PCI 9080/V830 AN

NEC V830 to PCI bus Application

October 1, 1997

Version 1.0

Note

Features

- Complete Application Note for designing a PCI adapter or embedded system based on the NEC V830 processor including:
  - Detailed Design Description
  - OrCad Schematics
  - Verilog HDL Source Code
- Superior PCI performance based on PCI 9080 bus master interface chip which supports:
  - PCI burst master, DMA and slave cycles
  - PCI configuration cycles
  - Asynchronous PCI/ V830 operation
  - I_2O™ Messaging Unit
- Combined with PLX's I2OSDK®, provides a powerful tool for developing an V830-based I_2O IOP

General Description

This application note describes how to interface the NEC V830 CPU to the PCI bus using the PLX PCI 9080 "PCI to Local Bus Bridge" IC. The information can be used to build either a PCI adapter or embedded system.

The PCI 9080 has Direct Master, DMA and Direct Slave data transfer capabilities. The Direct Master mode allows a device (V830) on the Local Bus to perform memory, I/O, and configuration cycles to the PCI bus. The Direct Slave gives a master device on the PCI bus the ability to access memory on the Local Bus. The PCI 9080 allows the Local Bus to run asynchronously to the PCI bus through the use of bi-directional FIFOs.

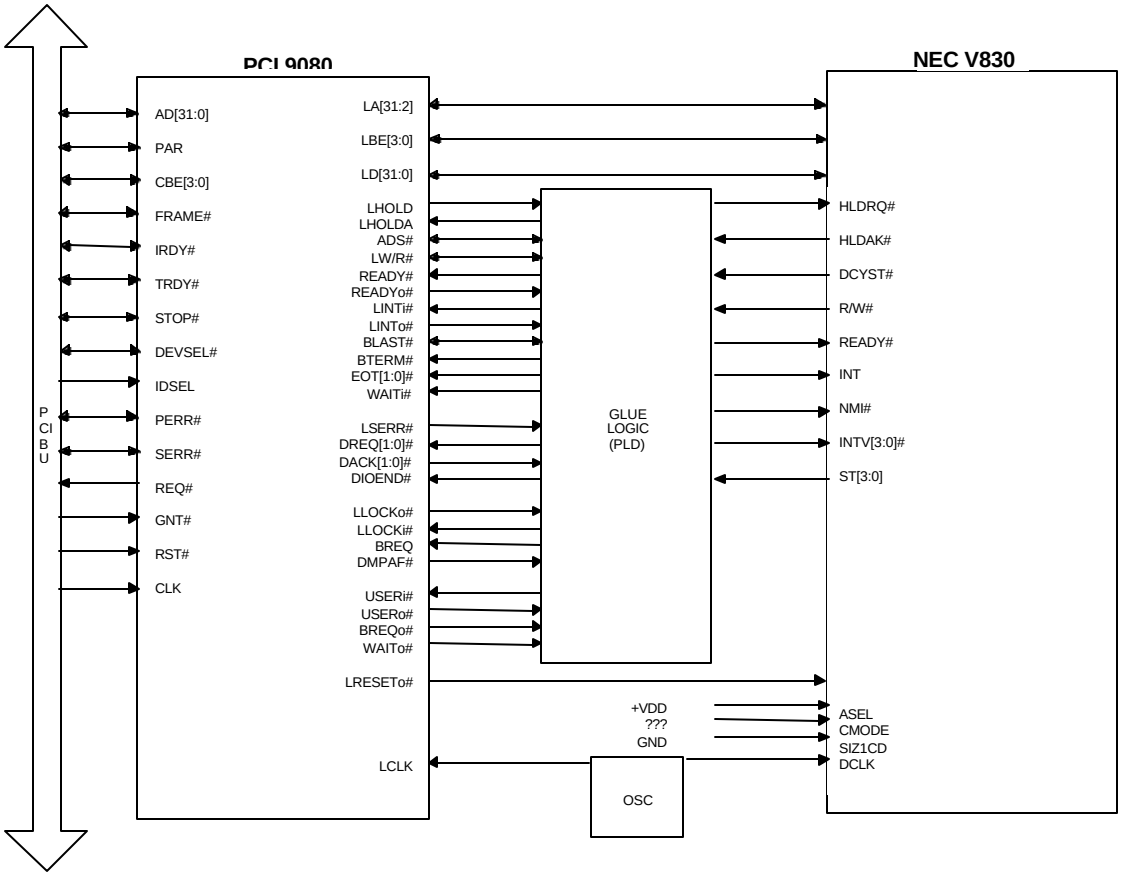


Figure 1. PCI to V830 Interface. V830 subsystem

Table of Contents:

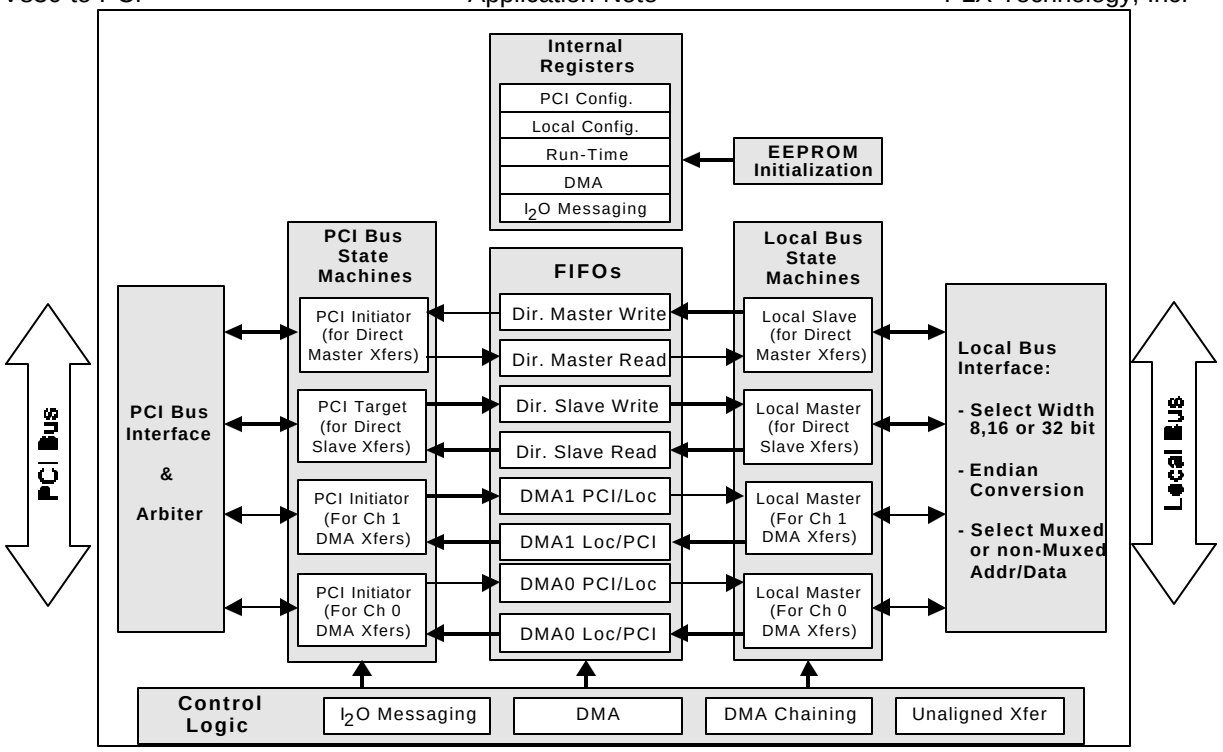


Figure 2. PCI 9080 Internal Block Diagram

PCI 9080 Overview

Features

- PCI Version 2.1 compliant Bus Master Interface chip for adapters and embedded systems
- I_2O Compatible Messaging Unit
- 3.3 or 5 Volt PCI signaling, 5 volt core, low-power CMOS in 208-pin PQFP
- Two independent DMA channels for local bus memory to/from PCI host bus data transfers
- Eight programmable FIFOs for zero wait state burst operation
- PCI ⇔ Local data transfers up to 132 MB/sec
- Programmable local bus supports non-multiplexed 32-bit address/data, multiplexed 32 or 16 bit, and slave accesses of 32, 16, or 8 bit local bus devices
- Local bus runs asynchronously to the PCI bus
- Eight 32 bit mailbox and two 32 bit doorbell registers
- Performs Big Endian/Little Endian conversion
- Upward compatibility with PCI 9060/9060ES/9060SD (See compatibility notes)

General Description

The PCI 9080 provides a compact, high performance PCI bus master interface for adapter boards and embedded systems. The programmable local bus of the chip can be configured to directly connect a wide variety of processors, controllers and memory subsystems.

The PCI 9080 contains an Intelligent I/O (I_2O) messaging unit that allows high performance and compatible software implementations of the I_2O bus protocol specification. Users of the PCI 9060, 9060ES and 9060SD chips may upgrade their products to support I_2O, 3.3 Volts and other new features with little or no change to existing hardware and software.

The PCI 9080 provides two independent chaining DMA channels with bi-directional FIFOs supporting zero wait state burst transfers between host and local memory. Slave transfers are performed through a third FIFO. A fourth FIFO allows the local processor and other controllers to perform direct bus master transfers to the PCI bus. The PCI 9080 also allows a local processor to configure other PCI devices in the system

Applications for the PCI 9080

PCI Adapter Cards

Major PCI adapter card applications for the PCI 9080 include high performance communications, networking, disk control, multimedia and video adapters. The PCI 9080 moves data between the host PCI bus and the adapter local bus in several ways. First, the local CPU or host processor may program the DMA controller of the PCI 9080 to move data between the adapter memory and the host PCI bus. Second, the 9080 can perform "Direct Master Transfers," whereby a local CPU or controller accesses the PCI bus directly through a PCI master transfer. The 9080 also supports slave transfers in which another PCI device is the master. Finally, the 9080 contains a complete messaging unit with mailbox registers, doorbell registers and queue management pointers that can be used for message passing under the I_2O protocol or a custom protocol.

Embedded Systems

Another application for the 9080 is in embedded systems, such as network hubs and routers, printer engines and industrial equipment. In this configuration, all four of the above-mentioned data transfer modes are used. In addition, the PCI 9080 supports Type 0 and Type 1 PCI configuration cycles, which allows the embedded CPU to act as the embedded system host and to configure the other PCI devices in the system.

Major Features

PCI 2.1 Compliant. The PCI 9080 is compliant with all aspects of PCI specification version 2.1.

I_2O Messaging Unit. The PCI 9080 incorporates an I_2O messaging unit. This enables the adapter or embedded system to communicate with other I_2O-supported devices. The I_2O messaging unit is fully compatible with the PCI extension of the I_2O Version 1.5 specification.

Dual Independent Programmable DMA Controllers with Bi-directional FIFOs. The PCI 9080 provides two independently programmable DMA controllers with bi-directional FIFOs for each channel. Each channel supports both non-chaining and chaining DMA modes and demand mode DMA.

Direct Bus Master. The PCI 9080 supports both memory mapped burst transfer accesses and I/O mapped single transfer accesses to the local bus from the PCI bus. The PCI 9080 also supports PCI bus interlock ("LOCK#") cycles. Bi-directional FIFOs for both Read- and Write-enable high-performance bursting on the local and PCI buses.

Direct Slave. The PCI 9080 supports both memory mapped and I/O mapped burst accesses to the local bus from the PCI bus. Bi-directional FIFOs for both Read- and Write-enable high-performance bursting on the local and PCI buses.

PCI Host Capability. In direct master mode, the PCI 9080 can generate Type 0 or Type 1 PCI configuration cycles.

Programmable Local Bus Modes. The PCI 9080 is a PCI bus master interface chip that connects a PCI bus to one of three local bus types, selected through mode pins. The PCI 9080 may be connected to any local bus with a similar design with little or no glue logic. Table 0-1 lists the three modes:

Table 0-1. Programmable Local Bus Modes

Mode	Description
C	32-bit address/32-bit data, non-multiplexed
J	32-bit address/32-bit data, multiplexed
S	32-bit address/16-bit data, multiplexed

Interrupt Generator. The PCI 9080 can generate PCI and local interrupts from several sources.

Clock. The PCI 9080 local bus interface runs from a local TTL clock and generates the necessary internal clocks. This clock runs asynchronously to the PCI clock.

3.3 Volt and 5 Volt Operation. The 9080 provides either 3.3 Volt or 5 Volt signaling on the PCI bus. A 3.3 V signaling environment requires 5 V and 3.3 V VCC. A 5 V PCI bus and local bus environment requires 5 V VCC.

Serial EEPROM Interface. The PCI 9080 contains an optional serial EEPROM interface that can be used to load configuration information. This is useful for loading information that is unique to a particular adapter (such as Network ID or Vendor ID).

Mailbox Registers. The PCI 9080 contains eight 32 bit mailbox registers that may be accessed from either the PCI or the local bus.

Doorbell Registers. The PCI 9080 includes two 32 bit doorbell registers. One generates interrupts from the PCI bus to the local bus. The other generates interrupts from the local bus to the PCI bus.

Unaligned DMA Transfer Support. The PCI 9080 can transfer data on any byte boundary.

Big/Little Endian Conversion. The PCI 9080 supports dynamic switching between Big Endian and Little Endian operations.

NEC V830 Overview

The V830 microprocessor is the first V830 family product to be offered by NEC for data processing applications.

The V830 is a high-performance 32-bit RISC microprocessor. With an operation(internal) frequency of 100MHz, the V830 can perform the data processing demanded by multimedia devices in only a few cycles. Besides a high interrupt responsibility and an optimized pipeline structure, a sum-of-products instruction, double-word shift instruction, and high-speed branch instruction using branch prediction have been added to support multimedia functions.

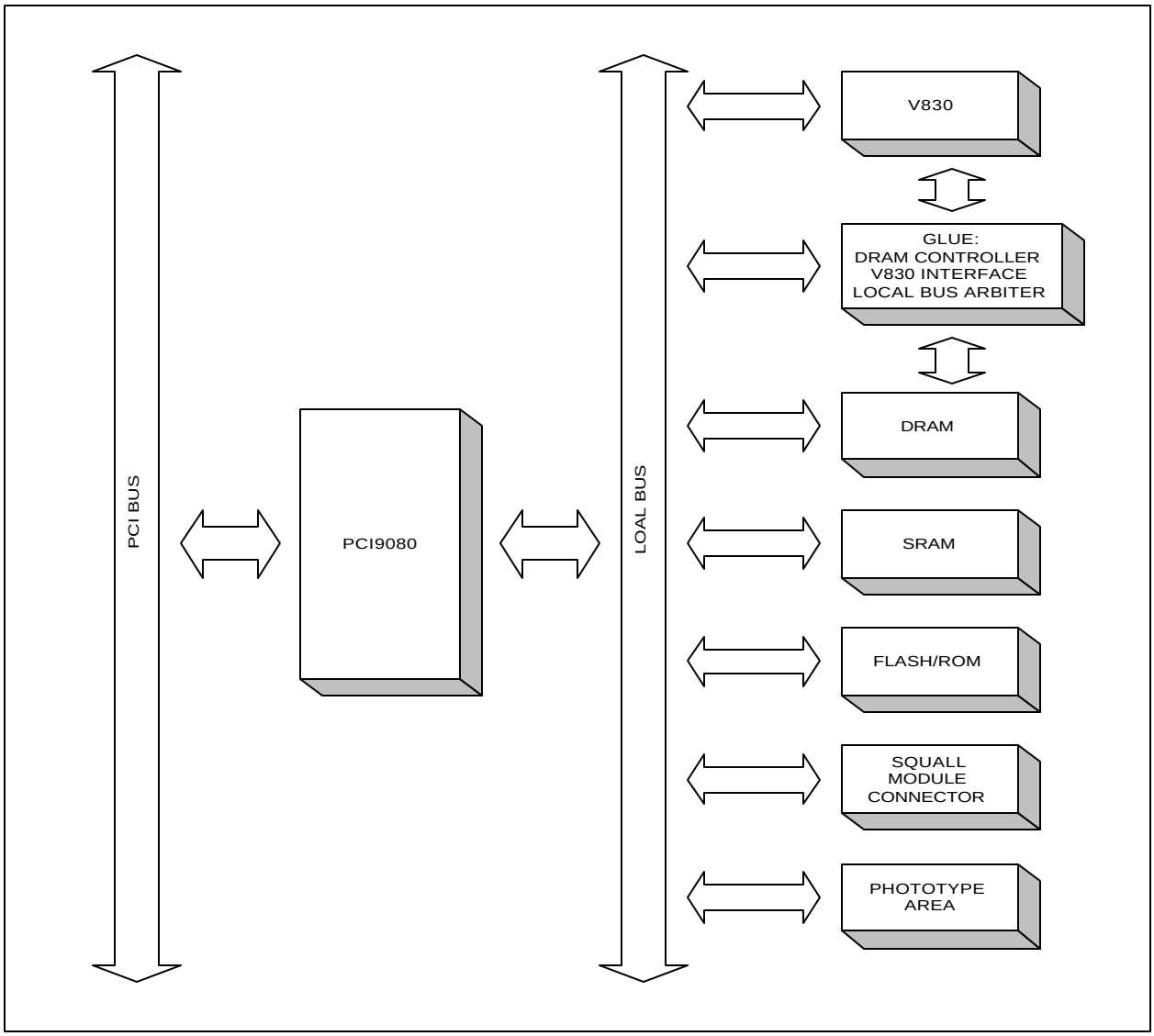
Furthermore, by inheriting V810 family basic instruction set at the object level, V810 Family software can be used as is.
The V830 offers high-performance data processing for applications such as image processing, game machines, car navigation, high-performance TVs, color facsimile machines, etc.

NEC V830 Features

- **Number of instructions: 102**
- **Minimum number of instruction execution cycles: 1**
- **General-purpose registers: 32 bits X 32**
- **Instruction set:**
  - V810 basic instruction set
  - Sum-of-products operation (32bits X 32bits + (upper/lower) 32bits): 1-3 cycles
  - Saturatable arithmetic operation (with a saturation detection function)
  - Double-word shift (64-bit data shift): 1-2 cycles
  - High-speed branch
  - Block transfer instruction
- **Memory space: Memory space, I/O space: 4G-byte Linear addressing**
- **External bus interface:**
  - Supports the (16byte) burst bus cycles.
  - Address/Data separate bus
  - Bus hold
  - Bus lock
  - Four Chip Select outputs
- **Internal memory:**
  - Instruction cache (direct mapping): 4K bytes
  - Data cache (direct mapping/write-through): 4K bytes
  - Instruction RAM: 4K bytes
  - Data RAM: 4K bytes
- **Interrupts:** One Non-maskable interrupt (NMI), and 16 levels of maskable interrupt.
- **Power control:** stop mode and sleep mode.
- **Package:** CMOS 144-pin plastic LQFP (fine pitch) (20 X 20 mm).

Introduction

This application note describes how to interface the V830, an NEC 32-bit microprocessor, to the PCI bus using the PLX PCI 9080 Bridge chip for I2O-ready applications. This application note demonstrates the connection between the PLX PCI 9080 bridge chip, NEC V830, SRAM, UART, FLASH or EEPROM, and DRAM via glue logic. In addition to the PCI 9080's direct master and direct slave transfer capabilities, this application will use the DMA controllers in the PCI 9080 Bridge chip to transfer data in both directions. The Direct Slave mode will be used to access all the local memory spaces (32bit address space or 4Gbyes.) The Direct Master mode may be used by the local master(s) (V830 in this case) to access all 32bit local memory spaces including PCI 9080 internal registers and the PCI memory spaces (All 32-bit PCI memory spaces excluding PCI memory spaces assigned to the current PCI 9080 Direct slave memory spaces for Space 0, Space 1, Expansion ROM, and Run Time internal memory spaces of 256bytes.) The PCI9080 allows the local bus to operate asynchronously to the PCI bus through the use of bi-directional FIFOs. In this application the PCI bus operates at 33 MHz while the local bus is also clocked at 33MHz (Asynchronous to PCI bus clock). The V830 internal operating frequency can be doubled or tripled of the local bus clock frequency. The following block diagram shows the basic connections between the major components required for this application.



V830 interface to PCI 9080 local bus

The V830 can access all devices on the local bus via glue logic. The V830 signals are translated to the PCI 9080 local bus signals: BCYST# => ADS#, READYo9080# => READY830, ST[3:0] => LLOCK9080, BLAST#, etc. Please refer to the glue logic source code for all other translated signals. Software can access the local bus peripherals by simply addressing correctly. Please refer to the local bus address map for each local device.
The V830 can access the PCI bus memory and I/O spaces via the PCI 9080 Direct Master interface and glue logic. The PCI 9080 provides a memory space (window), and an I/O space which can be used by the V830 to access any PCI I/O device or port. This I/O space is also used for generating PCI Configuration Cycles. Please refer to the PCI 9080 data book for more information.

Bus Arbitration

The PCI bus arbitration are done by the PCI bus arbiter. All local bus arbitration is done by the PCI 9080RDK-V830 local arbiter. In the special cases, where the local bus has no other device then the PCI 9080 and the V830, LHOLD and LHOLDA may be tied to HLDRQ# and HLDAK# via inverters. In this application note, the local bus is arbitrated between the PCI 9080, the V830, and the Squall master (SQLBR#, SQLBG#) by the glue logic. The exclusive PCI bus cycle via LOCK# signal is also supported when the V830 generates a Bus Lock Read/Write cycles. When the status information, ST[3:0], is '1011b', the glue logic asserts LLOCK# during ADS# cycle to indicate this is a locked direct master cycle. Note that locked cycle must start with a read and followed by a write cycle. The PCI 9080 can support many writes and reads after the first read cycle with the LLOCK#. In turn, the PCI 9080 will assert the PCI LOCK# signal for a locked PCI transaction. Please refer to the PCI Specification 2.X and the PCI 9080 data book for further information.

Clocks

The PCI9080 allows the local bus to operate asynchronously to the PCI bus through the use of bi-directional FIFOs. In this application the PCI bus operates at 25 MHz, 30 MHz, or 33 MHz while the local bus is clocked at 33MHz. The V830 internal clock may run at 2x (66MHz) or 3x (100MHz) of the local bus clock rate via jumper settings. Please refer to the schematic for proper settings.

The PCI bus clock (PCLK): PCI Bus clock from host system.

The Local bus clock (LCLK): same as BCLK for the V830

The V830 clock (BCLK): Clock output from U301 oscillator.

V830 internal clock: No jumper on the JP300 ---------- BCLK X 3 = 100MHz

Jumper installed on the JP300 --- BCLK X 2 = 66Mhz

Note: CMODE pin has pull-down resistor.

Interrupts

The Local Interrupt Input (LINTi#) is used to generate the PCI interrupt (INTA#) when an abnormal situation arises from the local glue logic or from the V830. The V830 can also generate the PCI interrupt via PCI 9080 internal doorbell registers. The doorbell register interrupt must be cleared from the PCI side. Please refer to the PCI 9080 data book for further information.

The local interrupt output (LINTo#) can be used by the V830 internal interrupt controller. The LINTo# may be active because one or more tasks are done, such as DMA done interrupt, or one or more error have occurred during the operation. Please refer to the PCI 9080 data book for further information on interrupts.

The V830 has built in interrupt controller with NMI#, INT, and four interrupt vector input pins (INTV[3:0]#). Because of pin constraint, the interrupt logic has been implemented in discrete logic. LINTo9080# from the PCI 9080, UART interrupt (INTUART#), and Two interrupts from the Squall module are ORed to generate the V830 interrupt input (INT830). The PCI 9080 interrupt input (LINTI9080#) is also generated with above logic (Note that LINTI9080# excludes the LINTO9080). Please refer to the schematic for more information.

When INTUART# is active, for example, INTUART# activates the INT830 and INTV1# to indicate interrupt vector is 1101b. LINTO9080#, LINTSQL0#, and LINTSQL1# are interrupt vector of 1110b, 1011b, and 0111b respectively. If interrupts are served from the V830, the local interrupt input (LINTI9080#) should be disabled via the PCI 9080 internal register. If the interrupts are routed to the PCI bus, the host software needs to find out which device generated the interrupt by reading the respective interrupt sources when the PCI interrupt (INTA#) is active. Note that all interrupt sources must to be level triggered.

In addition to the interrupts for better performance, this application note supports the system error and Dead-Lock backoff mechanism. When a system error is detected by the PCI 9080, the PCI 9080 will generate System Error (SERR#) and the Local System Error (LSERR#). Please refer to the PCI Spec. 2.X and PCI 9080 data book for conditions of this error generation. These are system errors and should be routed to NMI of the host processor. In this application, LSERR# is routed to the NMI# of V830 (NMI830#).

Another type of system error is the Dead-Lock. Please refer to the PCI 9080 data book for further information on Partial and Full Dead-Lock. When the PCI 9080 detects any possible Dead-Lock situation, mainly Direct Slave and Direct Master cycle occurring at the same time and programmed amount of time expires, PCI 9080 outputs BREQO9080. This signal also generates NMI830# to request a back-off from the V830. V830 MUST release the bus and wait until BREQO9080 is deasserted. Without the V830 back-off, the local bus will hang and the PCI9080 will issue continuous RETRYs to the master device responsible for the Direct Slave cycle. The V830 can check if LSERR# or BREQO9080 by reading the PCI 9080 internal status information. Reading of PCI 9080 internal register is allowed after the V830 gives up the Direct Master access (V830 accessing the PCI bus.) These signals can also be disabled.

Direct Master Mode

A Direct Master read/write cycle can be generated by the V830 or a Squall Module Master device. In other words, V830 or a Squall Module Master starts a cycle. The address must match the Range and the Direct master base address programmed in the PCI 9080 internal register in order to be recognized as a Direct Master Access by the PCI 9080. A Direct Master memory cycle should be between 8000 0000h and 8FFF FFFFh and a I/O cycle should be between A000 0000h and AFFF FFFFh. In turn, the PCI 9080 will generate appropriate PCI cycles: memory, I/O, or configuration cycle. The PCI starting address, that is when the Direct Master memory cycle address is 8000 0000h, can be programmed in the PCI Re-Map address register in the PCI 9080. For example, if the local Direct Master address is 8008 0000h, then the PCI address will be 0008 0000h if the Re-Map Register for Direct Master to PCI contains 0000 0003h (3 indicates that memory and I/O cycles are enabled. Therefore, the basic Re-Map address is 0000 0000h). The Re-Map address can be changed on-the-fly. Note that the Range Register must be a power of 2, and the Re-Map and base address value must be a multiple of the Range(not Range Register value). Where the Range Register value is the inverse of the Range.

A PCI Device Configuration cycle is also supported. In an embedded system, the immediate PCI 9080 internal registers are programmed by the V830 by accessing the memory region defined, starting from 3000 0000h. Simple memory read/write can modify any internal register. If V830 needs to configure any other PCI device, V830 needs to generate a PCI Configuration cycle via PCI 9080. Please refer to the PCI 9080 data book for more detailed information. In a PC environment, most of the initializations are done by the system BIOS.

Direct Slave Mode

A Direct Slave read/write cycle can be performed by the host CPU to the local bus through the PCI 9080 and local interface logic. Space 0, Space 1, or the Expansion ROM space of the PCI 9080 can be used to access the local bus peripherals such as SRAM, DRAM, UART, Squall connector, etc. The user must assign the Range Register values for Space 0, Space 1, and Expansion ROM space via the serial EEPROM or via the local processor (V830) during power up. Note that the Range Register assignment must occur before the PCI BIOS scans the PCI bus. The Re-Map Register & the Bus Region Descriptor may be assigned at the same time or after power-up sequence. When the PCI BIOS scans the PCI devices, the BIOS will assign base address for each PCI address space found. The Base, Range, Re-Map, and Bus Region Descriptor Register must be programmed correctly before any local bus access can be attempted. Note: Additional register such as the Arbitration Register may need to be programmed. Please refer to the memory map and PCI 9080 data book for detailed internal register locations. Please refer to the initial register value in this document.

DMA Controller

The PCI 9080 has built-in two independent DMA controllers. These DMA controllers support data transfer of any size, up to 1Gbyte at a time, from the PCI bus to the local bus, or from the local bus to the PCI bus. The DMA controller supports normal DMA mode, chained DMA mode, and demand DMA mode. In a normal DMA mode, the user must program the PCI 9080 internal DMA registers (DMA mode, PCI starting address, Local starting address, byte count, the direction of transfer(PCI to Local, or Local to PCI)) and DMA enable/start bits to kick off the DMA transfer. In addition to the above registers, the user can program the DMA FIFO control registers, DMA done interrupts, etc. Please refer to the PCI 9080 data book for more information.

In a chained DMA mode, the user must generate one or more DMA chain information either on the local bus or the PCI bus memory space. The mode register must indicate that the current DMA transfer is a chained DMA via current DMA channel (channel 0 or 1) mode register. Finely, the user must write the location of the first descriptor (memory address and PCI side or Local side). Once the DMA start bit is written, the DMA controller reads in the first descriptor, which contains all the information about the current DMA transfer, the next descriptor location or end of chain. The DMA controller transfers data and when it is done, reads in the next descriptor if there is any.

A Demand Mode DMA utilizes DMA Request (DREQ#, input to the DMA controller) and DMA Acknowledge (DACK#, output from the DMA controller) handshake signals to control the flow of data. To use it, setup the DMA controller in either normal or chained DMA transfer with demand mode turned on in the DMA control register. The DMA controller will read in the first descriptor information if it is in chained DMA transfer mode but will not start the DMA data transfer until the DREQ# is asserted from the local bus. When the DREQ# is asserted, the DMA controller will arbitrate for the both side of the bus (Local and PCI). When the DMA controller gets the local bus, the DMA controller asserts DACK# to start current DMA data transfer. If the DREQ# is deasserted during the DMA data transfer, DMA controller will pause the DMA data transfer and deassert DACK# and release both side of the bus. The DMA data transfer can be resumed by asserting DREQ# again. Or DMA data transfer can be aborted by writing the DMA abort bit in the DMA control register. (Demand Mode DMA is not supported by current RDK board.)

In addition to three DMA transfer mode, the PCI 9080 supports the End Of Transfer(EOT[1:0]#) pins to terminate current DMA. This mode is not supported by the current RDK board.

GLUE LOGIC

For most flexibility and comprehensive interface between PCI 9080, V830, and peripherals an Altera EPF8452AQC160 is used as glue logic. This glue logic interfaces between PCI 9080 and V830, DRAM controller, SRAM control signals, Squall control signals, UART control signals, and FLASH control signals. The DRAM, SRAM, UART,FLASH can be accessed from the host via PCI 9080, V830, and Squall master module.

SRAM

512Kbytes of SRAM are on board. These SRAM can be read with 0 wait state (ws) and written with one wait state. The address counter is provided (74FCT163) for faster address valid time, or address provided by the
PCI 9080 or V830 address can be used instead.

DRAM

The DRAM controller resides inside the glue logic to provide the DRAM control signals as well as address lines and RAS and CAS signals. 4MB, 8BM, 16MB, or 32MB Fast Page Mode DRAM module can be used.

UART

UART, serial port, is used as remote monitoring and control. The V830 ROM monitor can be directly controlled via this serial port terminal connection.

SQUALL CONNECTOR

Any other master device(s), such as Ethernet networking card, can be connected to the Squall connector which can be accessible by the host processor or the V830. Please refer to the Squall Connector specifications. Check Cyclone web site (www.cyclone.com) or Intel web site (www.intel.com) for more information on the Squall connector.

Memory map

0000 0000 --- 0FFF FFFF : V830 internal RAM (not accessible from external)
1000 0000 --- 1FFF FFFF : DRAM region
2000 0000 --- 2FFF FFFF : SRAM region

3000 0000 --- 3FFF FFFF : PCI 9080 internal registers region

4000 0000 --- 4FFF FFFF :
5000 0000 --- 5FFF FFFF :
6000 0000 --- 6FFF FFFF :
7000 0000 --- 7FFF FFFF :

8000 0000 --- 8FFF FFFF : Direct Master Memory region

9000 0000 --- 9FFF FFFF :
A000 0000 --- AFFF FFFF : Direct Master I/O region
B000 0000 --- BFFF FFFF :

C000 0000 --- CFFF FFFF : Squall Module region
D000 0000 --- DFFF FFFF : UART region
E000 0000 --- EFFF FFFF :
FE00 0000 --- FE00 0FFF : V830 Build-in instruction RAM
FF00 0000 --- FFFF FFFF : FLASH region

PCI9080 Configuration Registers

The following configuration registers must be programmed (from serial EEPROM) before Direct Slave or Direct Master accesses. Note that DMA Start bit (and DREQ0# if demand mode DMA) must be set after DMA registers are programed in order for the DMA controller to start.

Extra Long EEPROM Load

EEPROM Offset	EEPROM Value	Description
0	9080	Device ID
2	10B5	Vendor ID
4	0680	Class Code
6	0002	Class Code, Revision
8	0000	Maximum Latency, Minimum Grant
A	0100	Interrupt Pin, Interrupt Line Routing
C	0123	MSW of Mailbox 0 (User Defined)
E	4567	LSW of Mailbox 0 (User Defined)
10	89AB	MSW of Mailbox 1 (User Defined)
12	CDEF	LSW of Mailbox 1 (User Defined)
14	FF00	MSW of Range for PCI to Local Address Space 0 (16 MB)
16	0000	LSW of Range for PCI to Local Address Space 0 (16 MB)
18	2000	MSW of Local Base Address (Remap) for PCI to Local Address Space 0 (POINT TO SRAM)
1A	0001	LSW of Local Base Address (Remap) for PCI to Local Address Space 0
1C	0000	MSW of Local Arbitration Register
1E	0000	LSW of Local Arbitration Register
20	0000	MSW of Local Bus Big/Little Endian Descriptor Register
22	0000	LSW of Local Bus Big/Little Endian Descriptor Register
24	0000	MSW of Range for PCI to Local Expansion ROM
26	0000	LSW of Range for PCI to Local Expansion ROM
28	FF00	MSW of Local Base Address (Remap) for PCI to Local Expansion ROM (POINT TO FLASH)
2A	0000	LSW of Local Base Address (Remap) for PCI to Local Expansion ROM
2C	4903	MSW of Bus Region Descriptors for PCI to Local Accesses
2E	00C3	LSW of Bus Region Descriptors for PCI to Local Accesses
30	F000	MSW of range for Direct Master to PCI (256MB)
32	0000	LSW of range for Direct Master to PCI
34	8000	MSW of Local Base Address for Direct Master to PCI Memory
36	0000	LSW of Local Base Address for Direct Master to PCI Memory
38	A000	MSW of Local Bus Address for Direct Master to PCI IO/CFG
3A	0000	LSW of Local Bus Address for Direct Master to PCI IO/CFG
3C	0000	MSW of PCI Base Address (Remap) for Direct Master to PCI
3E	0003	LSW of PCI Base Address (Remap) for Direct Master to PCI
40	0000	MSW of PCI Configuration Address Register for Direct Master to PCI IO/CFG
42	0000	LSW of PCI Configuration Address Register for Direct Master to PCI IO/CFG
44	0830	Subsystem ID
46	10B5	Subsystem Vendor ID
48	FF00	MSW of Range for PCI to Local Address Space 1 (16 MB)
4A	0000	LSW of Range for PCI to Local Address Space 1 (16 MB)
4C	1000	MSW of Local Base Address (Remap) for PCI to Local Address Space 1 (POINT TO DRAM)
4E	0001	LSW of Local Base Address (Remap) for PCI to Local Address Space 1
50	0000	MSW of Bus Region Descriptors (Space 1) for PCI to Local Accesses
52	00C3	LSW of Bus Region Descriptors (Space 1) for PCI to Local Accesses
54	0000	MSW of PCI Base Address for local expansion ROM
56	0000	LSW of PCI Base Address for local expansion ROM

VERILOG HDL SOURCE CODE FOR GLUE LOGIC

```
INCLUDE "arb";
INCLUDE "dram";
INCLUDE "led";
INCLUDE "memdec";
-- INCLUDE "reset";
INCLUDE "sram";
INCLUDE "v830-9080";


SUBDESIGN 9080V830
(


---*************************************************************************************************************************
--*
--*                       Input pins
--*
--*************************************************************************************************************************

--- V830
            /hldak830           : INPUT;  -- HoldAcknoledge from V830
            /bcyst              : INPUT;  -- write/read signal form v830
            /wr830              : INPUT;  -- ads  form v830
            /st[3..0]           : INPUT;  -- status bits 3 to 0
            int830              : INPUT;  -- interrupt input(this int goes to V830 too)
                                        --need to generate LINTi9080#, int input to 9080

---PCI 9080
            lclk                : INPUT;  -- local bus clk (common)
            la[31..2]           : INPUT;  -- address bus
            /lbe[3..0]          : INPUT;  -- local bus byte enables (common)
            /ads                : bi;     -- address strobe (common)
            /lwr                : bi;     -- write/read# (common)
            /blast              : bi;     -- burst last (common)
            /den                : bi;     -- data enable for transceivers (common)
            /wait               : bi;     -- wait for wait states (common)

            /readyo9080         : INPUT;  -- ready out from 9080
            breqo9080           : INPUT; --  bus request output to 9080
            /lreseto9080        : INPUT;  -- local bus reset out from 9080
            hold9080            : INPUT;  -- hold from 9080
            user9080            : INPUT;  -- user out from 9080
            /locko9080          : INPUT;  -- lock out from 9080

            /dack09080          : INPUT; -- DMA ACK for CH0 from PCI 9080
            /dack19080          : INPUT; -- DMA ACK for CH1 from PCI 9080
            /lserr9080                  : INPUT; -- Local system error from PCI 9080
            /ldshold9080        : INPUT; -- Direct Slave Hold request from 9080

---Squall
            /sqlbr              : INPUT;  -- bus request from Squall
            sqlsda              : INPUT; --  squall serial data line for eeprom
            sqlscl              : INPUT; --  squall serial eeprom clock
            /locksql            : INPUT; -- lock input from Squall
            /sqlextend          : INPUT; --  squall extend

---Misc
```

```
        /por2           : INPUT;  -- local bus reset from power-on reset circuit
        srcy            : INPUT;


---*****************************************************************************************************************
--*
--*                   output pins
--*
--*****************************************************************************************************************

---v830
        /hldrq830       : OUTPUT;  -- Hold request to 830
        /ready830               : OUTPUT;  -- Ready signal to 830
        /nmi830         : OUTPUT;  -- NMI input to 830(when LSERR or BREQo#)

---Arbitration
        breq9080        : OUTPUT; -- bus request to 9080
        hlda9080        : OUTPUT; -- hold ack. to 9080
        /lock9080       : OUTPUT; -- lock input to 9080
        /sqlbg          : OUTPUT; -- bus grant to Squall
        /readysql       : OUTPUT; -- sql ready signal
        /lreset         : OUTPUT; -- local reset signal
        9080debug       : OUTPUT; -- debug mode LED output
        failo           : OUTPUT; -- failed mode LED output

---CS
        /cs9080         : OUTPUT; -- 9080 chip select
        /cssql          : OUTPUT; -- Squall chip select
        /cssram         : OUTPUT; -- SRAM chip select
        /csflash        : OUTPUT; -- Flash chip select
        /csuart         : OUTPUT; -- Uart chip select

---PCI 9080
        /waiti9080      : OUTPUT; -- wait input to 9080
        /bterm9080      : OUTPUT; -- bus terminate to 9080
        /lreset         : OUTPUT; -- local bus reset (active low)
        9080debug       : OUTPUT; -- 9080 bedug output to LED
        /ready9080      : OUTPUT; -- local bus Ready to 9080
        /dreq09080      : OUTPUT; -- DMA REQ for CH0 to PCI 9080
        /dreq19080      : OUTPUT; -- DMA REQ for CH1 to PCI 9080

---Flash, UART
        /flashoe        : OUTPUT; -- Flash output enable
        /flashwe        : OUTPUT; -- Flash write enable
        /uartr          : OUTPUT; -- Uart Read
        uartw           : OUTPUT; -- Uart Write (Note: active high)
        /ioads          : OUTPUT;  -- ads for uart
        /iobuf          : OUTPUT; -- data tranceiver enable for Flash & Uart

---SRAM
        /sramoe         : OUTPUT;
        /srinc          : OUTPUT;
        /sramwe[3..0]   : OUTPUT;

---DRAM
        /dramwe         : OUTPUT; -- Dram write enable
        /dramras[3..0]  : OUTPUT; -- Dram Row Address Strobe
        /dramcas[3..0]  : OUTPUT; -- Dram Column Address Strobe
        drama[10..0]    : OUTPUT; -- Dram Multiplexed addresses


)


VARIABLE
        sm_clk          : NODE; -- State machine clock
```

```
        sm_rst          : NODE; -- State machine reset
        /readydram      : NODE; -- local bus bus ready
        /readyflash     : NODE; -- local bus bus ready
        /readysram      : NODE;
        ---boffok       : NODE;
        /readyuart      : NODE; -- local bus bus ready
        count[8..0]     : DFF;      --
BEGIN

-- ready signals

/ready9080 = (/readydram & /readyflash & /readysql & readysram & /readyuart);
/ready830=(/readydram & /readyflash & /readysql & /readysram & /readyuart & /readyo9080);


-- These signals are presently unused for this design.
--unused pins to 9080

--/bigend9080
--useri9080
--/eot0
--/eot1
--/dack09080
--/dack19080

/bterm9080 = '1';
/dreq09080 = '1';
/dreq19080 = '1';
/waiti9080 = '1';
breq9080 = '0';

/locksql = vcc;                 -- not used
/lreseti9080 = vcc; -- not used



-- Local Bus Arbitration
(/lock9080, /hldrq830, breq9080, hlda9080, hlda9080, /sqlbg) =
        arb (sm_clk, sm_rst, hold9080, /hldak830, /sqlbr, /locko9080)

-- DRAM Controller
(/dramwe, /dramras[3..0], /dramcas[3..0], drama[10..0], /readydram) =
        dram (sm_clk, sm_rst, la[31..2], /wait, /blast, /ads, /lwr, /lbe[3..0], /den);

-- LED Controller
(failo, 9080debug) = led ( user9080);

-- Memory Decoder
(/cs9080, /csflash, /csuart, /cssql, /flashoe, /flashwe, /iobuf, /uartr, uartw, /readyflash, /readyuart, /ioads) =
         memdec (la[31..24], /ads, /blast, sm_clk, sm_rst, /lwr, /wait, /den, /readysql);

-- SRAM Controller
(/sramwe[3:0], /sramoe, /srinc, /readysram, /cssram) =
        sram (sm_clk, sm_rst, /blast, /lwr, srcy, la[31..24], /ads, /lbe[3:0]);

--v830-9080
(/ready830, /nmi830, /ads, /blast, /lwr ) =
        v830-9080 (sm_clk, sm_rst, /st[3:0], /lserr9080, /wr830, readyo9080, /bcyst, breqo9080);

-- State machine Reset Logic
sm_clk = global(lclk);
sm_rst = dffe(vcc, sm_clk, !(!/por2 # !/lreseto9080), , count[2]);

-- Local Bus Reset Logic
/lreset = !(!/por2 # !sm_rst # /lreseto9080);

-- Reset Timer
```

```
count[].clk = lclk;
count[].clrn = !(!/por2 # !/lreseto9080);
count[] = count[] + 1;


END;
```

```
---*************************************************************************************************************************
--*
--*                    Arbiter (Local bus)
--*
--*************************************************************************************************************************

SUBDESIGN arb
(
        sm_clk          : INPUT; -- state machine clock
        sm_rst          : INPUT; -- state machine reset
        hold9080        : INPUT; -- bus request from 9080
        /hldak830       : INPUT; -- bus grant from the V830
        /sqlbr          : INPUT; -- squall bus request
        /locko9080      : INPUT; -- PCI locked output from 9080


        /lock9080       : OUTPUT; -- LOCK INPUT TO 9080
        /hldrq830       : OUTPUT; -- bus request to V830
        breq9080        : OUTPUT; --
        hlda9080        : OUTPUT; -- bus grant for the 9080
        /sqlbg          : OUTPUT; -- bus grant for squall



--following signals are not used yet.
--Squall Lock, Direct slave Lock, Demand DMA(/Dack[1:0]9080, Dreq[1:0]9080, EOT[1:0], etc), Sqlsda,Sqlscl,/Sqlextend
--Bterm9080, waiti9080, etc.

        int830          : INPUT;  -- interrupt input(this int goes to V830 too)
                                  -- need to generate LINTi9080#, int input to 9080
        /dack09080      : INPUT; -- DMA ACK for CH0 from PCI 9080
        /dack19080      : INPUT; -- DMA ACK for CH1 from PCI 9080
        sqlsda          : INPUT; --  squall serial data line for eeprom
        sqlscl          : INPUT; --  squall serial eeprom clock
        /sqlextend      : INPUT; --  squall extend
        /locksql        : INPUT; -- lock input from Squall
        /ldshold9080    : INPUT; -- Direct Slave Hold request from 9080

        /lock9080       : OUTPUT; -- lock input to 9080
        /waiti9080      : OUTPUT; -- wait input to 9080
        /bterm9080      : OUTPUT; -- bus terminate to 9080
        /dreq09080      : OUTPUT; -- DMA REQ for CH0 to PCI 9080
        /dreq19080      : OUTPUT; -- DMA REQ for CH1 to PCI 9080


)


VARIABLE

sm      : MACHINE
        OF BITS (q[3..0])
        WITH STATES
        (
                idle            = b"0001",      -- V830 is default master
                9080            = b"0010",
                sql             = b"0100",
                wait            = b"1000",
        );

BEGIN

DEFAULTS
        /hldrq830 = vcc;
        breq9080 = gnd;
        hlda9080 = gnd;
        /sqlbg = vcc;
```

```
        sm = idle
END DEFAULTS;

sm.clk = sm_clk;
sm.reset = !sm_rst;



-- Bus Arbiter.

CASE sm IS
        WHEN idle =>        -- v830 has control of the bus
                /hldrq830 = vcc;                -- don't request bus from v830
                IF (hold9080) THEN
                        /hldrq830 = gnd;  --request the bus from v830
                        sm = 9080;
                ELSEIF (!/sqlbr) THEN
                        /hldrq830 = gnd;
                        sm = sql;
                ELSE
                        sm = idle;
                END IF;

        WHEN 9080 =>        -- 9080 wants bus
                IF (!/hldak830) THEN      --V830 gives up the bus
                        hlda9080 = vcc;     --grant the bus to 9080
                        sm = wait;
                ELSEIF (/locko9080) THEN  --If PCI is locked with 9080,
                                                        -- Grant the bus to 9080.
                        sm = 9080;
                ELSEIF (!hold9080) THEN
                        sm = idle;
                ELSE
                        sm = 9080;
                END IF;

        WHEN sql  =>        -- squall master wants bus
                IF (!/hldak830) THEN   --V830 gives up the bus
                        /sqlbg = gnd;  --grant the bus to squall master
                        sm = wait;
                ELSEIF (/sqlbr) THEN
                        sm = idle;
                ELSE
                        sm = sql;
                END IF;

        WHEN wait =>        -- wait before release the bus by the 9080 or squall
                IF (!hold9080) THEN     --if 9080 gives up the bus
                        hlda9080= gnd;    --remove the holda9080
                        sm = idle;
                ELSEIF (/sqlbr) THEN    -- if squall master gives up the bus
                        /sqlbg= vcc;      --remove the bus grant from squall master
                        sm = idle;
                ELSE
                        sm = wait;
                END IF;


END CASE;

END;
```

```
---*************************************************************************************************************************
--*
--*                    DRAM (Address 0x10000000)
--*
--*************************************************************************************************************************


SUBDESIGN dram
(
        sm_clk          : INPUT; -- the state machine's clock
        sm_rst          : INPUT; -- the state machine's reset
        la[31..2]       : INPUT;  --
        /wait           : INPUT;  --  Wait from 9080 & V830
        /blast          : INPUT;  --
        /ads            : INPUT; --
        /lwr            : INPUT; --
        /lbe[3..0]      : INPUT; --
        /den            : INPUT;

        /dramwe         : OUTPUT; --
        /dramras[3..0]  : OUTPUT; --
        /dramcas[3..0]  : OUTPUT; --
        drama[10..0]    : OUTPUT; --
        /readydram      : OUTPUT; --
)


VARIABLE

        /ras            : NODE; --
        /cas[3..0]      : NODE; --
        /csdram         : NODE; --
        count[8..0]     : DFF;      -- refresh counter
        ref_req         : NODE; -- refresh request
        /mux            : NODE;
        ref_cyc         : NODE;
        cpu_cyc         : NODE;
        hold_off        : NODE;
        r4              : NODE;


rassm   : MACHINE
        OF BITS (a[1..0])
        WITH STATES (
                rasidle         = b"01",
                rasactive               = b"10");

muxsm   : MACHINE
        OF BITS (b[4..0])
        WITH STATES (
                muxidle         = b"00001",
                muxwait         = b"00010",
                muxwait1                = b"00100",
                muxwait2                = b"01000",
                muxactive       = b"10000");


cas0sm  : MACHINE
        OF BITS (c[1..0])
        WITH STATES (
                cas0idle        = b"01",
                cas0active      = b"10");

cas1sm  : MACHINE
        OF BITS (d[1..0])
        WITH STATES (
                cas1idle        = b"01",
```

```
                cas1active      = b"10");

cas2sm  : MACHINE
        OF BITS (e[1..0])
        WITH STATES (
                cas2idle        = b"01",
                cas2active      = b"10");

cas3sm  : MACHINE
        OF BITS (f[1..0])
        WITH STATES (
                cas3idle        = b"01",
                cas3active      = b"10");

drdysm  : MACHINE
        OF BITS (g[1..0])
        WITH STATES (
                drdyidle        = b"01",
                drdyactive      = b"10");

cpu_cycsm       : MACHINE
        OF BITS (h[1..0])
        WITH STATES (
                cpu_cycidle     = b"01",
                cpu_cycactive   = b"10");

ref_cycsm       : MACHINE
        OF BITS (i[1..0])
        WITH STATES (
                ref_cycidle     = b"01",
                ref_cycactive   = b"10");

ref_reqsm       : MACHINE
        OF BITS (j[1..0])
        WITH STATES (
                ref_reqidle     = b"01",
                ref_reqactive   = b"10");

hold_offsm      : MACHINE
        OF BITS (k[1..0])
        WITH STATES (
                hold_offidle    = b"01",
                hold_offactive  = b"10");

addrsm  : MACHINE
        OF BITS (l[3..0])
        WITH STATES (
                addridle        = b"0001",
                addrs1          = b"0010",
                addrs2          = b"0100",
                addrs3          = b"1000");



BEGIN

DEFAULTS
                /ras = gnd;                     -- RAS inactive
                /cas[3..0] = vcc;               -- CAS inactive
                /readydram = vcc;
                /mux = vcc;
                ref_cyc = gnd;
                cpu_cyc = gnd;
                ref_req = gnd;
                hold_off = gnd;

END DEFAULTS;
```

```
rassm.clk = sm_clk;
rassm.reset = !sm_rst;

muxsm.clk = sm_clk;
muxsm.reset = !sm_rst;

cas0sm.clk = sm_clk;
cas0sm.reset = !sm_rst;

cas1sm.clk = sm_clk;
cas1sm.reset = !sm_rst;

cas2sm.clk = sm_clk;
cas2sm.reset = !sm_rst;

cas3sm.clk = sm_clk;
cas3sm.reset = !sm_rst;

drdysm.clk = sm_clk;
drdysm.reset = !sm_rst;

cpu_cycsm.clk = sm_clk;
cpu_cycsm.reset = !sm_rst;

ref_cycsm.clk = sm_clk;
ref_cycsm.reset = !sm_rst;

ref_reqsm.clk = sm_clk;
ref_reqsm.reset = !sm_rst;

hold_offsm.clk = sm_clk;
hold_offsm.reset = !sm_rst;

addrsm.clk = sm_clk;
addrsm.reset = !sm_rst;

-- RAS is common to all. Bytes are selected using CAS
/dramras0 = /ras;
/dramras1 = /ras;
/dramras2 = /ras;
/dramras3 = /ras;


/dramcas[3..0] = /cas[3..0];

/dramwe = !(!/den & /lwr & !ref_cyc & !/ras);


count[].clk = sm_clk;
count[].clrn = !ref_req;
count[] = count[] + 1;

/csdram = !(!/ads & (la[31..28] == H"1") );

r4 = (count[7] == b"1") & (count[3] == b"1");

-- RAS Control State Machine
CASE rassm IS
        WHEN rasidle =>
                /ras = vcc;
                IF ( (!/csdram & /mux & !ref_req) # (cpu_cyc & !ref_cyc) # (!/cas0 & ref_cyc) ) THEN
                        rassm = rasactive;
                ELSE
                        rassm = rasidle;
                END IF;
        WHEN rasactive =>
```

```
                        /ras = gnd;
                        IF (!/blast & !/readydram)  THEN
                                rassm = rasidle;
                        ELSIF (/cas0 & ref_cyc) THEN
                                rassm = rasidle;
                        ELSE
                                rassm = rasactive;
                        END IF;
END CASE;


-- MUX Control State Machine
CASE muxsm IS
        WHEN muxidle =>
                --drama[10..2] = la[21..13];
                drama[9..2] = la[21..14];
                drama[10] = la[23];
                IF (!/ras) THEN
                        muxsm = muxwait;
                ELSE
                        muxsm = muxidle;
                END IF;
        WHEN muxwait =>
                --drama[10..2] = la[21..13];
                drama[9..2] = la[21..14];
                drama[10] = la[23];
                muxsm = muxwait1;
        WHEN muxwait1 =>
                --drama[10..2] = la[21..13];
                drama[9..2] = la[21..14];
                drama[10] = la[23];
                muxsm = muxwait2;
        WHEN muxwait2 =>
                --drama[10..2] = la[21..13];
                drama[9..2] = la[21..14];
                drama[10] = la[23];
                muxsm = muxactive;
        WHEN muxactive =>
                /mux = gnd;
                drama[9..2] = la[11..4];
                drama[10] = la[22];
                IF (/ras) THEN
                        muxsm = muxidle;
                ELSE
                        muxsm = muxactive;
                END IF;
END CASE;


-- CAS0 Control State Machine
CASE cas0sm IS
        WHEN cas0idle =>
                IF ( (!/ras & !/mux & !/lbe0 & !ref_cyc) # (!cpu_cyc & ref_req) ) THEN
                        cas0sm = cas0active;
                ELSE
                        cas0sm = cas0idle;
                END IF;
        WHEN cas0active =>
                        /cas0 = gnd;
                        IF ( (!ref_cyc) # (!/ras & ref_cyc) ) THEN
                                cas0sm = cas0idle;
                        ELSE
                                cas0sm = cas0active;
                        END IF;
END CASE;

-- CAS0 Control State Machine
```

```
CASE cas1sm IS
        WHEN cas1idle =>
                IF ( (!/ras & !/mux & !/lbe1 & !ref_cyc) # (!cpu_cyc & ref_req) ) THEN
                        cas1sm = cas1active;
                ELSE
                        cas1sm = cas1idle;
                END IF;
        WHEN cas1active =>
                        /cas1 = gnd;
                        IF ( (!ref_cyc) # (!/ras & ref_cyc) ) THEN
                                cas1sm = cas1idle;
                        ELSE
                                cas1sm = cas1active;
                        END IF;
END CASE;


-- CAS2 Control State Machine
CASE cas2sm IS
        WHEN cas2idle =>
                IF ( (!/ras & !/mux & !/lbe2 & !ref_cyc) # (!cpu_cyc & ref_req) ) THEN
                        cas2sm = cas2active;
                ELSE
                        cas2sm = cas2idle;
                END IF;
        WHEN cas2active =>
                        /cas2 = gnd;
                        IF ( (!ref_cyc) # (!/ras & ref_cyc) ) THEN
                                cas2sm = cas2idle;
                        ELSE
                                cas2sm = cas2active;
                        END IF;
END CASE;


-- CAS3 Control State Machine
CASE cas3sm IS
        WHEN cas3idle =>
                IF ( (!/ras & !/mux & !/lbe3 & !ref_cyc) # (!cpu_cyc & ref_req) ) THEN
                        cas3sm = cas3active;
                ELSE
                        cas3sm = cas3idle;
                END IF;
        WHEN cas3active =>
                /cas3 = gnd;
                IF ( (!ref_cyc) # (!/ras & ref_cyc) ) THEN
                        cas3sm = cas3idle;
                ELSE
                        cas3sm = cas3active;
                END IF;
END CASE;



-- DRDY Control State Machine
CASE drdysm IS
        WHEN drdyidle =>
                IF (!/ras & !/mux & !ref_cyc) THEN
                        drdysm = drdyactive;
                ELSE
                        drdysm = drdyidle;
                END IF;
        WHEN drdyactive =>
                /readydram = gnd;
                IF (!ref_cyc) THEN
                        drdysm = drdyidle;
                ELSE
```

```
                        drdysm = drdyactive;
                END IF;
END CASE;



-- CPU_CYC Control State Machine
CASE cpu_cycsm IS
        WHEN cpu_cycidle =>
                IF (!/csdram)  THEN
                        cpu_cycsm  = cpu_cycactive;
                ELSE
                        cpu_cycsm  = cpu_cycidle;
                END IF;
        WHEN cpu_cycactive =>
                cpu_cyc = vcc;
                IF ( (!/blast & !/readydram) ) THEN
                        cpu_cycsm  = cpu_cycidle;
                ELSE
                        cpu_cycsm  = cpu_cycactive;
                END IF;
END CASE;


-- REF_CYC Control State Machine
CASE ref_cycsm IS
        WHEN ref_cycidle =>
                IF (!cpu_cyc & ref_req)  THEN
                        ref_cycsm   = ref_cycactive;
                ELSE
                        ref_cycsm   = ref_cycidle;
                END IF;
        WHEN ref_cycactive =>
                ref_cyc = vcc;
                IF (/ras & !/mux)  THEN
                        ref_cycsm   = ref_cycidle;
                ELSE
                        ref_cycsm   = ref_cycactive;
                END IF;
END CASE;


-- REF_CYC Control State Machine
CASE ref_reqsm IS
        WHEN ref_reqidle =>
                IF (r4 & !hold_off)  THEN
                        ref_reqsm           = ref_reqactive;
                ELSE
                        ref_reqsm   = ref_reqidle;
                END IF;
        WHEN ref_reqactive =>
                ref_req = vcc;
                IF (ref_cyc)  THEN
                        ref_reqsm   = ref_reqidle;
                ELSE
                        ref_reqsm   = ref_reqactive;
                END IF;
END CASE;

-- HOLD_OFF Control State Machine
CASE hold_offsm IS
        WHEN hold_offidle =>
                IF (ref_cyc)  THEN
                        hold_offsm          = hold_offactive;
                ELSE
                        hold_offsm   = hold_offidle;
                END IF;
```

```
        WHEN hold_offactive =>
                hold_off = vcc;
                IF (!r4)  THEN
                        hold_offsm   = hold_offidle;
                ELSE
                        hold_offsm   = hold_offactive;
                END IF;
END CASE;



CASE addrsm IS
        WHEN addridle =>
                IF (/mux) THEN
                        --drama[1] = la[12];
                        --drama[0] = la[11];
                        drama[1] = la[13];
                        drama[0] = la[12];
                ELSE
                        drama[1] = la[3];
                        drama[0] = la[2];
                END IF;

                -- generate next addresses
                IF (!/readydram & /blast & !la[3] & !la[2]) THEN
                        addrsm = addrs1;
                ELSIF (!/readydram & /blast & la[3] & !la[2]) THEN
                        addrsm = addrs3;
                ELSE
                        addrsm = addridle;
                END IF;

        WHEN addrs1 =>
                -- generate 01
                drama[0] = vcc;
                drama[1] = gnd;
                IF (!/readydram & /blast)  THEN
                        addrsm = addrs2;
                ELSIF (!/readydram & !/blast) THEN
                        addrsm = addridle;
                ELSE
                        addrsm = addrs1;
                END IF;
        WHEN addrs2 =>
                -- generate 10
                drama[0] = gnd;
                drama[1] = vcc;
                IF (!/readydram & /blast)  THEN
                        addrsm = addrs3;
                ELSIF (!/readydram & !/blast) THEN
                        addrsm = addridle;
                ELSE
                        addrsm = addrs2;
                END IF;
        WHEN addrs3 =>
                -- generate 11
                drama[0] = vcc;
                drama[1] = vcc;
                IF (!/readydram & !/blast) THEN
                        addrsm = addridle;
                ELSE
                        addrsm = addrs3;
                END IF;

END CASE;
```

```
END;
```

```
---*************************************************************************************************************************
--*
--*                    LED
--*
--*************************************************************************************************************************

SUBDESIGN led
(
        user9080        : INPUT; -- User Out 9080

        failo           : OUTPUT; -- Fail LED
        9080debug       : OUTPUT; -- debug LED
)


--VARIABLE

BEGIN

9080debug = (!user9080);
failo = 9080debug;

END;
```

```
---*************************************************************************************************************************
--*
--*                    Memory dec., FLASH controller , UART controller
--*
--*************************************************************************************************************************

-- V830 internal RAM          :0000 0000h – 0FFF FFFFh
-- DRAM address               :1000 0000h – 1FFF FFFFh
-- SRAM address               :2000 0000h – 2FFF FFFFh
-- PCI9080 internal register  :3000 0000h – 3FFF FFFFh

-- Direct Master memory       : 8000 0000h - 8FFF FFFFh
-- Direct Master I/O          : A000 0000h - AFFF FFFFh
-- Squall module              : C000 0000h - CFFF FFFFh
-- UART                       : D000 0000h - DFFF FFFFh

-- V830 Build in RAM          : FE00 0000h - FE00 0FFFh
-- FLASH                      : FF00 0000h - FFFF FFFFh


SUBDESIGN memdec
(
        sm_clk              : INPUT;  -- state machine clock
        sm_rst              : INPUT;  -- state machine reset

        la[31..24]          : INPUT;  -- address bus
        /ads                : bi      ;  -- address strobe
        /blast              : bi      ;  -- burst last signal
        /lwr                : bi      ;  -- write/read~
        /wait               : bi      ;  -- wait for wait states
        /den                : bi      ;  -- data enable used with bus tranceiver
        /readysql           : INPUT;  -- ready from squall

        /cs9080             : OUTPUT; -- 9080 Registers Chip Select
        /csflash            : OUTPUT; -- FLASH Chip Select
        /csuart             : OUTPUT; -- UART Chip Select
        /cssql              : OUTPUT; -- Squall Chip Select
        /flashoe            : OUTPUT; -- FLASH Output Enable
        /flashwe            : OUTPUT; -- FLASH Write Enable
        /iobuf              : OUTPUT; -- I/O tranceiver Enable
        /uartr              : OUTPUT; -- UART read
        uartw               : OUTPUT; -- UART Write (active high)
        /readyflash         : OUTPUT; -- ready flash
        /readyuart          : OUTPUT; -- ready UART
        /ioads              : OUTPUT; -- UART ads

)


VARIABLE

CS9080          : NODE;
CSFLASH         : NODE;
CSSQL           : NODE;
CSUART          : NODE;

/cntclr          :NODE;
cnten            :NODE;
/iobufuart      : NODE;

count[8..0]     : DFFE;    --

csflashsm       : MACHINE
        OF BITS (q[4..0])
        WITH STATES (
                csflashidle         = b"00001",
```

```
                csflashactive       = b"00010",
                csflashwait         = b"00100",
                csflashactive2      = b"01000",
                csflashready        = b"10000");

csuartsm : MACHINE
        OF BITS (r[4..0])
        WITH STATES (
                csuartidle                  = b"00001",
                csuartactive        = b"00010",
                csuartactive1       = b"00100",
                csuartactive2       = b"01000",
                csuartactive3       = b"10000");


BEGIN

DEFAULTS
        /csflash   = vcc;
        /readyflash = vcc;
        /cntclr     = vcc;
        cnten       = vcc;
        /iobufuart = vcc;
        /readyuart = vcc;
        /ioads = vcc;
        /uartr = vcc;
        uartw = gnd;
END DEFAULTS;

csflashsm.clk = sm_clk;
csflashsm.reset = !sm_rst;

csuartsm.clk = sm_clk;
csuartsm.reset = !sm_rst;

count[].clk = sm_clk;
count[].clrn = !(!sm_rst # !/cntclr);
count[].ena = vcc;
count[] = count[] + 1;


---*************************************************************************************************************************
--*
--*                     Memory dec.
--*
--*************************************************************************************************************************

-- see dram.tdf file for dram decoding.

-- CSSRAM = (la[31..24] == H"20"); -- 0x20000000 -> 2FFF FFFFh
CS9080 = (la[31..24] == H"30"); -- 0x30000000 -> 3FFF FFFFh
-- Direct Master decodings are done inside PCI 9080.

CSSQL = (la[31..24] == H"C0");  -- 0xC0000000 -> CFFF FFFFh
CSUART = (la[31..24] == H"D0"); -- 0xD0000000 -> DFFF FFFFh
CSFLASH = (la[31..24] == H"FF"); -- 0xFF000000 -> FFFF FFFFh

/cs9080 = !(CS9080 & !/ads);
/cssql = dff( !( (CSSQL & !/ads) # (!/cssql & (/readysql)) ), sm_clk, , sm_rst );
/csuart = dff( !( (CSUART & !/ads) # (!/csuart & (/readyuart)) ), sm_clk, , sm_rst );
/csflash = !CSFLASH;

---*************************************************************************************************************************
--*
--*                     FLASH controller , UART controller
--*
--*************************************************************************************************************************
```

```
-- data tranceiver control line
/iobuf = !( (!/csflash # !/iobufuart) & !/den);

-- FLASH control lines
--/flashoe = !(!/csflash & !/lwr);
--/flashwe = !(!/csflash & /lwr & !/wait);
/flashoe = gnd;
/flashwe = vcc;

-- CSFLASH Control State Machine
-- 1 wait state using /ready
CASE csflashsm IS
        WHEN csflashidle =>
                /csflash = vcc;
                IF (CSFLASH & !/ads) THEN
                        csflashsm = csflashactive;
                ELSE
                        csflashsm = csflashidle;
                END IF;
        WHEN csflashactive =>
                IF (!/wait) THEN
                        csflashsm = csflashwait;
                ELSE
                        csflashsm = csflashactive2;
                END IF;
        WHEN csflashwait =>
                IF (!/wait) THEN
                        csflashsm = csflashwait;
                ELSE
                        csflashsm = csflashidle;
                END IF;
        WHEN csflashactive2 =>
                csflashsm = csflashready;
        WHEN csflashready =>
                /readyflash = gnd;
                IF (/blast # /wait) THEN
                        csflashsm = csflashidle;
                ELSE
                        csflashsm = csflashready;
                END IF;
END CASE;


-- CSUART Control State Machine
-- 1 wait state using /ready
CASE csuartsm IS
        WHEN csuartidle =>              -- UART is in Idle State
                /cntclr = gnd;
                /iobufuart = vcc;
                IF (!/csuart) THEN
                        csuartsm = csuartactive;
                ELSE
                        csuartsm = csuartidle;
                END IF;
        WHEN csuartactive =>            -- Continue to assert ADS
                /ioads = gnd;
                IF (count[] == 2)  THEN
                        csuartsm = csuartactive1;
                ELSE
                        csuartsm = csuartactive;
                END IF;
        WHEN csuartactive1 =>           -- Enable I/O buffer & r/w
                uartw = /lwr;
                /uartr = /lwr;
                IF  (count[] == 5) THEN
                        /iobufuart = gnd;
```

```
                        csuartsm = csuartactive2;
                ELSE
                        csuartsm = csuartactive1;
                END IF;
        WHEN csuartactive2 =>  -- wait again while enabled
                csuartsm = csuartactive3;
        WHEN csuartactive3 =>  -- assert ready
                /readyuart = gnd;
                csuartsm = csuartidle;
END CASE;



END;
```

```
---************************************************************************************************************************
--*
--*                    reset
--*
--************************************************************************************************************************

SUBDESIGN reset
(
        /lreseto9080        : INPUT; -- 9080 local bus reset out
        /por2               : INPUT; -- Power-on reset
        lclk                : INPUT;

        /lreset             : OUTPUT;
)

VARIABLE

count[8..0]                 : DFFE;    --

BEGIN

count[].clk = lclk;
count[].clrn = /por2;
count[] = count[] + 1;

/lreset = dffe(vcc, lclk, /por2 # lreseto9080, , (count[] == 30) );

END;
```

```
-- why there is a wait states for SRAM writes.

---*************************************************************************************************************
--*
--*                       SRAM
--*
--*************************************************************************************************************


SUBDESIGN sram
(
        sm_clk          : INPUT; -- the state machine's clock
        sm_rst          : INPUT; -- the state machine's reset
        /blast          : INPUT;  --
        /lwr            : bi    ; --
        srcy            : INPUT; --
        la[31..24]      : INPUT; -- address bus
        /ads            : bi    ; --
        /lbe[3..0]      : INPUT;

        /sramwe[3..0]   :OUTPUT; --
        /sramoe         : OUTPUT; --
        /srinc          : OUTPUT; --
        /readysram      : OUTPUT; --
        /cssram         : OUTPUT; --
)

VARIABLE
CSSRAM : NODE;

sramsm  : MACHINE
        OF BITS (a[4..0])
        WITH STATES (
                sramidle        = b"00001",
                sramreadwrite   = b"00010",
                sramread        = b"00100",
                sramwritewait   = b"01000",
                sramwrite       = b"10000");

BEGIN

DEFAULTS
        /readysram = vcc;
        /cssram = gnd;

END DEFAULTS;

sramsm.clk = sm_clk;
sramsm.reset = !sm_rst;

/srinc = gnd;
/sramoe = /lwr;
CSSRAM = (la[31..24] == H"20");  -- 0x20000000 -> 2fffffff
/cssram = dff( !( (CSSRAM & !/ads) # (!/cssram & /readysram) ), sm_clk, , sm_rst );

/sramwe0 = !(/lwr & !/lbe0);
/sramwe1 = !(/lwr & !/lbe1);
/sramwe2 = !(/lwr & !/lbe2);
/sramwe3 = !(/lwr & !/lbe3);


-- SRAM Control State Machine
-- 0 wait read, 1 wait write
CASE sramsm IS
        WHEN sramidle =>
```

```
                    /cssram = vcc;
                    IF (CSSRAM & !/ads)   THEN
                              sramsm = sramreadwrite;
                    ELSE
                              sramsm= sramidle;
                    END IF;
          WHEN sramreadwrite =>
                    /cssram = gnd;
                    IF (/lwr) THEN                          -- write
                              sramsm= sramwritewait;
                    ELSE                                    -- read
                              sramsm= sramread;
                    END IF;
          WHEN sramread =>
                              /readysram = gnd;
                              /cssram = gnd;
                              IF (!/blast)  THEN
                                        sramsm = sramidle;
                              ELSE
                                        sramsm = sramread;
                              END IF;
          WHEN sramwritewait =>
                              /cssram = gnd;
                              sramsm = sramwrite;
          WHEN sramwrite =>
                              /readysram = gnd;
                              /cssram = gnd;
                              IF (!/blast)  THEN
                                        sramsm = sramidle;
                              ELSE
                                        sramsm = sramwrite;
                              END IF;
END CASE;

END;
```

```
---*************************************************************************************************************************
--*
--*                    V830 to PCI 9080 interface.
--*
--*************************************************************************************************************************


SUBDESIGN v830-9080
(
        sm_clk          : INPUT; -- state machine clock
        sm_rst          : INPUT; -- state machine reset

        /st[3..0]       : INPUT;  -- status bits 3 to 0
        /lserr9080              : INPUT; -- local system error from 9080
        /wr830          ; INPUT; -- V830 write# and read
        readyo9080      : INPUT; -- ready to 9080
        /bcyst          : INPUT; -- ads from V830
        breqo9080       : INPUT; -- back-off-request-out from 9080

        /ready830               : OUTPUT; -- ready signal to V830
        /nmi830         : OUTPUT; -- nmi# to 830 for either back-off or local system error


        /ads            :bi     ; -- ads to local bus.
        /blast          ;bi     ; -- blast(burst last) for local bus.
        /lwr            ;bi     ;

)


VARIABLE

sm      : MACHINE
        OF BITS (q[6..0])
        WITH STATES
        (
                idle            = b"0000001",       -- V830 is default master
                single          = b"0000010",
                burst0          = b"0000100",
                burst1          = b"0001000",
                burst2          = b"0010000",
                burst3          = b"0100000",
                locked_dm       = b"1000000",
        );

BEGIN

DEFAULTS

        /ready830 = '1';
        /lock9080 = '1';
        /nmi830   = '1';
        /ads      = '1';
        /blast    = '1';
        /lwr      = '1';
        /lock9080 = '1';
        /nmi      = '1';
        sm = idle
END DEFAULTS;


sm.clk = sm_clk;
sm.reset = !sm_rst;


-- Unconditional back-off request or system error from PCI9080 to V830 when DEAD-LOCK ocurrs.
-- PCI 9080 will generate BREQo# signal when a possible dead-lock exists.
```

```
-- When the BREQo# or the LSERR# is asserted, this logic will assert NMI# to V830 (system errors)
-- Software needs to take care of these situations.

/nmi830 = !(/lserr9080 # breqo9080);


/lwr = !(/wr830);


-- Direct master or local bus access by V830.

CASE sm IS
        WHEN idle =>

                /ads = '1';
                /ready830 = '1';
                /blast = '1';
                IF (!/bcyst & (st[3..0]='0000' # st[3..0]='1000' # st[3..0]='1001')) THEN
                        /ads = '0';
                        sm = single;
                ELSEIF (!/bcyst & (st[3..2]='11')) THEN
                        /ads = '0';
                        sm = burst0;
                ELSEIF (!/bcyst & (st[3..0]='1011')) THEN
                        /ads = '0';
                        /lock9080 = '0';
                        sm = locked_dm;
                ELSE
                        sm = idle;
                END IF;


        WHEN single =>      --single R/W cycle
                /ads = '1';
                /blast = '0';
                IF (!(/readyo9080)) THEN
                        /ready830 = '0';
                        /blast = '1';
                        sm = idle;
                ELSE
                        sm = single;
                END IF;

        WHEN burst0 =>      -- Burst 4 mode(first data)
                /ads = '1';
                IF (!(/readyo9080)) THEN
                        /ready830 = '0'
                        sm = burst1;
                ELSE
                        sm = burst0;
                END IF;

        WHEN burst1 =>      -- Burst 4 mode(second data)
                IF (!(/readyo9080)) THEN
                        /ready830 = '0'
                        sm = burst2;
                ELSE
                        /ready830 = '1';
                        sm = burst1;
                END IF;

        WHEN burst2 =>      -- Burst 4 mode(3rd data)
                IF (!(/readyo9080)) THEN
                        /ready830 = '0'
                        sm = burst3;
                ELSE
                        /ready830 = '1';
```

```
                        sm = burst2;
                END IF;

        WHEN burst3 =>      -- Burst 4 mode(4th data)
                /blast = '0';
                IF (!(/readyo9080)) THEN
                        /ready830 = '0'
                        sm = idel;
                ELSE
                        /ready830 = '1';
                        sm = burst3;
                END IF;

        WHEN locked_dm =>           -- Locked cycle to PCI bus(must start with a read and end with a write)
                /ads = '1';
                /blast = '0';
                IF ((!/readyo9080) & (/wr830)) THEN
                        /ready830 = '0';
                        sm = idle;
                IF ((!/readyo9080) & (!/wr830)) THEN
                        /ready830 = '0';
                        /lock9080 = '1';
                        sm = idle;
                ELSE
                        sm = locked_dm


END CASE;

END;
```