Multiple Function Interface Design Challenges

National Semiconductor
Application Note 976
January 1995

- Bus Mastering and DMA
- Accessing On Card Memory Resources
- Accessing System Memory Resources
- Power Management
- Common Memory Access

1.0 GOALS OF THE PCMCIA MULTI-FUNCTION PC CARD SPECIFICATION

The PCMCIA Standard Release 2.1 addresses a standardized technique to bring added functionality to a portable computer system using a PC® Card. This has enabled applications to be added or removed from a portable system at the discretion of the user without removing the system cover. Popular applications for PC Cards include those traditionally found as an add in card on an IBM® Compatible ISA system such as a Network Controller, Modem, Sound or Memory function. Unlike desktop computer systems, a portable system typically contains only one or two PCMCIA expansion slots. This has driven a need to include more than one function in a PC Card. Adding this multi-function capability, however, has proven complex since the PCMCIA Standard only addresses single function operation. In addition, adding advanced functions in a PC Card poses a number of design challenges since the Standard does not have key hardware capabilities found in ISA. This paper addresses how to solve major design issues in the development of a multi-function PC Card that are not sufficiently addressed by the current Standard. Topics discussed include the Multi-Function Extension, bus mastering and DMA devices, power management, and common memory access. The techniques and issues presented have been addressed by National in the implementation of the PCM16C00VNG Multi-Function Interface IC. This chip implements the Multi-Function Extension and supports up to 2 I/O functions on a PC Card. *Figure 1* shows a block diagram of a multi-function card using the PCM16C00VNG.

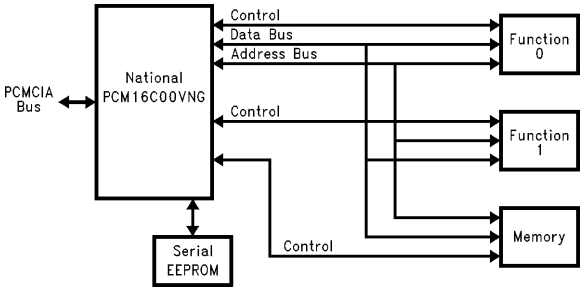
2.0 PCMCIA MULTI-FUNCTION EXTENSION

Before the proliferation of portable computer systems such as laptops or notebooks, the ISA expansion bus was used in IBM Compatible systems for adding functionality. The original IBM PC® and PC/XT® could support 8-bit ISA expansion boards. These contained signaling to handle a wide variety of applications by including an 8-bit data bus, 20-bit address bus, and 6 usable interrupt lines. Both Memory cycles and I/O cycles were permitted. The IBM PC/AT® introduced a 16-bit ISA expansion bus that provided a 16-bit data bus, 24-bit address bus, and 11 interrupt lines. The system could perform 8-bit accesses to the new cards for backward compatibility. Typically, these systems contained 4–6 expansion slots.

For portable systems, PCMCIA has allowed expansion capability through the use of PC Cards. Unlike traditional desktop machines with ISA add-in capability, PCMCIA machines typically only have 1 or 2 slots, which has driven a need to have two or more functions on a single PC Card. This creates the need to address two I/O functions, share a single interrupt line, and configure I/O functions individually on the card while preserving existing client drivers to the largest extent possible. The goals of the architecture are shown in Table I.

TABLE I. Goals of the Multi-Function Extension

Tie Function Specific Information to CIS Configuration Entries
Allow Independent use by Generic Enablers
Avoid Inter-Client Communication
Develop an **-IREQ** Sharing Protocol that allows function specific software to be unaware that other functions are on the card
No change to Socket Adapter Hardware
Minimal impact on system software (particularly legacy applications)



TL/F/11687–1

FIGURE 1. Block Diagram of a Multi-Function PC Card Using the National® PCM16C00VNG

RRD-B30M75/Printed in U. S. A.

To achieve these goals, the following hardware has been implemented in the National PCM16C00VNG Multi-Function interface IC. The implementation conforms to the Multi-Function Extension.

- Separate function configuration registers for each function
- Enhanced Interrupt protocol mechanism
- Base and limit registers for each function

The Function Configuration Option Registers (FCORs) contain the *SRESET, LevIREQ,* and *Function Configuration Index* similar to the current PCMCIA Standard. The paradigm, however, is that these bits now apply to a function and not an entire card. For example, writing a one (1) to the *SRESET* bit will reset the function, not the card. The PCM16C00VNG accomplishes this by passing a **RESET** signal to each function. The *LevIREQ* bit indicates the type of interrupts (level or pulse) that are generated during an interrupt event on the PCMCIA **-IREQ** line. Since there is only one *LevIREQ* bit per FCOR, all FCOR *LevIREQ* bits are aliased together. Within the *Function Configuration Index,* the lower three bits are defined by the Multi-Function Extension. They are *Enable Function, Enable Base and Limits Registers,* and *Enable IREQ* routing. The function is enabled by setting the *Function Enable* bit to a one (1). The PCM16C00VNG will switch the PCMCIA bus interface mode from a memory interface to an I/O interface once one function's FCOR *Function Enable* bit is set to one (1). This is necessary since there is only one PCMCIA interface. The *Enable Base and Limits Registers* bit in each FCOR's *Function Configuration Index* informs the PCM16C00VNG to qualify a PCMCIA I/O transaction based upon its address by testing it against the Base and Limits Registers. This type of testing is required to either enable the appropriate function chip select or to reject a bus cycle. While the host adapter may be used to create address windows that pass system addresses to the PC Card, the PC Card interface would have no standardized means of determining which function should be addressed. This is solved by the presence of Base and Limit Registers. The *Enable IREQ* routing bit is used as a mask in allowing a function interrupt to generate an **-IREQ** interrupt. If this bit is cleared (0), a function asserting its interrupt input to the PCM16C00VNG will not generate an interrupt event on **-IREQ**.

The Function and Configuration Status Registers (FCSRs) now replace the older Configuration Status Register. The *Changed* bit now is set only when the corresponding Function Pin Replacement Register (FPRR) changes state. Likewise the *SigChg* bit is function context sensitive. *IOIs8* is unused on the PCM16C00VNG. In the PCM16C00VNG implementation, the **-SPKR** signal may be driven any time an *Audio* bit is set in the FCSR of a function that has been configured (*Enable Function* bit is set to one (1) in its FCOR). The *PwrDn* bit is function context sensitive and will directly control a power control output pin from the PCM16C00VNG for that function. This control overrides the control provided by the PCM16C00VNG power management unit described later in this document.

One of the most challenging circuits in the PCM16C00VNG's implementation of the MultiFunction Extension is the interrupt protocol mechanism. This uses the *Intr* and *IntrReset* bit in each FCSR. The *Intr* bit always represents the value on the PCM16C00VNG's interrupt input signal from that function when read. If *IntrReset* is cleared to a zero (0), writing a value to the *Intr* bit shall do nothing. This corresponds to the Release 2.10 PCMCIA Standard in which *IntrReset* is a *reserved* field. When *IntrReset* is set to one and a value of zero (0) is written to any FCOR's *Intr* bit, the PCM16C00VNG inspects all function interrupt input pins whose *Enable IREQ* bit is set in its FCOR for a pending interrupt.

If one exists, the PCM16C00VNG shall generate another interrupt. This requires only one minimal size attribute window in the host adapter to be open in cases where FCOR's are spaced apart. In pulse mode, the PCM16C00VNG generates a high-low-high pulse on **-IREQ**. See *Figure 2.*

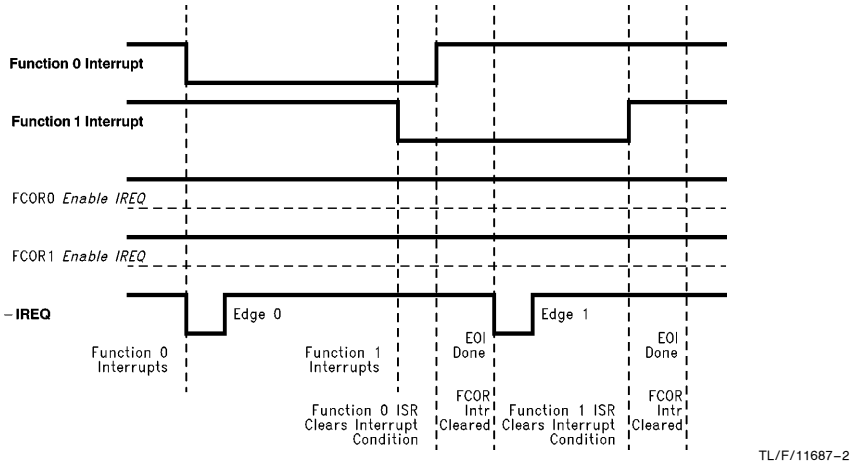


FIGURE 2. Multi-Function Extension Pulse Mode Interrupt Timing Implementation Using Interrupt Acknowledge Protocol

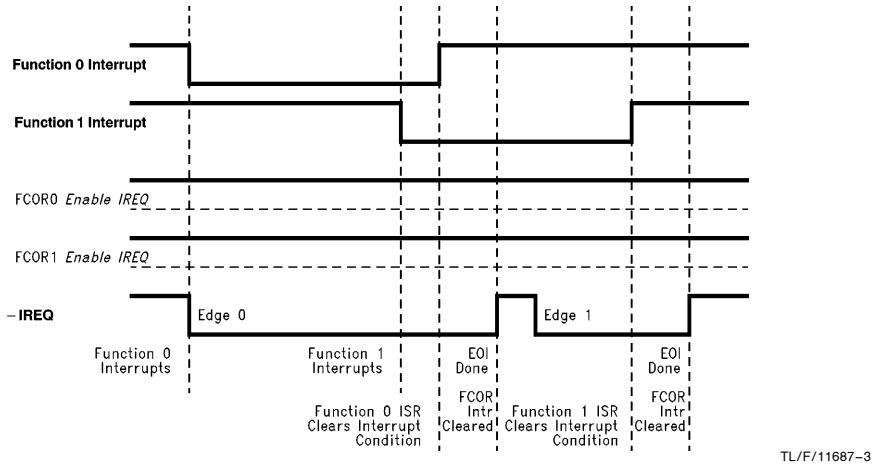


FIGURE 3. Multi-Function Extension Level Mode Interrupt Timing Implementation Using Interrupt Acknowledge Protocol

The Function Base and Limits Registers have been implemented in the PCM16C00VNG. The PCM16C00VNG contains fully implemented Function Limit Registers and the lower 2 Bytes of the Function Base Address Registers. This covers the 64 kByte I/O space of 80x86 systems. See *Figure 4.*

PC Card I/O Address Space

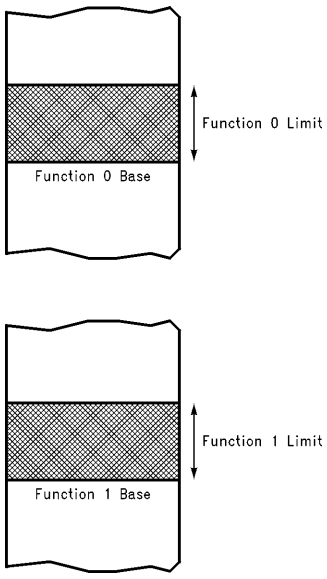


FIGURE 4. PCM16C00VNG I/O Window Decoding for Bus Cycle Steering to Appropriate I/O Function

3.0 BUS MASTERING AND DMA

The ISA expansion bus in the original IBM PC and PC/XT offered 4 DMA channels. The IBM PC/AT and most ISA based systems today allow cards with either 4 or 7 DMA channels. In addition, 16-bit ISA cards may become bus masters using the **-MASTER** signal. While the bus mastering utility on ISA is limited, the architecture supports it. Direct Memory Access is used even though its performance is questionable with respect to having the CPU perform block data moves itself. On EISA or MCA based systems, bus mastering is fully supported and utilized. An example of such utilization is high performance LAN devices that feature 32-bit bus-mastering interfaces. These devices can send and receive network data packets with a much smaller burden on the CPU. They do this by being able to move data packets in and out of memory on their own. This allows concurrent operation with the CPU. With modern architectures that use a hierarchical bus approach where busses are interconnected by bridges, true concurrent operation can occur. Even in older systems, the advantages of improved system utilization are present due to cycle stealing. This technique is possible due to instruction pre-fetch queues found in most CPU bus units.

On PCMCIA Revision 2.10 PC Cards there is no standardized DMA or bus-mastering capability support in the PC Card Specification. While there is DMA, it requires changes to the host adapter of which most of the current installed base does not have. The PCMCIA Cardbus committee is working on a 32-bit bus mastering proposal, but, this again will require host adapter changes and major card design changes due to its PCI derived protocol. Therefore, there is a need for legacy bus-mastering or DMA functions using the current PC Card specification. Therefore, to support N functions on a card, it may be desirable for some sub-set of these functions to be bus mastering. They may desire access to either on card memory resources (perhaps within another function) or system memory resources.

3.1 Accessing On Card Memory Resources

One technique of solving this problem is using a split bus (or point to point) architecture for bussing on the PC Card itself. This offers a simpler control mechanism, but requires bus multiplexing in addition to a more complex PC Card board layout. This generates higher silicon pin counts and adds to the cost of the card. The best technique uses a multi-access scheme to share a single bus resource. Therefore, a single data, address bus pair can handle all data traffic within the PC Card. Doing so, however, requires a controller mechanism that grants access to functions upon making a request. National has found that the controller or bus arbiter is best placed in the interface silicon. For a multi-function card, this arbiter must be capable of handling N+1 devices that may request the bus. These are the host system itself generating PCMCIA bus cycles or one of the N functions. When a function requires the bus it may generate a **Bus-Request**. The arbiter may respond in kind using a **Bus-Acknowledge** signal. Before signaling a **Bus-Acknowledge**, the arbiter must insure all other bus masters are removed from the bus by de-asserting their **Bus-Acknowledge**. In kind, these other devices TRI-STATE® their bus output drivers. In theory this all works well. In practice, however, there are a few potential issues.

First, the PCMCIA bus is not capable of recognizing it may not drive the PC Cards internal busses. The host has no **Bus-Request** or **Bus-Acknowledge** protocol mechanism. For example, while the arbiter on the card has provided bus control to a LAN device for moving a packet to a memory device using the on-card data and address bus, a PCMCIA bus cycle may asynchronously occur. From the system's perspective, it has exclusive right to gain access to the card's internal bus structures to complete the transaction. So how does the card inform the system the internal bussing is busy? The card could assert the **+RDY/-BSY** signal, but this signal is not available when the card is configured for I/O operation. Of course the *RRdy/-Bsy* bit in the Pin Replacement Register could be used, however, there is no protocol in place to guarantee that system software will read this bit before performing a transaction. If there was, a performance penalty would be paid in doing so. Of course, an interrupt could be generated. This, however, would require an interrupt handler and again, a performance penalty. Besides, interrupt processing is already complicated to support multifunction operation. A potential solution to this issue that is used by the PCM16C00VNG is to assert the **-WAIT** signal. The arbiter asserts **-WAIT** which will delay the PCMCIA host bus cycle and allow the bus mastering function on the card to continue operating or relinquish the bus immediately. Once complete, the arbiter would essentially grant the card bussing to the host by de-asserting **-WAIT**. If no device was using the card's bussing, the host could be allowed to perform its bus cycle immediately. If even more performance is desired, the multi-function interface silicon could allow the host access to internal IC registers while the function continued using the card bussing. Allowing this requires either a guarantee that these registers can not be accessed by a bus mastering function or that a protocol is put in place to access these registers. This could be done using memory based semaphores. In either case, the interface silicon is acting as a bridge between the PCMCIA bus and the PC Card's internal bussing.

Second, how are latency and throughput concerns satisfied for a broad range of applications? This implies that the arbiter policy may need to change given a certain arbiter mechanism implemented in silicon. For example, the description given for delaying a PCMCIA bus cycle must insure that the **-WAIT** signal is never asserted for more than 12 μs according to the specification. To do this and meet latency requirements for various functions, a programmable arbiter policy is best. This can be done with a simple register in which each possible bus mastering function, including the host, is given a priority value. When two devices are requesting bus access, the one with the highest priority is granted the bus. This master may then perform bus transactions until complete or a higher request is received. Once done, the process is repeated. An interesting variation on this scheme is to allow equal priorities. If all functions are programmed with an equal priority in the arbiter and a latency timer is used, then a fair policy may be implemented that insures consistent time slots for only those functions that request the bus. To do this, a latency timer is implemented that begins counting down when one function is currently granted the bus and another requests the bus. The function waiting for the bus is granted the bus when either the function currently using it completes or the timer expires, whichever comes first. In this manner, a function may own the bus indefinitely if no one else is requesting it. Other functions may also be assured a guaranteed latency from the time they request the bus until they are granted it. This ability to program a priority for each bus mastering function and a latency value offers the most flexibility in implementing PC Cards with various functions and allows latency and/or throughput to be controlled dependent on the functions used.

Third, not all devices are preemptable. From the above discussion, the arbiter has the ability to work with advanced bus mastering functions that can be preempted by the bus arbiter in its effort to implement a given programmed policy. Using the latency value which is designed to guarantee access time, a given bus master can also be assured this time on the bus even though another function is requesting the bus. Even with all this, some functions may not be preempted. If this is the case, the arbiter may be enhanced to include a non-preempt mode in addition to a preempt mode. In the preempt mode, the arbiter may fully implement policy decisions immediately. In non-preempt mode, the arbiter may not preempt a bus mastering function even if a new request has a higher priority than a current bus owner's. The current bus owner is allowed to complete and then a priority decision is made among the set of functions currently requesting the bus.

3.2 Accessing System Memory Resources

If implementing functions on PCMCIA Release 2.10 cards that desire to use bus-master or DMA across the PC Card/System boundary, other techniques must be pursued. A classic example of this problem and solution is the Programmed I/O Mode in 10 Mbit/s Ethernet® LANs. Here, the LAN IC itself or in conjunction with the multi-function interface silicon contains an I/O port. This port is mapped into

the PC Card's I/O Address space and subsequently the system's I/O address space (80x86 based). Typically these ports are 2 Bytes. The LAN function may use DMA or bus mastering techniques on the PC Card to place packet data in the register. The LAN device driver running on the system CPU may then read this data out. In doing so, the LAN function may then place more data in the port. This is repeated until all data is moved through the port. In this way, the function is using DMA to send data through the port to the system's memory. As you've surely noticed, this is not true DMA to system memory and it requires software overhead. It does, however, use less system memory resources for allocation to the LAN. If a traditional memory mapped approach is desired (Shared Memory mode for LAN), a portion of the common memory space on the PC Card may be mapped directly into the system memory address space. This requires system memory resources and still requires system software to move the data across the card/system boundary. It is typically, however, faster.

4.0 POWER MANAGEMENT

The current PCMCIA Standard does not fully address power management. While there is a power management working group within PCMCIA, they have not addressed power management in multi-function cards. A multi-function PC Card with more than one function must address power management. If a function is not in use, it should be placed in the lowest power state possible. Moreover, power should be minimized in all cases including when a function is in use. This management of power for one function should not disturb the ability of other functions to operate. Table II lists the power management methods used in devices today.

TABLE II. Typical Power Management Approaches

Function's power management in hardware.
Function's power management is signaled using hardware.
Function's power management is signaled using software.
Function' has its own power management capabilities.

If a function is capable of its own power management in hardware, it may be used on a multi-function card without disturbing other functions. Even if its associated software components control its power management by writing to a register within it, the device is easily used.

If a function has power management features that must be signaled by hardware, the multi-function interface silicon must address the issue. To do this, the PCM16C00VNG provides a **power-control** signal to each function. See *Figure 5.* These lines may be set high or low by writing to a power-management register. In doing this, software may exercise the functionality provided by a power control input to each function. For example, this could be **power-on/off** or a **power-on/sleep**. No assumptions need to be made regarding polarity.

While this is acceptable in some cases, many software packages will be unaware of a PCM16C00VNG device or would prefer not to use a software approach that would only work in some instances. Software techniques work well for macro-time power management such as when a business user is going on a 2 week trip and will not require the use of a LAN function. For micro-time power management, such as putting a MODEM into sleep mode, software techniques do not work due to the high frequency of power control commands such as power-on, sleep, and then power-on again. This requires power management that's transparent to system software. A solution to this problem is to add hardware power management capability to the multi-function interface silicon such as done with the PCM16C00VNG. In this case, the PCM16C00VNG may be configured to use a hardware power control scheme. When in this mode each function has a keep-alive timer that's programmable. Each time a keep-alive event occurs for a function (defined as either a host transaction to the function, a bus request from the function, or a ring-indicate), the counter is preset. If the keep-alive timer expires for a function, the PCM16C00VNG will de-assert the **power-control** pin to the function. This allows for dynamic hardware power control without software intervention.

If a function has absolutely no power management capabilities, there are still options that can be taken using the multi-function interface silicon. The most common one is to stop the clock on a CMOS device. The PCM16C00VNG allows this to be done by writing to the power-management register in the PCM16C00VNG. In addition, the PCM16C00VNG allows the functions clock to be divided by 32 for reduced power operation. To do all this, the PCM16C00VNG implements a clock gating and buffering scheme. These techniques provide macro-time power control with functions that have not been designed with reduced power states. See *Figure 5.*

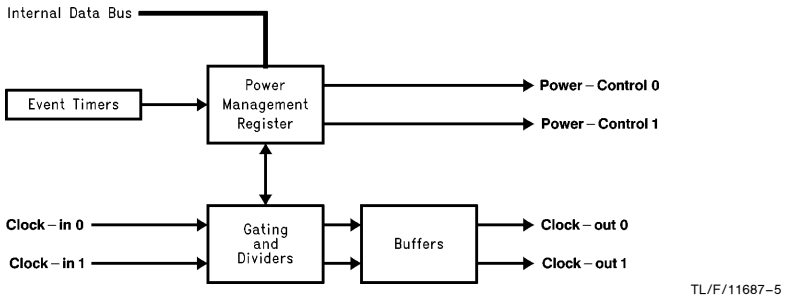


FIGURE 5. PCM16C00VNG Power Management Block Diagram

5.0 COMMON MEMORY ACCESS

In addition to the multiple I/O functions allowed by the multi-function card proposal, the common memory space may be used by memory devices. These devices may be easily implemented by passing all common memory accesses through or sideband to the interface silicon. In applications where common memory access uses the card side bussing in a multi-master environment, interface silicon must be present to delay a host common memory transaction. This is done as described previously using an arbiter within the interface silicon.

In the PCM16C00VNG implementation, additional decoding circuitry was not added to the silicon to decode common memory addresses. This saves additional hardware in the silicon. Typically, one or two devices require mapping into common memory. For this, one address line may be used to fragment the address space into two. Since most ICs have an active-low and active-high **chip select**, no silicon is needed for this fragmentation. If additional fragmentation is required, a simple logic IC may be added depending on the fragmentation requirements and the common memory devices themselves.

6.0 CONCLUSIONS

The Multi-Function Extension addresses the clear need of adding more functionality to a PC Card. This standard addresses complex system issues that have resulted from PCMCIA's incomplete implementation with respect to ISA. This standard has been implemented in a multiple-function interface chip.

The Revision 2.10 PCMCIA Standard and the MultiFunction Extension do not address bringing legacy device hardware that may operate as a bus-master of DMA device. While not optimal, there are solutions to using these devices in the current environment. Moreover, solutions exist for creating a bus-mastering environment on the PC Card itself. Such solutions may be employed until either the DMA proposal or Cardbus become predominant in the marketplace.

There is currently no standardized mechanism for managing power at the individual device level on a PC Card. Both hardware and software solutions exist, however, and may be employed to work with a wide range of power-capable devices.

With the Multi-Function Extension, to the PCMCIA PC Card Specifications solutions to using bus-mastering devices, and solutions to on-card device level power management, a multi-function interface device capable of working with a broad range of functions is possible.

National Semiconductor Corporation
2900 Semiconductor Drive
P.O. Box 58090
Santa Clara, CA 95052-8090
Tel: 1(800) 272-9959
TWX: (910) 339-9240

National Semiconductor GmbH
Livry-Gargan-Str. 10
D-82256 Fürstenfeldbruck
Germany
Tel: (81-41) 35-0
Telex: 527649
Fax: (81-41) 35-1

National Semiconductor Japan Ltd.
Sumitomo Chemical
Engineering Center
Bldg. 7F
1-7-1, Nakase, Mihama-Ku
Chiba-City,
Ciba Prefecture 261
Tel: (043) 299-2300
Fax: (043) 299-2500

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductores Do Brazil Ltda.
Rue Deputado Lacorda Franco
120-3A
Sao Paulo-SP
Brazil 05418-000
Tel: (55-11) 212-5066
Telex: 391-1131931 NSBR BR
Fax: (55-11) 212-1181

National Semiconductor (Australia) Pty, Ltd.
Building 16
Business Park Drive
Monash Business Park
Nottinghill, Melbourne
Victoria 3168 Australia
Tel: (3) 558-9999
Fax: (3) 558-9998