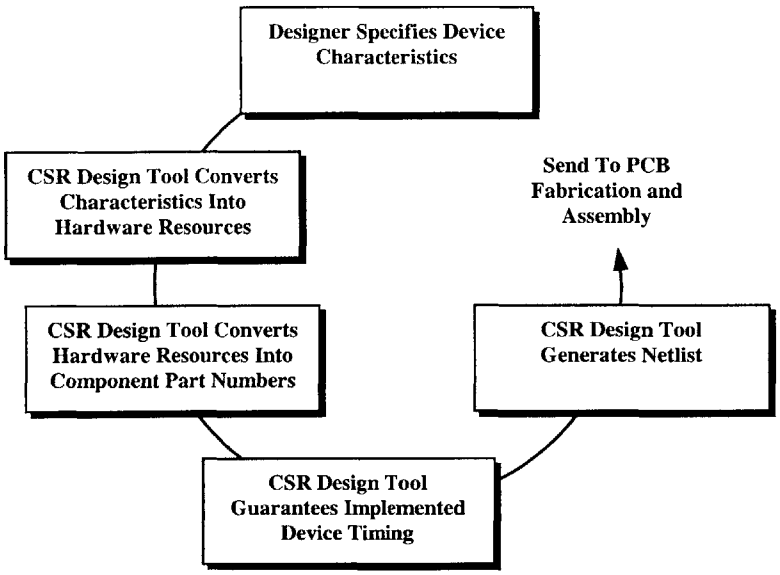


Figure 44: CSR-based device design flow and development steps.

Figure 44 depicts the steps involved in characteristic-based CSR device design. The designer must first specify the desired characteristics of the proposed device. In the next step, these characteristics must be mapped to simple hardware resources, such as audio speakers and digital-to-analog converters. Depending on the level of the initial characteristic specification, this mapping may be trivial or more complex. In the third step, the simple hardware resources must be realized through selection of appropriate off-the-shelf components. During this process, sets of simple hardware resources may be implemented through complex components. For example, four analog-to-digital converters (DACs) may be implemented by a single quad-DAC component, or a digital-to-analog converter and an analog-to-digital converter may be implemented by a single codec component. The selection of off-the-shelf components, including complex components, may be driven by specified benchmarks such as maximum cost-of-goods and minimum device battery life. In the next step, the implemented design must be verified for proper timing and quality-of-service with the CSR architecture and its configurable universal

interface. This step involves guaranteeing sufficient quality-of-service for not only the narrowpath channels but also for additional CSR modules (on two-socket CSR-IFB-based devices). The final step is to generate a useful representation for the implemented device. This may involve generating an EDIF netlist, a routed PCB Gerber file, or perhaps even delivery of a fully fabricated and assembled PCB.

The next sections more fully describe these five steps to characteristic-based CSR device design.

5.3 Device Characteristic Specification

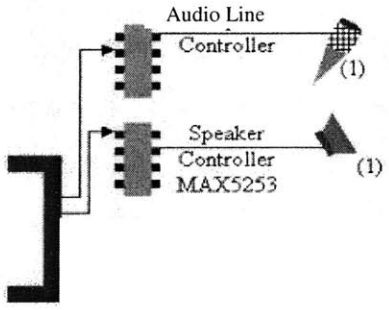
The first step in CSR-based systems design is to specify the general characteristics of the device. As shown in Figure 45, this may simply involve checking off a set of desired characteristics from an enumerated list. For example, consider a simple audio device capable of outputting audio through a speaker and is based on voice input through a microphone. The designer would simply select the Microphone and the Speaker checkboxes, which would result in the design tool generating not only the microphone and the speaker, but also any other components necessary to support data communication between the selected components and CSR modules. The schematic-style graphical representation of the components appear on the right of the checkboxes as feedback to the designer.

The characteristic specification must not only include information about the physical components of the device but also qualitative information. Such qualitative information includes resolution of LCD panels, which affects panel selection, and audio quality, which affects DAC and speaker selection. Qualitative information also affects bandwidth requirements, which in turn, affects the mix of modalities that can be supported by the IFB. For example, consider the bandwidth requirements for voice-quality audio versus CD-quality audio and even stereo CD-quality audio.

Feature and Part Selection

Use	Feature	Part	Help	Power
☐	Audio Microphone	Hitachi 44780	Help	
☒	Audio Line In	K1452052	Help	
☒	Audio Speaker	LCS-2416	Help	
☐	Audio Line Out	LN28RP	Help	
☐	LEDs	MR-11	Help	
☐	Motor Controller	G4UAP	Help	
☐	Infrared Obstacle Sensor	Mabuchi 280	Help	

LivePort Schematic Generation




```
<device_builder_design_specifications>

    <physical_characteristics type=AUDIO_OUTPUT_SPKR>
      <speaker_size spec="1">
      <audio_quality spec=CD>
    </physical_characteristics>

  <physical_characteristics type=AUDIO_INPUT_LINE>
      <audio_quality spec=voice>
    </physical_characteristics>

  <desired_lead_time spec="one month">
  <desired_quantity spec=10>
  <desired_size spec=min>

<device_builder_design_specifications>
```

Figure 45: Device builder characteristic selection and underlying representation.

Figure 45 also shows the underlying representation that is gathered from user input through the front-end GUI. The specification consists of two primary segments. The first segment is the physical characteristics specification, which details the desired physical user interface of the device, while the second segment is the process specification, which details the logistical issues of the device.

The physical characteristics specification shown in Figure 45 consists of two pre-defined types: AUDIO_OUTPUT_SPKR and AUDIO_INPUT_LINE. The AUDIO_OUTPUT_SPKR type takes data from CSR modules and outputs them through an audio speaker. The specifications for SPEAKER_SIZE and AUDIO_QUALITY are sub-types pre-defined for the type AUDIO_OUTPUT_SPKR. SPEAKER_SIZE specifies the approximate desired size of the physical audio speaker component, while AUDIO_QUALITY specifies the desired audio bit rate. CD is a pre-defined heuristic for CD-quality audio. Similarly, AUDIO_INPUT_LINE specifies the capture and digitization of audio signals not through a microphone but through a line-in jack. This type has its own set of pre-defined sub-types that are interpreted with respect to audio input as opposed to audio output.

The process specification shown in Figure 45 enumerates logistical information such as the number of devices that will be developed and the amount of time budgeted for their development. The desired_lead_time type specifies the maximum amount of time that is available until the delivery of finished goods is necessary. This could be for prototype development or volume product development. This and other process specifications are used to more appropriately select the components to implement the specified physical characteristics.

Once a desired set of device characteristics have been transformed into a set of simple I/O resources, the next step is to choose the actual off-the-shelf component part numbers and to appropriately place them on the CSR IFB.

5.4 Resource Mapping, Placement, and Quality-of-Service

Selection and placement of off-the-shelf components on CSR IFBs is primarily based on the following properties:

- Functionality: Components are selected that implement the functionality as specified by the characteristic specification. The Resource Mapper (as described in Section 5.4.1) maps

physical characteristics into simple resources that can be implemented by off-the-shelf components.

- Cost Function: Of the many off-the-shelf components that can implement a device's UI as specified by the characteristic specification, the actually selected set of devices minimize a designer-specified cost function. This cost function is determined using the process specification (as shown in Figure 45) as well as with information from the manufacturer about each component, such as unit cost and lead-time.
- Interface support: CSR computer modules must support the electrical interface of each selected component. Many off-the-shelf I/O components use SPI, Microwire, and 8-bit parallel interfaces, all of which are supported by the current implementation of the PC Card-based CSR architecture. Components are placed onto the appropriate tracks of CSR IFBs such that they match the pin-out and protocol engine positioning of narrowpath channels of CSR computer modules.

The following sub-sections further describe component selection and placement.

5.4.1 Resource Mapping

Resource mapping reduces specified device characteristics into simple resources. These simple resources, once enumerated, can be implemented by available off-the-shelf components.

Once a designer has specified a device characteristic, such as `AUDIO_OUTPUT _SPEAKER`, the Resource Mapper (RM) enumerates all of the resources necessary to appropriately interface with CSR modules on the back-end and to output audio waveforms through a speaker on the front-end. An `AUDIO_OUTPUT_SPEAKER` characteristic is mapped to an `AUDIO_OUTPUT` resource and an `AUDIO_SPEAKER` resource. Characteristic specifications are mapped to simple resources by using pre-defined heuristic tables.

Constraints between simple resources are also enumerated by the RM. Consider again the example characteristic specification of `AUDIO_OUTPUT_SPEAKER`, and the simple resources

AUDIO_OUTPUT and AUDIO_SPEAKER. One possible constraint between the two resources is impedance. An impedance matching specification avoids the selection of a digital-to-analog converter with 1 kΩ impedance and a speaker with 4 Ω impedance.

The RM also optimizes sets of simple resources into complex resources. For instance, the use of two single function AUDIO_OUTPUT resources can be optimized into a dual function AUDIO_OUTPUT resource. The component selection and placement step is then able to use a dual output digital-to-analog converter (DAC), instead of two single-output DACs.

The RM, however, does not eliminate simple resources that it optimizes into complex resources. This is because designer-specified characteristics, such as cost, lead time, and desired device quantity, that the RM does not take into consideration during resource mapping, may in fact be better satisfied by un-optimized resources. For instance, returning to the example of single and dual function DACs, if the desired lead time for a prototype device is better met by the use of two DACs instead of a single dual DAC, the simple resources should be used instead of the complex one.

The RM enumerates all possible combinations of simple and complex resources that can be used to implement the characteristic specification. The comparative analyses between multiple possible implementations are undertaken during the cost function analysis step.

5.4.2 Cost Function Analysis

The cost function analysis determines which of the multiple possible implementations of a characteristic specification is to be selected. The actual cost function is derived from the process specification, with values received from component manufacturers and distributors. Typical contributors to a device's cost function include component cost, power consumption, lead-time, and device size.

The following equations show the relationship between individual implementation decisions and the overall device cost function:

$$C = \sum_{i=o}^{n} c_i \times w_i$$

where C is the overall device cost function, c is the individual cost function for each of the n specified metrics and w is the weight signifying the importance to the designer of each metric. Next, we investigate the cost functions c of a few metrics.

During prototype development, perhaps the most important metric is lead-time. The path to quickly implementing a prototype of an envisioned device UI is oftentimes blocked with numerous re-designs as components cannot be sourced sufficiently fast. Although the devices that can be sourced quickly and are available at low volume may not be the optimal ones with respect to cost and power consumption, they nonetheless represent a viable means to realize the envisioned device UI and functionality. The cost function for lead-time $c_{LeadTime}$ is:

$$c_{LeadTime} = MAX(l_1, l_2, ..., l_k)$$

where l_i is the lead-time for the ith component of the device UI, and MAX represents the function that returns the maximum of all of its arguments.

Cost is also an important metric, as it is oftentimes the gatekeeper between prototypes and mass production. The cost of a device is comprised of the cost of the individual components that implement the device UI's functionality, the cost of the CSR module connectors, the cost of the IFB controller, the cost of the printed circuit board that houses all of the aforementioned parts, and the cost of assembly. The cost function for device UI cost c_{Cost} is:

$$c_{Cost}(n,q) = \sum_{i=0}^{n} k_i(q) + m + b + a(n,q)$$

where q is the desired quantity of devices that are to be built, k_i is the unit cost of the ith (of the n total) component, m is the cost of CSR module connectors together with the cost of the IFB controller, b is the cost of the PCB, and a is the cost of assembly. The cost of the PCB b is determined by the area (x x y) and the number of layers necessary to route all of the n components, as well as the material type. The cost of assembly a is based on the number of components to be populated as well as the quantity q of boards to be built.

The overall device cost function C is evaluated over each set of components (or implementation) that satisfies the resources enumerated by the RM. Although C is specified as an analytical equation, it is typically difficult to determine the single most efficient (with respect to the cost function) implementation. For instance, consider the following information:

- Designer-specified information: Lead-time < 30 days; Cost < $10
- Device implementation one: Lead-time = 32 days; Cost = $5.74
- Device implementation two: Lead-time = 23 days; Cost = $9.49

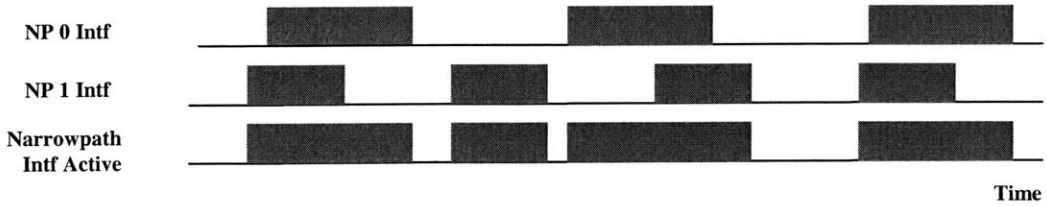


Figure 46: The time utilized for communication with all of the narrowpath components trade-off with the time available for module-to-module communications.

Although implementation two meets both of the specifications of the designer, implementation one may actually be more attractive. By returning the cost function results for more than just a single implementation of a device, the designer is free to choose the actual optimal implementation.

5.4.3 Quality-of-Service Guarantees

The design environment guarantees that peak bandwidth requirements between CSR modules as well as between CSR computer modules and device "shell" components are met over the time-multiplexed bussed CSR IFB environment. In order to provide such quality-of-service guarantees, the design environment determines the amount of time that is utilized by all narrowpath channels, which then allows it to determine the amount of bandwidth that is available for module-to-module communications.

There is a direct relationship between the bandwidth that is used for module-to-module communications and the bandwidth that is used for narrowpath communications. This relationship is depicted by the following equation:

$$BW_{Module} = 1 - BW_{NP} + c$$

where BW_{Module} and BW_{NP} are the bandwidth used for module-to-module communications and the bandwidth used for narrowpath communications, respectively, and c is the amount of time for which the IFB is idle and communications are stopped. Communications are stopped either

because communications during that time period is unnecessary (e.g., sufficient bandwidth already exists) or, as was discussed in Section 4.4.1, the time slot is too small and insufficient to support even a single module-to-module transaction.

Each narrowpath channel is modeled as either asynchronous or periodic. A channel is modeled as asynchronous if it is unlikely to be activated multiple times in a periodic or semi-periodic fashion. For example, the Play button of an audio device is likely to be asserted infrequently, and can be modeled as an asynchronous event. On the other hand, a button that is used as a video game controller (e.g., a fire button) is usually activated multiple times, and oftentimes as fast as the user can possibly press the button. This sort of a button is modeled as periodic, with the periodicity determined by the worst-case frequency an user can assert the button. For example, assuming a minimum reaction time of 100 milliseconds, the worst-case frequency of the video game button is 10 Hz. Each narrowpath channel is formalized as follows:

$$a_k(t) = \begin{cases} 1 \text{ when } \mathrm{Rem}(\frac{t}{p_k}) < c_k \\ 0 \text{ otherwise} \end{cases}$$

$$c_k(sync) = b_k \times \frac{1}{clk_k}$$

$$c_k(async) = l_k \times n_k$$

$$p_k = \frac{1}{f_k}$$

where $a_k(t)$ relates whether the kth narrowpath channel is active at time t, and $Rem(x/y)$ is the function that returns the integer remainder of the division of x by y. c_k is the cycle time for the kth narrowpath channel. c_k is determined for channels using a synchronous protocol by the product of b_k, the bit width of each packet, and clk_k, the clock frequency of the synchronous protocol, and for channels using an asynchronous protocol by the product of l_k, the maximum length of each

cycle, and n_k, the number of cycles per transaction. The periodicity of each narrowpath channel is determined by the inverse of the channel communication frequency f_k.

Manufacturers' datasheets for components together with heuristics provide information sufficient with which to determine the value of $a_k(t)$ for all narrowpath channels. For instance, c_k for a SPI-based component is fully specified by the number of bits per SPI packet and the frequency of the serial clock. Periodicity, on the other hand, usually cannot be determined from component datasheets as the same component may be used for many different applications. Accordingly, the design environment uses pre-defined heuristics for determining values of narrowpath periodicity p_k. For example, typical packet delivery rates for implementing CD-quality audio is 44 kHz, while that for voice-quality audio is 11 kHz. It is important to note that ranges are usually provided for datasheet values as well as for application-specific heuristics. A SPI-based digital-to-analog component may consistently use 16-bit packets, but may support serial clock frequencies up to 10 MHz. The ranges in value of each component or application-specific heuristic lead to a range in value for $a_k(t)$, which in turn affects the amount of time available for module-to-module communications. This is quantified by the following equation:

$$p_{k-\min} \cdot c_{k-\max} \leq m_k \leq p_{k-\max} \cdot c_{k-\min}$$

where m_k is the range of time available for module-to-module communications if the kth narrowpath channel were the only one active. c_k and p_k were defined above.

The design environment evaluates the various potential implementations (sets of narrowpath components) as output from the RM to determine the amount of time that is available for module-to-module communications. If sufficient bandwidth is not available for module-to-module communications, the equation for m_k shows how narrowpath components and application-specific heuristics can be used to trade-off narrowpath component quality or speed for more module-to-module communication time.

After iterating through the potential implementations output by the RM and the designer is content with the properties of the selected implementation, the design environment generates a netlist representation for the device UI that can be used for simple device manufacturing. This is discussed in the next section.

5.5 Netlist Generation and System Configuration

Once components matching the desired characteristics have been selected, bandwidth and quality-of-service have been guaranteed, a system netlist can be generated. A netlist provides a textual representation and specification of pin-to-pin connections between all components. A netlist in the popular EDIF [22] format combined with layout information (commonly available for commodity components) can be automatically routed for printed circuit board fabrication and component population.

Information about the device "shell" is also stored in read-only memory on the IFB. This allows general CSR modules to be connected to custom device "shells" and have the ability to communicate with and control each of the "shell's" components. This information may be stored in ROM as software readable data or it may simply be a short identifier that can be de-referenced (possibly over a network) to access the actual device-specific information. The salient aspects of this information are the components used to implement the device "shell", the IFB architecture, and the quality-of-service model used to guarantee performance.

5.6 System Limitations and Third-Party Integration

The CSR-based device design environment leverages the abstraction between CSR modules and the device UI to simplify the selection of off-the-shelf components that implements a desired device UI functionality. The key capability of this design environment is to create a netlist that interconnects off-the-shelf components from a set of high-level device characteristics and design metrics, such as component unit cost and lead time. Reducing this netlist into a manufacturable

printed circuit board (PCB) introduces a set of problems and issues that are not addressed. We discuss these issues in this section.

PCB design comprises a number of distinct pieces, including electronics and mechanical design, fabrication, electromagnetic interference (EMI) and radio frequency interference (RFI) certification, testing, quality assurance, field service, and repair. Issues, such as interconnect delay, which had a small effect on PCB routing decisions are becoming more important as clock speeds continue to increase (and delay budgets continue to decrease). Electrical noise, primarily from crosstalk, is increasing as board sizes shrink and traces are routed increasingly close together. The use of higher density packages, such as ball grid arrays (BGAs), are exacerbating simultaneous switching noise and ground bounce [29].

Proper PCB design is the hub of a successful system, and much work has been done to simplify and automate this process. Hardware design has typically been a sort of "black magic" – with many decisions based on rules-of-thumb, experience, and "gut feelings". Design environments and automation tools have been made that try to capture and encapsulate many of these rules-of-thumb and designer experiences such that constraints can be specified as electrical specifications (e.g., clock speed and signal setup and hold times) that are automatically converted into PCB layering decisions, floorplans, and trace lengths [29].

Design for manufacturing (DFM) places another set of constraints on PCB design. Examples of DFM constraints include placing all capacitors on the reverse side of a PCB so as to facilitate board assembly, and thermal issues related to soldering components such as heat sinks. Essentially, DFM constraints concern PCB design techniques that enable high-quality, volume PCB manufacturing and assembly for reasonable costs.

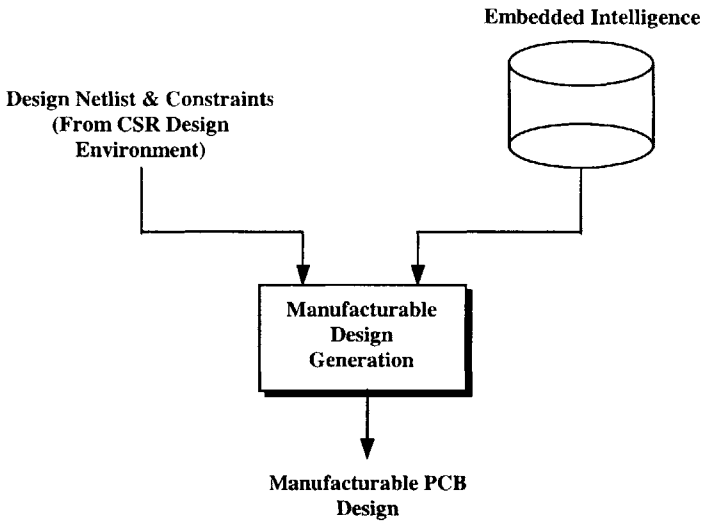


Figure 47: Extensions to the CSR-based Design Environment for generating manufacturable PCB systems.

Since many engineers do not have the knowledge and experience to properly design PCBs and much time and money is wasted with PCB iteration and re-design, vendors are introducing tools and constraint management systems that incorporate captured knowledge for reducing netlists to manufacturable PCBs. In this scheme, engineers enter high-level electrical constraints that are automatically converted by the software tool into mechanical designs. For example, crosstalk and noise constraints can be specified in millivolts (mVs), which can be determined as the minimum of the maximum noise (in mV) tolerable by any component placed on the PCB. Xynetix's EDANavigator uses such a constraint management system that converts high-level constraints into rules for thermal manufacturability, as well as for electromagnetic and radio frequency interference [29]. Similarly, Cadence's SPECCTRA [33] and SPECCTRAQuest [34] tools support floorplanning, signal integrity analysis, auto-routing, and post-layout verification.

Reducing netlists to manufacturable PCBs is a problem inherent to all hardware designs. Given the specialized skills and design tools required to properly build PCBs, many companies are outsourcing the actual PCB development, and instead focusing internal efforts on innovative system designs and netlist creation. The CSR design environment facilitates the design,

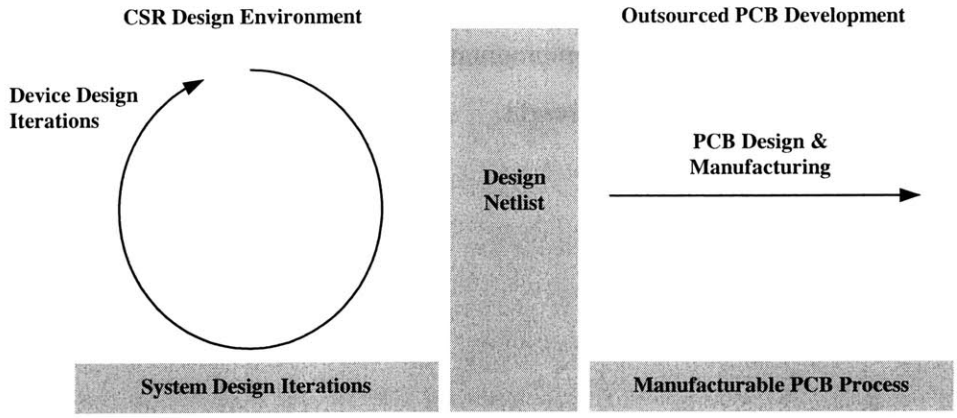


Figure 48: The relationship between the CSR design environment and a manufacturable PCB design process.

development, and iterations of devices based on the CSR interface abstraction and implements device user interface functionality with off-the-shelf components. The CSR design environment generates a device design netlist that can then be reduced to a manufacturable PCB. The relationship between the CSR design environment and the manufacturable PCB design process is shown in Figure 48.

5.7 Discussion

This chapter investigated opportunities to simplify CSR-based device design. Since the CSR architecture is based on using pluggable computational resource modules, device design consists of developing device UIs or "shells". Manufacturers and consumers can then optimize the performance of each "shell" with respect to user preferences for metrics including, cost, power consumption, and interactive performance.

This chapter specified an automated design environment for CSR-based devices that greatly simplifies device design and development, and in fact, makes it appropriate for non-technical designers. The high-level interactivity between the designer and the design environment is centered around user interface design and not systems design. The physical design of the user

interface is followed by a set of iterative optimization phases that allow the designer to direct the implementation of the UI such that the implementation is within desired specifications, including lead time, cost, power consumption, and weight.

Chapter 6
Conclusion

As digital content subsumes analog content, and more and more networks connect content repositories together, consumers are faced with information overload that cannot be mitigated through a single content access device and user interface as provided by status quo PCs. Recognizing the need for multiple content access devices, the CSR architecture proposes the use of simple building-blocks for easily developing and maintaining devices for accessing network-based content.

As value migrates away from the devices themselves and toward network content, the devices and the device user interface, in particular, becomes the enabler or gate-keeper for the delivery of content. Accordingly, network devices must be inexpensive. Moreover, in order to support a variety of user interfaces and interactive paradigms – including mobility – network devices must cater to multiple metrics, including power consumption, battery life, interactive performance, security, size, and weight. Since these metrics are oftentimes at odds with one another, user preferences can be used to determine the most attractive system design on an individual basis instead of building one-size-fits-all devices.

To this end, the CSR architecture presents an abstraction for device development that lowers development time and costs, while, at the same time, catering to individual user preferences for the actual make-up of the device's computational resources. The CSR architecture consists of a set of pluggable resource modules that can be composed together to form a desired computational system, and then connected to a device UI to realize a complete, custom device. CSR modules can be connected to any device UI (or "shell") and, thus, are high-volume, while the device UIs themselves are custom and reasonably low volume. By composing together a set of CSR modules, a device platform can be built that meets user preferences, and eliminates one of the most time consuming aspects of system development – porting and configuring the software environment to the hardware.

On the other hand, custom device UI design is simplified as it must only be built around the well-defined and structured CSR interface without having to delve into the technical innards of the system platform. Independent, standards-based narrowpath channels implemented by CSR modules support the use of off-the-shelf commodity components with which to build the device UI's functionality. The abstraction between device UI and device platform presented by the CSR architecture together with the ability to use off-the-shelf components facilitate device development. The CSR design environment specifies a high-level design automation scheme that leverages the abstraction presented by the CSR architecture to enable designers to interactively design a complete, custom device – from idea to system netlist.

The CSR architecture supports the rapid development of custom network devices that cater to individual preferences for system performance. By minimizing the risk, development time, and costs associated with custom device development, the CSR architecture and the CSR design environment enable the development of a variety of devices with innovative user interfaces that simplify interaction with the growing number and type of digital content repositories.

6.1 Future Work

The concepts and ideas presented herein can be enhanced and extended with future research and development work. One of the greatest challenges to CSRs is to achieve broad usage and applicability. To this end, CSRs should track battery usage. Any device using batteries for electrical power should use CSR modules for computational and peripheral power. This requires that CSRs be correspondingly smaller in size and cheaper in cost.

One means to accomplish both is to base the electrical interface of CSR modules on emerging high-speed serial interface such as Universal Serial Bus (USB) [4] and IEEE 1394 FireWire [2]. The serial interface reduces connector size as well as the interface ASIC pin count and cost. The simplicity of using multiple, independent narrowpath channels can be maintained by using a de-multiplexor on the IFB that explodes the serial interface of CSR modules into multiple narrowpath channels. Multiple CSR modules can be supported by providing a point-to-point interface on the IFB between each pluggable module and the IFB controller. The limited number of pins required for CSR modules implementing serial interfaces make a point-to-point solution more attractive from a cost as well as from a physical area perspective. The primary disadvantage of this serial-interface-based architecture at this time is the lack of available peripheral modules in the same or similar form factor.

As CSR module characteristics become more appropriate to track battery usage, CSR modules and batteries may even be integrated together to form a single structured building block that provides electrical, computational, and peripheral resources. The batteries in these integrated modules can be re-charged when the device is the on its docking-station, or simply replaced altogether, given a modular architecture between the batteries and the computational resources of each CSR module.

At the system platform level, more work needs to be done to understand the types of CSR modules that are attractive to typical users and their usage patterns. CSR modules reverse the one-size-fits-all approach to architecture and suggest a tighter coupling between system

architecture and individual usage patterns. This presents an opportunity to understand typical content access and application usage patterns as well as the technology necessary to sustain these patterns (e.g., SSL for encryption), and to create a set of system-on-chip (SoC) designs that are targeted these specific patterns.

References

1. Abnous, A. and J. Rabaey. *Ultra-Low-Power Domain-Specific Multimedia Processors.* in Proceedings of *Proceedings of the IEEE VLSI Signal Processing Workshop.* 1996. San Francisco.
2. Anderson, D., *Firewire System Architecture: IEEE 1394a.* 2nd ed. PC System Architecture Series. 1998: Addison-Wesley. 400.
3. Anderson, D., *PCMCIA System Architecture: 16-bit PC Cards.* 2nd ed. PC System Architecture Series. 1995: Addison-Wesley.
4. Anderson, D., *USB System Architecture.* PC System Architecture Series. 1997: Addison-Wesley. 321.
5. Anderson, D.M., *Agile Product Development for Mass Customization: How to Develop and Deliver Products for Mass Customization, Niche Markets, JIT, Build-to-Order, and Flexible Manufacturing.* 1997: McGraw-Hill.
6. Armstrong, J.R. and F.G. Gray, *VHDL Design Representation and Synthesis.* 2nd ed. 2000: Prentice Hall. 651.
7. Axelson, J., *USB Complete: Everything You Need to Develop USB Peripherals.* 1999: Lakeview Research. 398.
8. Axelson, J. and J.L. Axelson, *Parallel Port Complete.* 1997: Lakeview Research. 304.
9. Axelson, J.L. and J. Axelson, *Serial Port Complete.* 1998: Lakeview Research. 304.
10. Bhatnagar, H., *Advanced ASIC Chip Synthesis: Using Synopsys Design Compiler and Primetime.* 1999: Kluwer Academic.
11. Broderson, R., The Case Against Von Neumann Architectures in System-on-a-Chip Design, MIT Laboratory for Computer Science Distinguished Lecture Series, 2000.
12. Chatterjee, S. *The ModuleC Network Architecture: A Novel Approach to Computing Through Information Appliances.* in Proceedings of *IEEE International Symposium on Consumer Electronics.* 1997. Singapore.
13. Chatterjee, S. *SANI: A Seamless and Non-Intrusive Framework and Agent for Creating Intelligent Interactive Homes.* in Proceedings of *ACM Conference on Autonomous Agents.* 1998. Minneapolis/St. Paul, MN.
14. Chatterjee, S. *Towards a MASC Appliances-based Educational Paradigm.* in Proceedings of *ACM International Symposium for Applied Computing.* 1998. Atlanta, Georgia.
15. Chatterjee, S. *Towards Rapidly Deployable Intelligent Environments.* in Proceedings of *AAAI Spring Symposium on Intelligent Environments.* 1998. Stanford, CA.

16. Chatterjee, S. and S. Devadas. *The MASC Composable Computing Infrastructure for Intelligent Environments*. in Proceedings of *IEEE Industrial Electronics Society Conference*. 1999. San Francisco, CA.

17. Chatterjee, S. and S. Devadas, *MASC: An User-Embeddable Hardware Platform and Infrastructure for Information Appliances*, 1999, Massachusetts Institute of Technology Laboratory for Computer Science: Cambridge.

18. Cmelik, R.F., *et al.*, *US6031992: Combining hardware and software to provide an improved microprocessor*, 2000, Transmeta Corporation.

19. Dally, W. *Tomorrow's Computing Engines*. in Proceedings of *International Symposium on High-Performance Computer Architectures*. 1998. keynote speech.

20. Dally, W.J. and J.W. Poulton, *Digital Systems Engineering*. 1998: Cambridge University Press. 600.

21. Edwards, W.K., *Core JINI*. The Sun Microsystems Press Java Series. 1999: Prentice-Hall. 772.

22. Electronics Industry Association, *EDIF Version 4.0*. 1996: ANSI/EIA 682-1996 Standard.

23. Fallah, F., P. Ashar, and S. Devadas. *Simulation Vector Generation from HDL Descriptions for Observability Enhanced-Statement Coverage*. in Proceedings of *Design Automation Conference*. 1999.

24. Fallah, F., S. Devadas, and K. Keutzer. *OCCOM: Efficient Computation of Observability-Based Code Coverage*. in Proceedings of *Design Automation Conference*. 1998.

25. Fine, C.H., *Clockspeed: Winning Industry Control in the Age of Temporary Advantage*. 1998: Perseus. 288.

26. George, V., H. Zhang, and J. Rabaey. *Low-energy FPGA design*. in Proceedings of *Proceedings of ISLPED*. 1999.

27. Harris, E.P. and e. al. *Technology Directions for Portable Computers*. in Proceedings of *Proceedings of the IEEE*. 1995.

28. Havinga, P.J.M. and G.J.M. Smit. *Minimizing Energy Consumption for Wireless Computers in Moby Dick*. in Proceedings of *IEEE International Conference on Personal Wireless Communication*. 1997.

29. http://206.168.2.242/Editorial/1998/03//asic/398ASPCB.HTM, *Computer Design Editorial: PCB Design Becomes Focus of Entire Design Process*, 1998.

30. http://developer.intel.com/design/strong/, *StrongARM Developers Page*.

31. http://developer.intel.com/platforms/enterprise/, *Enterprise Computing Platforms*.

32. http://ergo.3com.com/ergo/html/homepage.html, *3Com Ergo Appliances*.

33. http://pcb.cadence.com/pcb/specctra/, *SPECCTRA*, . 2000.

34. http://pcb.cadence.com/pcb/specctraquest/, *SPECCTRAQuest*, . 2000.

35. http://www.annapmicro.com, *WildCard Reconfigurable Processor Card*, , Annapolis Micro Systems, Inc.

36. http://www.cs.berkeley.edu/~neefe/ntu.fa98/liem.project.html, *A Comprehensive Look at IO Buses*, 2000.

37. http://www.media.mit.edu/pia/Research/Hyphos, *Hyphos: A Wireless, Self-Organizing Network*, 1999.

38. http://www.microsoft.com/WindowsME, *Microsoft Windows Millennium Edition*, . 1999.

39. http://www.palm.com/products/index.html, *The Palm Personal Digital Assistant*, .

40. http://www.partfolio.com/JIT.html, *The Ideal Manufacturing Scene for Your Business*, . 2000.

41. http://www.sei.cmu.edu/ata/ata_init.html, *Software Architecture and the Architecture Tradeoff Analysis Initiative*, 2000.

42. http://www.sei.cmu.edu/cbs/icse99/papers/16/16.htm, *Component Based Software Engineering: A Broad Based Model is Needed*, 2000.

43. http://www.sun.com/jini, *Sun Microsystems Jini Connection Technology*, . 1999.

44. http://www.transmeta.com/crusoe/, *Transmeta Crusoe Processor.*

45. http://www.xilinx.com/partinfo/databook.htm, *Xilinx Programmable Logic Products Databook 2000*, 2000.

46. IEEE, *IEEE Standard 1284-1994: IEEE Standard Signaling Method for a Bi-directional Parallel Peripheral Interface for Personal Computers*. 1994: IEEE.

47. IEEE, *IEEE Standard for a High Performance Serial Bus*. 1998: IEEE.

48. Katz, R.H., *Contemporary Logic Design*. 1993: Addison-Wesley. 699.

49. Kipisz, S.M., B.E. Moore, and D.L. Beatty, *PC Card/PCMCIA: Software Developer's Handbook*. 2nd ed. 1999: Peer to Peer Communications. 450.

50. Kusse, E. and J. Rabaey. *Low-energy Embedded FPGA Structures*. in Proceedings of *International Symposium on Low Power Electronics and Design*. 1998.

51. Larson, L.E., *Radio Frequency Integrated Circuit Technology for Low-Power Wireless Communications*, in *IEEE Personal Communications*. 1998. p. 11-19.

52. Lorch, J.R. and A.J. Smith, *Software Strategies for Portable Computer Energy Management*, in *IEEE Personal Communications*. 1998. p. 60-73.

53. Micheli, G.D., *Synthesis and Optimization of Digital Circuits*. 1994: McGraw Hill College Division. 576.

54. Moorby, P.R. and D.E. Thomas, *The Verilog Hardware Description Language*. 4th ed. 1998: Kluwer Academic.

55. Mori, M.T. and D.W. Welder, *The PCMCIA Developer's Guide*. 3rd ed. 1999: Sycard Technology. 700.

56. Navabi, Z., *Verilog Digital System Design*. 1999: McGraw Hill. 500.

57. Newman, M. and J. Hong, *A Look At Power Consumption and Performance on the 3Com Palm Pilot*, University of California at Berkeley: Berkeley, California.

58. Olive, R., Personal Communication, 1998.

59. Ostwald, P.F., *Engineering Cost Estimating*. Third Edition ed. 1992: Prentice Hall. 576.

60. PCMCIA, *PC Card Standard Release 7.0: The Definitive PC Card Specification*. 2000: Personal Computer Memory Card International Association (PCMCIA).

61. PICMG, *PICMG 2.0 R3.0: CompactPCI Core Specifications*. 1995: PCI Industrial Computer Manufacturers Group.

62. Sahler, J.T., Personal Communication, 2000.

63. Shanley, T. and D. Anderson, *Cardbus System Architecture*. PC System Architecture Series. 1996: Addison-Wesley. 407.

64. Shanley, T. and D. Anderson, *PCI System Architecture*. 4th ed. PC System Architecture Series. 1999: Addison-Wesley. 787.

65. Srinivas, N., Personal Communication, 2000.

66. Stewart, R.D., R.M. Wyskida, and J.d. Johannes, eds. *Cost Estimator's Reference Manual*. 2nd ed. 1995: New York.

67. Venkatramani, A., M. Narasimhan, and R. Nagarajan, *Quantifying the Costs of Universal Encryption*. 2000: Austin, Texas.

68. Wan, M., *et al.*, *Design methodology of a low-energy reconfigurable single-chip DSP system.* VLSI Signal Processing, 2000.

69. Ward, S., *et al. The NuMesh: A Modular, Scalable Communications Substrate*. in Proceedings of *International Conference on Supercomputing*. 1993.

70. Weiser, M., *The Computer for the Twenty-First Century*, in *Scientific American*. 1991. p. 94-104.

71. Weiser, M., *Some Computer Science Issues in Ubiquitous Computing*, in *Communications of the ACM*. 1993.

72. Weiser, M., *Ubiquitous Computing*, in *IEEE Computer "Hot Topics"*. 1993.

73. Weiser, M., *The World Is Not a Desktop*, in *ACM Interactions*. 1993.

74. Woesner, H., *et al.*, *Power-Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective*, in *IEEE Personal Communications*. 1998. p. 40-48.

75. www.ampro.com, *AMRPO Gemini Backgrounder*, . 2000.

76. www.xilinx.com, *ASIC Alternatives*, . 2001.