

CHAPTER 1

1.1 INTRODUCTION	1-1
1.2 INSTALLING THE IQ80303 OR IQ80310 BOARD INTO THE BACKPLANE 80300BP	1-1
1.3 POWERING THE IQ80303/BACKPLANE 80300BP SYSTEM	1-1
1.4 66 MHZ_ENABLE (M66EN) PIN	1-1
1.5 INTERRUPT ROUTING AND IDSELS ON THE BACKPLANE 80300BP	1-1
1.6 POWER REQUIREMENTS	1-2
1.7 REFERENCE MANUAL	1-3
1.8 BILL OF MATERIAL	1-3

CHAPTER 2

2.1 PCI HOST DETECTION AND CONFIGURATION	2-1
2.2 PCI INITIALIZATION	2-1
2.3 PCI BIOS ROUTINES	2-1
2.4 ADDITIONAL MON960/CYGMON COMMANDS	2-6
2.5 PAL EQUATIONS.....	2-7
2.5.1 TITLE: '2825A'	2-7
2.5.2 TITLE: '2377A'	2-8

CONTENTS



LIST OF FIGURES

Figure 1-1. EVAL80303/80310 Backplane Board Illustration.....	1-2
---	-----

LIST OF TABLES

Table 1-1. PCI Interrupt Routing and IDSELs.....	1-2
Table 1-2. 80300BP Backplane Power Requirements	1-2

1.1 INTRODUCTION

The Backplane 80300BP board allows Intel's IQ80303 and IQ80310 board to be installed into a PCI socket and function as a host system board. The Backplane 80300BP contains three PCI slots, J2 through J4. The PCI backplane board contains arbitration for the three PCI slots, arbitration for 66MHz and 33MHz bus clock, an ATX power supply connector with stand by voltage power on circuit, a reset circuit, and three LEDs indicating +5V and +3.3V power on circuit and M66EN indicating 66MHz bus clock.

1.2 INSTALLING THE IQ80303 OR IQ80310 BOARD INTO THE BACKPLANE 80300BP

The IQ80303 or IQ80310 board must be installed into slot J2 of the backplane board with 8-pin header on the IQ80303 or IQ80310 board locking with the 8-pin header J1 on the backplane.

1.3 POWERING THE IQ80303/BACKPLANE 80300BP SYSTEM

The PCI backplane board is designed to be powered via an ATX power supply. The problem of 3.3V not being regulated was seen with some of the ATX power supplies and was solved by adding some load on +5V. We have seen this problem when using ATX power supply from TOP Microsystems. One of the companies we recommend to use for ATX power supply is Channel Well Technology Co. Ltd. ATX power supply, Model: CWT-300ATX-A (300W Max.), which does not need any load to be added on +5V.

Before inserting the power cable into the ATX power connector on the backplane board make sure that the +5VSB (stand-by voltage) is off. Inserting the power cable with +5VSB on may cause damage to the start up circuit and the backplane board may not power up. After the connector is installed, depress the momentary power switch to turn on the +5V and +3.3V rails on the ATX power supply.

1.4 66 MHz_ENABLE (M66EN) PIN

66MHz PCI clock generation circuitry is connected to M66EN to generate appropriate clock for the segment. If M66EN is asserted, clock is 66 MHz; if M66EN is deasserted, clock is 33 MHz.

1.5 INTERRUPT ROUTING AND IDSELS ON THE BACKPLANE 80300BP

The PCI slots J3 and J4 allow up to 2 peripheral boards to be installed on the system. These peripheral boards are configured by the IQ80303. The interrupt routing and IDSELS on the backplane board is shown in Table 1-1.

Table 1-1. PCI Interrupt Routing and IDSELS

SLOT	INTA	INTB	INTC	INTD	IDSEL
J2	S_INTA	S_INTB	S_INTC	S_INTD	PAD16
J3	S_INTC	S_INTD	S_INTA	S_INTB	PAD18
J4	S_INTD	S_INTA	S_INTB	S_INTC	PAD19

1.6 POWER REQUIREMENTS

The following table represents the power consumption of the 80300BP Backplane.

Table 1-2. 80300BP Backplane Power Requirements

Voltage	Current Typical	Current Maximum
+3.3V	0.21 Amps	0.29 Amps
+5V	0.14 Amps	0.19 Amps

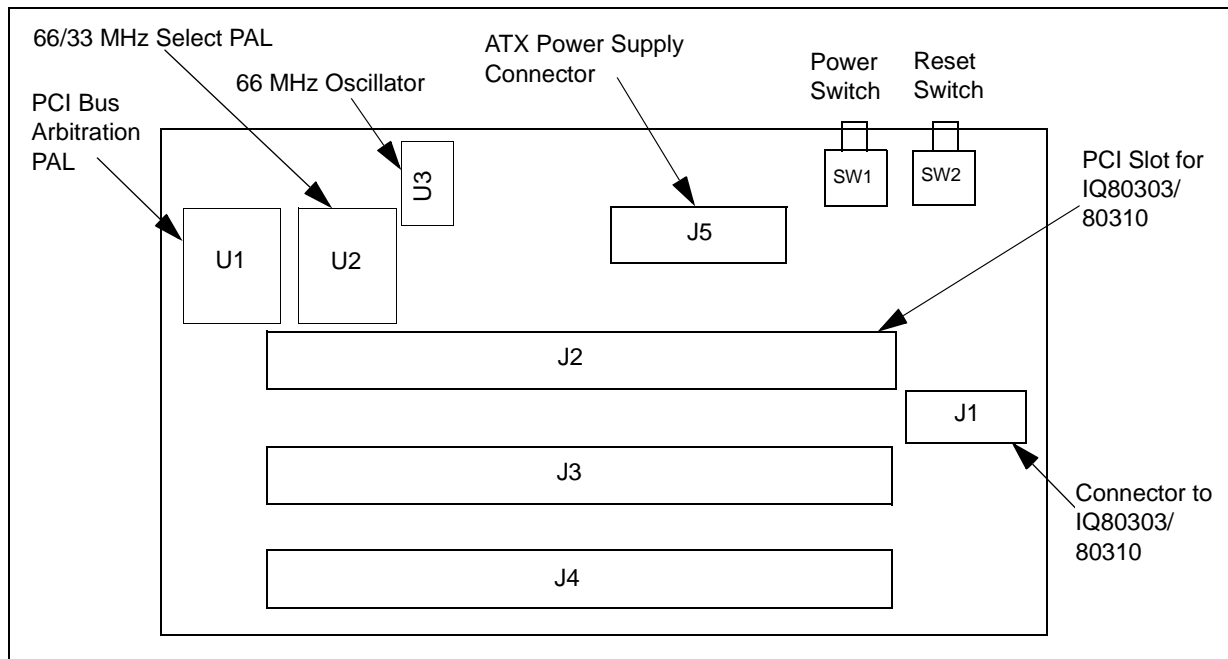


Figure 1-1. EVAL80303/80310 Backplane Board Illustration

1.7 REFERENCE MANUAL

PCI Local BIOS Specification, Revision 2.1
 PCI Special Interest Group
 2575 NE Kathryn Street #17
 Hillsboro, OR 97214
 (800) 433-5177 (U.S.)
 (503) 693-6232 (International)
 (503) 693-8344 (Fax)

1.8 BILL OF MATERIAL

Cyclone P/N	Reference Designator	Description	Qty	Mfg	Mfg Part Number
100-1522	U11	IC/SM 74LVC04A SOIC-14	1	Texas Instruments	SN74LVC04AD
100-1900	U7	IC/SM QS3245SO SOIC-20	1	Quick Switch	QS32455O
1005013	U8	IC/SM MAX6343 Reset SOT23-6	1	MAXIM	MAX6343RUT-T
101-2377-01	U1	PAL MACH211-7JC PLCC44	1	Lattice Vantis	MACH211-7JC
101-2825-01	U2	PAL M4LV-32/32-10JC PLCC44	1	Lattice Vantis	M4LV32/32-10JC
100-1698	U10	IC/SM 74HC74 SOIC-14	1	Texas Instruments	SN74HC74D
100-1699	U9	IC/SM 74HC14 SOIC	1	National Semi.	74HC14
150-6031	U3	OSC66.0MHZSMT3.3V	1	Ecliptek	EC2600TS-66-000M
150-8007	U4	CDC319/PI6C182 SSOP28	1	Texas Instrument	CDC319
110-3209	C16, C34	CAP TANT SM 47uf, 16V (7343)	2	AVX	TPSD476K016 R015
110-3214	C38	CAP TANT SM 10UF 25/35v (7343)	1	Sprague	293D1060025 D2T
110-3301	C7,C9-C13, C17,C20-C25, C28-C32	CAP CERM SM 0.01uf (0805)	18		
110-3304	C1-C6, C8, C14,C15,C18, C19,C26,C27, C33,C35-C37, C39	CAP CERM SM, 0.1uf (0805)	18		
126-1003-05	R19,R21	R/SM 1/10W 5% 1K ohm (0805)	2		

INTRODUCTION



126-1004-05	R23, R27	R/SM 1/10W 5% 10K ohm (0805)	2		
126-2703-05	R11	R/SM 1/10W 5% 2.7K ohm (0805)	1		
126-3301-05	R7	R/SM 1/10W 5% 33 ohm (0805)	1		
126-3302-05	R4,R5	R/SM 1/10W 5% 330 ohm (0805)	2		
126-4702-05	R1	R/SM 1/10W 5% 470 ohm (0805)	1		
126-4703-05	R12	R/SM 1/10W 5% 4.7K ohm (0805)	1		
127-0000-05	R28	R/SM 1/8W 5% 0K ohm (1206)			
129-2105	R2,R6,R8,R9, R14,R15,R16, R17,R18,R20, R22,R24,R25, R26	Resistor Pk SM RNC4R8P 4.7kohm	14	CTS	742083472JTR
129-2113	R10	Resistor Pk SM RNC4R8P 33 ohm	1	CTS	742083330JTR
130-1462	J2	CONNPCI_64 3.3V 188 PCI	3	AMP	145168-4
130-1462	J3	CONNPCI_64 3.3V 188 PCI	3	AMP	145168-4
130-1462	J4	CONNPCI_64 3.3V 188 PCI	3	AMP	145168-4
130-1519	J5	CONN PCPWR Male/Mini-Fit/20p	1	Molex	39-29-9202
130-1621	Z1,Z2	Jumper JUMP2X1	2	Molex	22-28-4023
130-1765	J1	8Pin Header CONN 2X4	1	Samtech	DW-04-12-T-D-550
150-4000	SW1,SW2	Switch, SW-PUSH 3P (Right Ang)	2	C&K Switch	EP12SD1ABE
150-9100	CR1,CR2, CR3	LED Green	3	Hewlett Packard	HLMP-3507\$010
160-1108	CR5	Diode SM BAS16	1	National Semi.	BAS16
230-1197		SCREW-Nylon 4/40 x 3/8" Pan Slot	4	McMaster Carr	95000A108
230-1507		HARDWARE 4-40 x 3/8" Standoff	4	Keystone	1902B (0.375")
270-0017-02		3 Slot PCI Backplane Printed Circuit Board	1		

2.1 PCI HOST DETECTION AND CONFIGURATION

When installed in the slot J2 of the backplane, the IQ80303/80310 acts as the PCI host in the system. The IQ80303/80310 determines whether it is in the host state by the Backplane detect bit carried on J1 of the backplane. Upon system startup, MON960/CygMon uses the backplane detect bit to determine whether it needs to initialize the primary PCI bus. If the IQ80303/80310 is the PCI host, the ATU base address (PIABAR) is set to the base of DRAM (A000 0000h), and the master enable bit in the ATU command register (PATUCMD) is set.

2.2 PCI INITIALIZATION

When acting as a PCI host, MON960/CygMon extensions are responsible for initializing the devices on the PCI bus of the Backplane 80300BP. PCI initialization involves allocating address spaces (Memory, Memory Mapped I/O, and I/O), assigning PCI base addresses, assigning IRQ values (see Table D-1), and enabling PCI mastership. Devices containing PCI-to-PCI bridges and hierarchical buses are not supported.

2.3 PCI BIOS ROUTINES

The supported PCIBIOS functions are described in the subsections that follow.

```
pci_bios_present()
find_pci_device()
find_pci_class_code()
generate_special_cycle()
read_config_byte()
read_config_word()
read_config_dword()
write_config_byte()
write_config_word()
write_config_dword()
get_irq_routing_options()
set_pci_irq()
```

Although the calling interface is different from that used on a DOS-based host, these functions preserve, as closely as possible, the parameters and return values described in *PCI Local Bus Specification*, Revision 2.1. Functions that return multiple values do so by filling in the fields of a structure passed by the calling routine.

pci_bios_present

This function allows the caller to determine whether the PCI BIOS interface function set is present, and the current interface version level. It also provides information about the hardware mechanism used for accessing configuration space and whether or not the hardware supports generation of PCI Special Cycles.

Calling convention:

```
int pci_bios_present (  
    PCI_BIOS_INFO *info  
);
```

Return values:

This function always returns SUCCESSFUL.

find_pci_device

This function returns the location of PCI devices that have a specific Device ID and Vendor ID. Given a Vendor ID, a Device ID, and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Vendor ID and Device ID match the input parameters.

Calling software can find all devices having the same Vendor ID and Device ID by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND. A return value of BAD_VENDOR_ID indicates that the Vendor ID value passed had a value of all “1”s.

Calling convention:

```
int find_pci_device (  
    int    device_id,  
    int    vendor_id,  
    int    index  
);
```

Return values:

This function returns SUCCESSFUL if the indicated device is located, DEVICE_NOT_FOUND if the indicated device cannot be located, or BAD_VENDOR_ID if the vendor_id value is illegal.

find_pci_class_code

This function returns the location of PCI devices that have a specific Class Code. Given a Class Code and an Index, the function returns the Bus Number, Device Number, and Function Number of the Nth Device/Function whose Class Code matches the input parameters.

Calling software can find all devices having the same Class Code by making successive calls to this function starting with the index set to “0”, and incrementing the index until the function returns DEVICE_NOT_FOUND.

Calling convention:

```
int find_pci_class_code (  
    int class_code,  
    int index
```

Return values:

This function returns SUCCESSFUL if the indicated device is located or DEVICE_NOT_FOUND if the indicated device cannot be located.

generate_special_cycle

This function allows for generation of PCI Special Cycles. The generated special cycle is broadcast on a specific PCI Bus in the system.

PCI Special Cycles are not supported on the 80300BP backplane PCI bus.

Calling convention:

```
int generate_special_cycle (  
    int bus_number,  
    int special_cycle_data  
);
```

Return values:

Since PCI Special Cycles are not supported by the 80300BP backplane, this function always returns FUNC_NOT_SUPPORTED.

read_config_byte

This function allows the caller to read individual bytes from the configuration space of a specific device.

Calling convention:

```
int read_config_byte (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,1,2,...,255 */  
    UINT8 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated byte was read correctly or ERROR if there is a problem with the parameters.

read_config_word

This function allows the caller to read individual shorts (16 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to "0").

Calling convention:

```
int read_config_word (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,2,4,...,254 */  
    UINT16 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated word was read correctly or ERROR if there is a problem with the parameters.

read_config_dword

This function allows the caller to read individual longs (32 bits) from the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to "0").

Calling convention:

```
int read_config_dword (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,4,8,...,252 */  
    UINT32 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated long was read correctly or ERROR if there is a problem with the parameters.

write_config_byte

This function allows the caller to write individual bytes to the configuration space of a specific device.

Calling convention:

```
int write_config_byte (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,1,2,...,255 */  
    UINT8 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated byte was written correctly or ERROR if there is a problem with the parameters.

write_config_word

This function allows the caller to write individual shorts (16 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of two (i.e., bit 0 must be set to "0").

Calling convention:

```
int write_config_word (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,2,4,...,254 */  
    UINT16 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated word was written correctly or ERROR if there is a problem with the parameters.

write_config_dword

This function allows the caller to write individual longs (32 bits) to the configuration space of a specific device. The Register Number parameter must be a multiple of four (i.e., bits 0 and 1 must be set to "0").

Calling convention:

```
int write_config_dword (  
    int bus_number,  
    int device_number,  
    int function_number,  
    int register_number, /* 0,4,8,...,252 */  
    UINT32 *data  
);
```

Return values:

This function returns SUCCESSFUL if the indicated long was written correctly or ERROR if there is a problem with the parameters.

get_irq_routing_options

This routine returns the PCI interrupt routing options available on the 80300BP backplane. The PCI Interrupt routing fabric on the 80300BP backplane is not reconfigurable (fixed mapping relationships).

Calling convention:

```
int get_irq_routing_options (  
    PCI_IRQ_ROUTING_TABLE *table  
);
```

Return values:

This function always returns FUNC_NOT_SUPPORTED.

set_pci_irq

The PCI Interrupt routing fabric on the 80300BP backplane is not reconfigurable (fixed mapping relationships); therefore, this function is not supported.

Calling convention:

```
int set_pci_irq (  
    int int_pin,  
    int irq_num,  
    int bus_dev  
);
```

Return values:

This function always returns FUNC_NOT_SUPPORTED.

2.4 ADDITIONAL MON960/CYGMON COMMANDS

The following commands have been added to the UI interface of MON960 to support the 80300BP backplane.

print_pci Utility

A `print_pci` command to MON960 is accessed through the MON960 command prompt. This command displays the contents of the PCI configuration space on a selected adapter on the primary PCI interface or on the i960 VH processor itself. For more information on the meaning of the fields in PCI configuration space, refer to the *PCI Local Bus Specification* Revision 2.1. The syntax of this command is:

```
pp <bus number> <device number> <function number>
```

2.5 PAL EQUATIONS

2.5.1 TITLE: '2825A'

PATTERN 101-2825-01 to be installed on 80300BP
 REVISION A
 AUTHOR Tweety
 COMPANY Cyclone Microsystems
 CHIP M4A3-32/32 10JC

COMPILER ISP DESIGN EXPERT

"-----

"----- INPUT PINS -----

XCLK	pin 11	;"66MHz Osc. clock for divide by 2 counter
PRSTn	pin 2	;"System reset
PM66EN	pin 3	;"66MHz enable pin

"----- OUTPUT PINS -----

PCLK	pin 26	;"Osc. clock output->System clock
REQ64n	pin 19	ISTYPE'REG_D' ;"64-Bit request

"----- BURIED NODES -----

C0	node	ISTYPE'REG_D' ;"Divide by 2 counter
----	------	-------------------------------------

"-----

EQUATIONS

C0.CLK = XCLK;
 REQ64n.CLK = XCLK;

C0.SET = 0;
 REQ64n.SET = 0;

C0.CLR = 0;
 REQ64n.CLR = 0;

"-----

```

C0 := C0+1;                               Divide by 2 counter

PCLK = C0 & !PM66EN                         "66MHz OR 33MHz selection
  # XCLK & PM66EN;

!REQ64n := !PRSTn;                          REQ64n asserted when RESETn asserted ->64-bit

REQ64n.OE = !PRSTn;

```

"-----"

END

2.5.2 TITLE: '2377A'

```

PATTERN          101-2377-01 to be installed on 80300BP
REVISION         A
AUTHOR          Sylvester
COMPANY         Cyclone Microsystems

```

CHIP MACH211 - 7JC

;** INPUT PINS **;

```

pin      35 CLK                ; 33MHz system clk
pin      10 HREQn              ; Host (PCI9060) bus request
pin      11 AREQn              ; PCI expansion slot "A" bus request
pin      13 BREQn              ; PCI expansion slot "B" bus request
pin      32 CREQn              ; PCI expansion slot "C" bus request
pin      33 DREQn              ; PCI expansion slot "D" bus request
pin      41 FRAMEn             ; PCI bus FRAME# signal
pin      3 TRDYn               ; PCI bus target ready signal
pin      43 IRDYn              ; PCI bus initiator ready signal
pin      2 STOPn               ; PCI bus STOP# signal
pin      4 LOCKn               ; PCI bus LOCK# signal
pin      42 ASYNCRSTn          ; system reset, asynchronous, active low

```

;** OUTPUT PINS **;

```

pin 20 HGNTn                REG    ; Host (PCI9060) bus grant
pin 19 AGNTn                REG    ; PCI expansion slot "A" bus grant
pin 18 BGNTn                REG    ; PCI expansion slot "B" bus grant
pin 17 CGNTn                REG    ; PCI expansion slot "C" bus grant
pin 21 DGNTn                REG    ; PCI expansion slot "D" bus grant

pin 39 DUMMY1              COMB    ; dummy output to use TRDY, LOCK and STOP

```

```

; ** BURIED SIGNALS **;
NODE 3  P0      REG      ; priority state variable
NODE 7  P1      REG      ; priority state variable

NODE 11 P2      REG      ; priority state variable
NODE 15 P3      REG      ; priority state variable

NODE 6  B0      REG      ; PCI bus state variable
NODE 10 B1      REG      ; PCI bus state variable

NODE 34 T0      REG      ; time-out counter bit
NODE 36 T1      REG      ; time-out counter bit
NODE 38 T2      REG      ; time-out counter bit
NODE 49 T3      REG      ; time-out counter bit

NODE 37 TIMEOUT REG      ; time-out node
NODE 53 RST1n   REG      ; reset synchronizing register
NODE 54 RESETn  REG      ; synchronous reset

NODE 14 AGENTQUIT COMB    ; granted bus agent deasserts bus request

STRING  sH      `( /P3 * /P2 * /P1 * /P0 )`
STRING  sA      `( /P3 * /P2 * /P1 * P0 )`
STRING  sB      `( /P3 * /P2 * P1 * /P0 )`
STRING  sC      `( /P3 * P2 * /P1 * /P0 )`
STRING  sD      `( P3 * /P2 * /P1 * /P0 )`

STRING  sGRANTED `( /B1 * /B0 )`
STRING  sPARKED  `( B1 * B0 )`
STRING  sOFF     `( /B1 * B0 )`
STRING  sACTIVE  `( B1 * /B0 )`

EQUATIONS

HGNTn.CLKF = CLK      HGNTn.SETF = GND
AGNTn.CLKF = CLK      AGNTn.SETF = GND
BGNTn.CLKF = CLK      BGNTn.SETF = GND
CGNTn.CLKF = CLK      CGNTn.SETF = GND
DGNTn.CLKF = CLK      DGNTn.SETF = GND

P0.CLKF = CLK      P0.RSTF = /ASYNCRSTn
P1.CLKF = CLK      P1.RSTF = /ASYNCRSTn
P2.CLKF = CLK      P2.RSTF = /ASYNCRSTn
P3.CLKF = CLK      P3.RSTF = /ASYNCRSTn

B0.CLKF = CLK      B0.SETF = GND
B1.CLKF = CLK      B1.SETF = GND

```

```
T0.CLKF = CLK      T0.SETF = GND
T1.CLKF = CLK      T1.SETF = GND
T2.CLKF = CLK      T2.SETF = GND
T3.CLKF = CLK      T3.SETF = GND
```

```
RST1n.CLKF = CLK   RST1n.SETF = GND
RESETn.CLKF = CLK  RESETn.SETF = GND
```

```
TIMEOUT.CLKF = CLK  TIMEOUT.SETF = GND
```

```
;*****;
```

```
;** KEEP UNUSED INPUTS IN FILE **;
```

```
;*****;
```

```
DUMMY1 = TRDYn * STOPn * LOCKn
```

```
;*****;
```

```
;** SYNCHRONOUS RESET GENERATION **;
```

```
;*****;
```

```
RST1n  := ASYNCRSTn
RESETn := RST1n
```

```
;*****;
```

```
;** BUS GRANT TIMEOUT EQUATIONS **;
```

```
;*****;
```

```
T0 := RESETn * sGRANTED* IRDYn * /T0
T1 := RESETn * sGRANTED* IRDYn * /T1 * T0
    + RESETn * sGRANTED* IRDYn * T1 * /T0
T2 := RESETn * sGRANTED* IRDYn * /T2 * T1 * T0
    + RESETn * sGRANTED* IRDYn * T2 * /T1 * /T0
    + RESETn * sGRANTED* IRDYn * T2 * /T1 * T0
    + RESETn * sGRANTED* IRDYn * T2 * T1 * /T0
T3 := RESETn * sGRANTED* IRDYn * /T3 * T2 * T1 * T0
    + RESETn * sGRANTED* IRDYn * T3 * /T2 * /T1 * /T0
    + RESETn * sGRANTED* IRDYn * T3 * /T2 * /T1 * T0
    + RESETn * sGRANTED* IRDYn * T3 * /T2 * T1 * /T0
    + RESETn * sGRANTED* IRDYn * T3 * /T2 * T1 * T0
    + RESETn * sGRANTED* IRDYn * T3 * T2 * /T1 * /T0
    + RESETn * sGRANTED* IRDYn * T3 * T2 * /T1 * T0
    + RESETn * sGRANTED* IRDYn * T3 * T2 * T1 * /T0
```

```
TIMEOUT := T3 * T2 * T1 * T0
```

```
AGENTQUIT = /HGNTn * HREQn
           + /AGNTn * AREQn
           + /BGNTn * BREQn
           + /CGNTn * CREQn
           + /DGNTn * DREQn
```

```
;*****;
```

```
;** BUS GRANT EQUATIONS **;
```

```
;*****;
```

```
/HGNTn := sH * sGRANTED * /TIMEOUT
        + sH * sACTIVE
        + sH * sPARKED
/AGNTn := sA * sGRANTED * /TIMEOUT
        + sA * sACTIVE
        + sA * sPARKED
/BGNTn := sB * sGRANTED * /TIMEOUT
        + sB * sACTIVE
        + sB * sPARKED
/CGNTn := sC * sGRANTED * /TIMEOUT
        + sC * sACTIVE
        + sC * sPARKED
/DGNTn := sD * sGRANTED * /TIMEOUT
        + sD * sACTIVE
        + sD * sPARKED
```

```
;*****;
```

```
;** PCI BUS STATE MACHINE **;
```

```
;*****;
```

```
;**
```

```
;** STATE TRANSITIONS
```

```
;**
```

```
;** sPARKED := (any REQn=0)-> sOFF
;**           + (all REQn=1) -> sPARKED
```

```
;**
```

```
;** sOFF := (FRAMEn=1 & IRDYn=1)-> sGRANTED
;**           + (FRAMEn=0) -> sACTIVE
;**           + (FRAMEn=1 & IRDYn=0) -> sACTIVE
```

```
;**
```

```
;** sGRANTED := (FRAMEn=0 & ANY GNT=0) -> sOFF
;**           + (FRAMEn=0 & ALL GNT=1) -> sACTIVE
;**           + (FRAMEn=1 & TIMEOUT=0) -> sGRANTED
;**           + (FRAMEn=1 & TIMEOUT=1) -> sPARKED
;**           + (AGENTQUIT=1) & ALL REQ=1 -> sPARKED
;**           + (AGENTQUIT=1) & ANY REQ=0 -> sOFF
```

```

,**
,**
,**      sACTIVE := (any REQn=0 & FRAMEn=1)      -> sGRANTED
,**      + (all REQn=1 & FRAMEn=1)              -> sPARKED
,**      + (FRAMEn=0)                            -> sACTIVE
,**
,**
B0 := sPARKED * AREQn * BREQn * CREQn * DREQn * HREQn
      + sPARKED * (/AREQn + /BREQn + /CREQn + /DREQn + /HREQn)
      + sGRANTED * /FRAMEn * (/AGNTn + /BGNTn + /CGNTn + /DGNTn + /HGNTn)
      + sGRANTED * FRAMEn * TIMEOUT
      + sGRANTED * AGENTQUIT
      + sACTIVE * FRAMEn * AREQn * BREQn * CREQn * DREQn * HREQn
      + /RESETn

B1 := sPARKED * AREQn * BREQn * CREQn * DREQn * HREQn
      + sOFF * /FRAMEn
      + sOFF * FRAMEn * /IRDYn
      + sGRANTED * FRAMEn * TIMEOUT
      + sGRANTED * /FRAMEn * AGNTn * BGNTn * CGNTn * DGNTn * HGNTn
      + sGRANTED * AGENTQUIT * AREQn * BREQn * CREQn * DREQn * HREQn
      + sACTIVE * /FRAMEn
      + sACTIVE * FRAMEn * AREQn * BREQn * CREQn * DREQn * HREQn
      + /RESETn

;*****;

,** PRIORITY STATE MACHINE **;

;*****;

STATE
MOORE_MACHINE
DEFAULT_BRANCH HOLD_STATE

,** STATE ASSIGNMENT EQUATIONS **;

H = /P3 * /P2 * /P1 * /P0 ; This state assignment results
A = /P3 * /P2 * /P1 * P0  ; in 11 product terms per variable.
B = /P3 * /P2 * P1 * /P0  ; The conventional assignment
C = /P3 * P2 * /P1 * /P0  ; results in 14 or more product
D = P3 * /P2 * /P1 * /P0  ; terms for some variables.

,** STATE TRANSITION EQUATIONS **;
      H := H_TO_A -> A
           + H_TO_B -> B
           + H_TO_C -> C
           + H_TO_D -> D

```

A := A_TO_B -> B
 + A_TO_C -> C
 + A_TO_D -> D
 + A_TO_H -> H

B := B_TO_C -> C
 + B_TO_D -> D
 + B_TO_H -> H
 + B_TO_A -> A

C := C_TO_D -> D
 + C_TO_H -> H
 + C_TO_A -> A
 + C_TO_B -> B

D := D_TO_H -> H
 + D_TO_A -> A
 + D_TO_B -> B
 + D_TO_C -> C

CONDITIONS

;** STATE CONDITION EQUATIONS **;

H_TO_A = /AREQn * sOFF

H_TO_B = AREQn * /BREQn * sOFF

H_TO_C = AREQn * BREQn * /CREQn * sOFF

H_TO_D = AREQn * BREQn * CREQn * /DREQn * sOFF

A_TO_B = /BREQn * sOFF

A_TO_C = BREQn * /CREQn * sOFF

A_TO_D = BREQn * CREQn * /DREQn * sOFF

A_TO_H = BREQn * CREQn * DREQn * /HREQn * sOFF

B_TO_C = /CREQn * sOFF

B_TO_D = CREQn * /DREQn * sOFF

B_TO_H = CREQn * DREQn * /HREQn * sOFF

B_TO_A = CREQn * DREQn * HREQn * /AREQn * sOFF

SOFTWARE



$C_TO_D = \overline{DREQn} * sOFF$

$C_TO_H = DREQn * \overline{HREQn} * sOFF$

$C_TO_A = DREQn * HREQn * \overline{AREQn} * sOFF$

$C_TO_B = DREQn * HREQn * AREQn * \overline{BREQn} * sOFF$

$D_TO_H = \overline{HREQn} * sOFF$

$D_TO_A = HREQn * \overline{AREQn} * sOFF$

$D_TO_B = HREQn * AREQn * \overline{BREQn} * sOFF$

$D_TO_C = HREQn * AREQn * BREQn * \overline{CREQn} * sOFF$